WorldModel^2

Project: AGI

Alexander Epple AE ,Jonas Lang JL ,Thilo Dünßer TD ,Yannik Bretschneider YB

April 7, 2026

Abstract

Developing agents that generalize beyond previously seen tasks is a major challenge in Artifcial General Intelligence (AGI). In this work, we investigate whether exploration behavior itself can be learned and transferred across tasks. To do so, we introduce a switchboard environment with procedurally generated logical and temporal rules, enabling a large and diverse distribution of solvable tasks. To ensure consistency, all generated rules are validated using a symbolic ground-truth solver.

On this basis, we investigate three questions: (i) how DreamerV3-based agents compare to model-free reinforcement learning baselines, (ii) whether a Dreamer agent can be adapted to a continual self-learning setting in which it must infer new rules repeatedly, and (iii) how curriculum learning affects adaptation to increasing rule complexity. To support these experiments, we develop a tree-based rule generator, a GPU-accelerated switchboard simulator, and complexity scorers for rule filtering and curriculum design.

Our results show that Dreamer-based agents outperform the baselines on structured but challenging rule sets: on hard rules, Vanilla Dreamer (VD) reaches a success rate of 0.71 and Energy Dreamer (ED) 0.69, compared to 0.57 for the strongest baseline, Contrastive Reinforcement Learning (CRL). In contrast, all methods perform substantially worse on highly diverse random rules, suggesting that rule variability and sparse feasible solutions remain difficult. In the self-learning setting, Dreamer shows partial adaptation on direct-mapping tasks, with the smallest models reaching single-slot success rates of about 60%, while larger models are less stable and often perform worse despite higher reward. In the curriculum setting, we observe both negative transfer and catastrophic forgetting between early levels, but later checkpoints recover performance on previously seen levels and consistently outperform an untrained baseline on more complex ones.

Overall, the results provide partial evidence that transferable exploration can be learned in procedurally generated environments, while also highlighting that stable world-model learning under changing task rules remains a major challenge.

---

1 Introduction

AGI most commonly refers to AI systems that are at least as capable as humans across most tasks [19]. While some suggest that current Large Language Models (LLMs) already meet this bar [2], others claim that these approaches have fundamental limitations that prevent them from achieving the kind of learning abilities observed in animals and humans.

Schölkopf et al. [23] argues that although most Machine Learning (ML) algorithms can generalize between data points within a given domain, they remain specialized to particular tasks and cannot generalize across fundamentally different problems. This is because such algorithms disregard real-world interventions and domain shifts, which are both necessary for a causal understanding of the world, and in turn for out-of-distribution generalization.

Ha and Schmidhuber [10] and LeCun et al. [17] make a similar argument, proposing that animals and humans construct internal mental models of their environments, which enable far faster learning and appropriate responses to previously unseen situations.

We argue that the ability to generalize across novel situations requires more than statistical pattern matching over observed data — it demands active interaction with an environment and the construction of causal world models. To evaluate this, we apply DreamerV3 [14] to a switchboard-based Reinforcement Learning (RL) task. DreamerV3 is a Model-Based Reinforcement Learning (MBRL) algorithm that learns an internal model of its environment by imagining plausible future scenarios, which an Actor-Critic (AC) network then uses to select actions.

DreamerV3 has shown State-of-the-Art Performance across a wide variety of benchmark tasks. Most notably, it was the first algorithm to collect diamonds in Minecraft from scratch, without any human demonstrations or hand-crafted curricula [14]. This was considered a long-standing challenge in AI, as it requires navigating sparse rewards and long time horizons.

In this project, we adapt DreamerV3 to the switchboard environment, evaluate its performance, and compare it against several baselines. Beyond this, we introduce a — to our knowledge — novel extension that uses DreamerV3 as a meta-reinforcement learner: instead of training an agent to solve a fixed task, we train it to dynamically adapt to new tasks defined by explicit rules. This is motivated by the idea that a general agent must not only be able to learn, but to learn how to learn to be able to adapt to general environments. We evaluate this on environments with rules that

directly map switches to observations and look at how prior training influences the model's capability to learn new rulesets. We also explore how curriculum learning could scale this approach to switchboards with more complex rule structures, and present preliminary findings on how gradually introducing new concepts affects performance.
To support these experiments, we built a range of supporting infrastructure: a random rule generator based on tree-structured representations, a GPU-accelerated switchboard simulation for faster training, and a complexity scoring system that quantifies rule difficulty and is used to filters outlier rules during curriculum learning to stabilize training. Our main contributions are:

- **Infrastructure:** We develop a tree-based rule generator, GPU-accelerated switchboard execution, and a complexity scoring system for evaluation and training stabilization.
- **Switchboard Benchmark:** We adapt an implementation of the DreamerV3 architecture to the **Switchboard** environment and compare it against multiple other RL techniques as baselines.
- **Meta-Reinforcement Learning:** We introduce a — to our knowledge — novel use of DreamerV3 as a meta-reinforcement learner, training the agent to adapt to tasks defined by explicit rules.
- **Curriculum Learning:** We implement a curriculum learning framework for the Switchboard meta-learner and evaluate how increasing rule complexity affects performance.

1.1 Outline

The remainder of this paper is organized as follows: Section 2 provides the theoretical foundation for world models and discusses related work in autonomous exploration. In Section 3, we detail our methodological framework, including the procedural generation of rules (Section 3.2), complexity scoring (Section 3.4), Dreamer architecture adaptations for the switchboard (Section 3.5), and Curriculum Learning (Section 3.6). Section 4 presents our experimental results across baseline comparison, Dreamer self-learning and curriculum learning setups. Finally, Section 5 discusses our findings and their implications.

2 Background & Related Work

AGI is commonly associated with the ability to achieve goals across a wide range of environments rather than only on fixed benchmarks [18, 22]. As stated in Sutton's "Bitter Lesson", agents should be able to discover new concepts like humans, not just replicate what has already been discovered [27]. This makes two capabilities appear especially important: Learning internal models of the environment and exploring efficiently when faced with unfamiliar situations. World-model approaches address the first requirement by learning latent environment dynamics that support prediction, planning and policy improvement [10, 12, 14]. The second requirement is motivated by the difficulty of sparse reward RL, where progress often depends on gathering useful information through exploration rather than raw optimization alone [21, 8]. Ideally, a truly intelligent system has broad generalization capabilities to tasks unknown to both it and its designer [3]. To this end, we study these ideas in a controlled switchboard environment with procedurally generated logical and temporal rules. More specifically, we ask whether a world-model agent can learn exploratory behavior that transfers to previously unseen tasks, in the spirit of procedural generalization benchmarks [4]. Our work builds on a MBRL architecture, namely DreamerV3 [14], and focuses on learning to explore procedurally generated tasks. We highlight relevant prior work in these directions and provide a brief overview of the architecture.

2.1 World Models and Model-Based Reinforcement Learning (MBRL)

World models are based on the idea that an agent can improve decision making by learning an internal predictive model of its environment rather than relying on trial-and-error alone. Instead of directly mapping observations to actions, the agent learns latent dynamics that capture how the environment evolves over time. This enables prediction, planning, and more structured policy improvement, and has proven especially useful in settings where rewards are sparse [10, 12, 11, 24, 15].
MBRL is particularly interesting for AGI-motivated research because internal models appear to be an important prerequisite for adaptation. An agent that can model the temporal and causal relations of its environment is better suited to reason about the actions it can take, plan and form strategies for unseen situations. Among recent MBRL approaches, the Dreamer family of models is especially relevant, because it combines latent dynamics learning with imagined trajectories. Instead of using the raw observations directly, it learns a compact Recurrent State Space Model (RSSM) and uses rollouts in latent space to train value and policy networks [11, 13, 14]. This architecture is also described in more detail in Section 2.3. Other approaches such as *MuZero* and *TD-MPC2* similarly show that

learned latent models can support planning-based control [24, 15]. For our project, Dreamer is a natural backbone because its learned latent rollouts provide an internal simulation mechanism that may support more structured and transferable exploration in procedurally generated tasks.
The connection between Dreamer-style world models and exploration has already been examined in prior work. *Plan2Explore* uses self-supervised world models to seek informative experience and shows that model-based exploration can support fast adaptation to new tasks [25]. More recently, *Curious Replay* improves upon experience replay by prioritizing prior experiences based on curiosity and shows this improves model-based adaptation under environmental change [16]. Another recent work, *Enter the Void*, suggests that world-model-based agents can drive exploration through high-entropy planning, which reinforces the idea that exploration in MBRL can be more structured and deliberate rather than purely reactive [26].

2.2 Exploration, Generalization, and Procedural Tasks

Efficient exploration is one of the main challenges in RL, especially when reward is sparse or delayed. In such environments, progress often depends on the ability to gather information that is useful for later goal achievement, thus motivating a large amount of work on curiosity-driven learning. Agents are often rewarded for novelty or uncertainty, rather than immediate success [20, 21, 1]. Beyond novelty-based curiosity, there has also been work done in the direction of goal-directed skill discovery. In *IMGEP*, for example, agents self-organize exploration by selecting goals and curricula. This frames exploration around learning progress rather than random search [9, 6]. It has also been argued that difficult exploration requires even more structured behavior, such as revisiting promising states and exploring outward systematically rather than relying on randomness alone [8].
Exploration is closely related to generalization. An agent that performs well only on a fixed set of tasks may simply memorize useful patterns, whereas a more general agent should adapt its behavior when faced with unseen situations. Procedurally generated environments expose agents to a broad distribution of related tasks while reducing the risk of overfitting [5, 4]. Our switchboard environment is designed with this motivation in mind, by training on procedurally generated rules rather than hand-crafted single objectives. The agent is encouraged to learn effective exploration strategies and infer task structure, rather than memorizing the paths to reward.

2.3 Dreamer Architecture

Dreamer is a MBRL method that consists of a learned world model and an AC policy. Specifically, it learns an RSSM that maps observations into a latent space, predicts future latent states and rewards, and improves its policy through imagined rollouts in latent space rather than only by interacting with the environment directly [14].
It alternates between three stages, namely: Learning the latent dynamics model from replayed experience, imagining trajectories in latent space, and improving actor and critic networks based on these trajectories. The latent RSSM h_t and stochastic latent state z_t evolve as

$$h_t = f(h_{t-1}, z_{t-1}, a_{t-1}), \tag{1}$$
$$z_t \sim q(z_t \mid h_t, x_t), \tag{2}$$
$$\hat{z}_t \sim p(\hat{z}_t \mid h_t), \tag{3}$$

where x_t denotes the observation. The world model is trained using reconstruction and reward prediction terms together with a dynamics regularization term between prior and posterior latent states. This abstractly corresponds to

$$\mathcal{L}_{\text{wm}} = \mathcal{L}_{\text{pred}} + \mathcal{L}_{\text{dyn}} + \mathcal{L}_{\text{rep}}. \tag{4}$$

The actor and critic are optimized on generated, or imagined, trajectories. Dreamer computes bootstrapped λ-returns,

$$R_t^\lambda = r_t + \gamma c_t((1-\lambda)v_t + \lambda R_{t+1}^\lambda), \tag{5}$$

and trains the critic toward these targets. The actor is optimized to maximize the predicted return with entropy regularization. For our project, the main appeal of Dreamer lies in its compact predictive model and the imagined interactions, which make it a natural choice for transferable exploration and our procedurally generated environments.

3 Methods

In this section, we outline the methodological framework behind our rule infrastructure and Dreamer-based agents for the Switchboard environment. We first introduce the Tree-Rules formalism (Section 3.1), which encodes causal and temporal dependencies between actions and observations as hierarchical rule trees. Building on this representation, we explain our procedural random-rule-generation pipeline (Section 3.2), which produces a diverse set of solvable tasks under structural constraints. Section 3.3 introduces a symbolic ground-truth solver based on Z3 that verifies rule satisfiability and extracts executable action sequences. Next, we introduce two complementary complexity scorers — structural and action-based — which enable us to quantify rule difficulty for curriculum learning and rule-set comparison (Section 3.4). Furthermore, we describe how the Dreamer architecture is adapted to the switchboard domain (Section 3.5). Finally, we introduce the setup of advanced Dreamer and curriculum-learning pipelines (Section 3.5.2 and Section 3.6). The source code of the implementations is available on GitLab.

3.1 Tree-Rules

To enable the adaptive generation of diverse rules, a RuleTree Structure is constructed from a core TreeRule class composed of LogicNode subclasses. It represents causal and temporal links between agent actions and environmental observations as hierarchical trees. These include leaf nodes (raw actions or observations), standard logical operators (AND, OR, NOT), and temporal nodes (delays, holds, toggles, sequences). Each node implements simple recursive functions for evaluation, logging, and ground truth solving.
This structure underpins the random rule generation, enabling a large variety of scenarios for standard Dreamer training. More importantly, it forms the backbone of the curriculum learning pipeline: since tree depth, node composition, and temporal constraints are all parameterizable, rules can be generated across curriculum levels in a controlled and scalable way.

3.2 Random Rule Generation

To obtain a broad and diverse task distribution, our switchboard rules are generated procedurally rather than designed by hand. For each observation slot, the generator recursively creates a rule tree from a weighted distribution over the node types. We limit tree depth and the generator randomly inserts predefined templates, which are intended to increase the diversity beyond simple random composition. The templates include recurring motifs such as inhibitions, exclusive-or relations, and observation-gated actions. The resulting rule trees are composed of logical and temporal operators as well as atomic leaves (see Section 3.1).
We impose several constraints to keep the rules expressive but manageable and non-trivial. For example, repeated negations, recursive sequence nesting or direct stacking of temporal operators are prohibited because they feign complexity without increasing task variety. We also force observation leaves to only reference slots that already have an associated rule. These acyclic dependencies make validation checking easier while at the same time allowing rules to depend on more than one slot. After sampling, a candidate rule is checked by the solver and only accepted if it is satisfiable (see Section 3.3).
Overall, the generation process produces a large family of solvable logical-temporal tasks. As rules differ in parameters, composition, and dependency structure, the resulting environments are well-suited for studying whether an agent can learn transferable exploration behavior rather than memorize solutions to fixed tasks.

3.3 Ground Truth Solver

In order to ensure that generated rules are not only diverse but also solvable, we use a symbolic ground-truth solver, namely Z3 [7]. In an earlier implementation, we attempted to solve rules recursively with node-specific heuristics, but this approach proved too brittle for more complex compositions. Given a rule tree and a fixed planning horizon, the solver constructs a symbolic constraint system by recursively adding the semantics of each node. It then checks whether there exists an action sequence that activates the corresponding observation slot at any point within that horizon, in which case the rule is considered satisfiable and the corresponding action sequence is extracted.
We use the solver in two stages. During rule generation, candidate rules are rejected if they are not satisfiable. The extracted action sequence is also validated by simulating it directly in the environment against the full current ruleset. This step ensures that rules that are locally satisfiable still hold in the complete scenario, since they may depend on other observation slots. Additionally, the action sequence can be used for analysis in Section 3.4.
The ground-truth solver serves both as a consistency check for procedural generation and as a reference method to determine whether a rule is achievable within a bounded number of steps. This way, we can ensure that the agent always has the potential for successful exploration and does not fail merely because a task is impossible by construction.

3.4 Complexity Scorer

We introduce complexity scorers to categorize the difficulty of rules. This classification is essential for curriculum learning (see Section 3.6) and for comparing randomly generated rules with predefined ones (see Section 4.1). In this Section, we describe two complementary approaches to quantify rule complexity: (i) *structural complexity*, which derives complexity from the internal structure of a rule (see Section 3.4.1), and (ii) *action-based complexity*, which evaluates the rule based on the ground-truth action sequence that satisfies it (see Section 3.4.2). Finally, we discuss the respective advantages and limitations of both methods (see Section 3.4.3).

3.4.1 Structural Complexity

The structural complexity score is derived directly from the rule's hierarchical tree representation (see Section 3.1). Depending on the type of node, we add or multiply specific amounts to the score. The weighting factors were determined empirically by manually classifying and comparing generated rules.
Leaf nodes: These correspond to action or observation nodes that specify whether a particular event must occur. Each leaf node contributes a score of 1. The final score of the rule is obtained by combining these values according to their parent nodes (logical or temporal).
Logic nodes: Each logic node has a base cost of 1. *NOT nodes* add their children's scores multiplied by 2, reflecting the higher complexity of logical inversion. *AND nodes* add the sum of their children's scores plus 0.5 times the number of children, accounting for the increasing difficulty with more conditions. *OR nodes* use the reciprocal sum of all children's scores, inspired by the principle of parallel electrical currents. The more alternative conditions exist, the simpler the rule becomes.
Temporal nodes: Temporal nodes have a base cost of 2, since temporal dependencies add complexity beyond static logic. The total score equals the base cost plus the children's scores and an additional time-dependent factor computed as follows:

$$\sum_{t=0}^{T} (t+1)\, e^{-\text{decay}\cdot t}$$

This function reduces the impact of large time steps based on a decay parameter (default: 0.2, adjustable via the constructor). For *decay nodes*, we approximate the duration using the reciprocal of the decay rate. *Toggle nodes* use a virtual cost of 10 time steps to model switching complexity. *Sequence nodes* add 2 times the number of children, representing additional steps per sequence element.

3.4.2 Action-Based Complexity

The action-based complexity scorer evaluates how difficult it is to satisfy a rule in practice by analyzing the corresponding ground-truth action sequences (see Section 3.3). The overall score is a weighted sum of four criteria (default weights (w)): (i) number of steps (w = 1.0); (ii) total number of button presses (w = 0.5); (iii) number of distinct buttons used (w = 0.1); (iv) entropy of the button usage distribution (w = 0.3). The weights were empirically tuned to align the results with the structural complexity scores, but can be modified through optional parameters.

3.4.3 Strengths and Weaknesses

Both scorers produce unit-less scores, where smaller values denote simpler rules, and the most complex rules approach an infinite score. We combine both scoring methods to achieve a more reliable estimate of rule complexity, due to the following reasons: (i) Simple rules can yield complex action sequences, because the solver is not always optimal (see Appendix B, Figure 9). Furthermore, the solver is non-deterministic, finding various solutions if executed multiple times. (ii) Conversely, complex structural rules may be satisfiable with simple action sequences (see Appendix B, Figure 10). Combining both metrics mitigates these effects and provides more stable results, which is crucial for effective curriculum learning.

3.5 Adapting Dreamer for Switchboard

The lecture-provided Dreamer implementation is limited in scope to exclusively cover world model learning in the **Shapes** environment. For this, two implementations are provided, consisting of the standard Dreamer implementation and an energy-based variant. Both use the same Dreamer wrapper and the same RSSM transition model, but differ in the world-model training objective: while the standard variant learns through decoder reconstruction of the observation, the energy-based variant supplements this with a contrastive observation–state matching objective and detaches the reconstruction loss from directly shaping the latent dynamics.

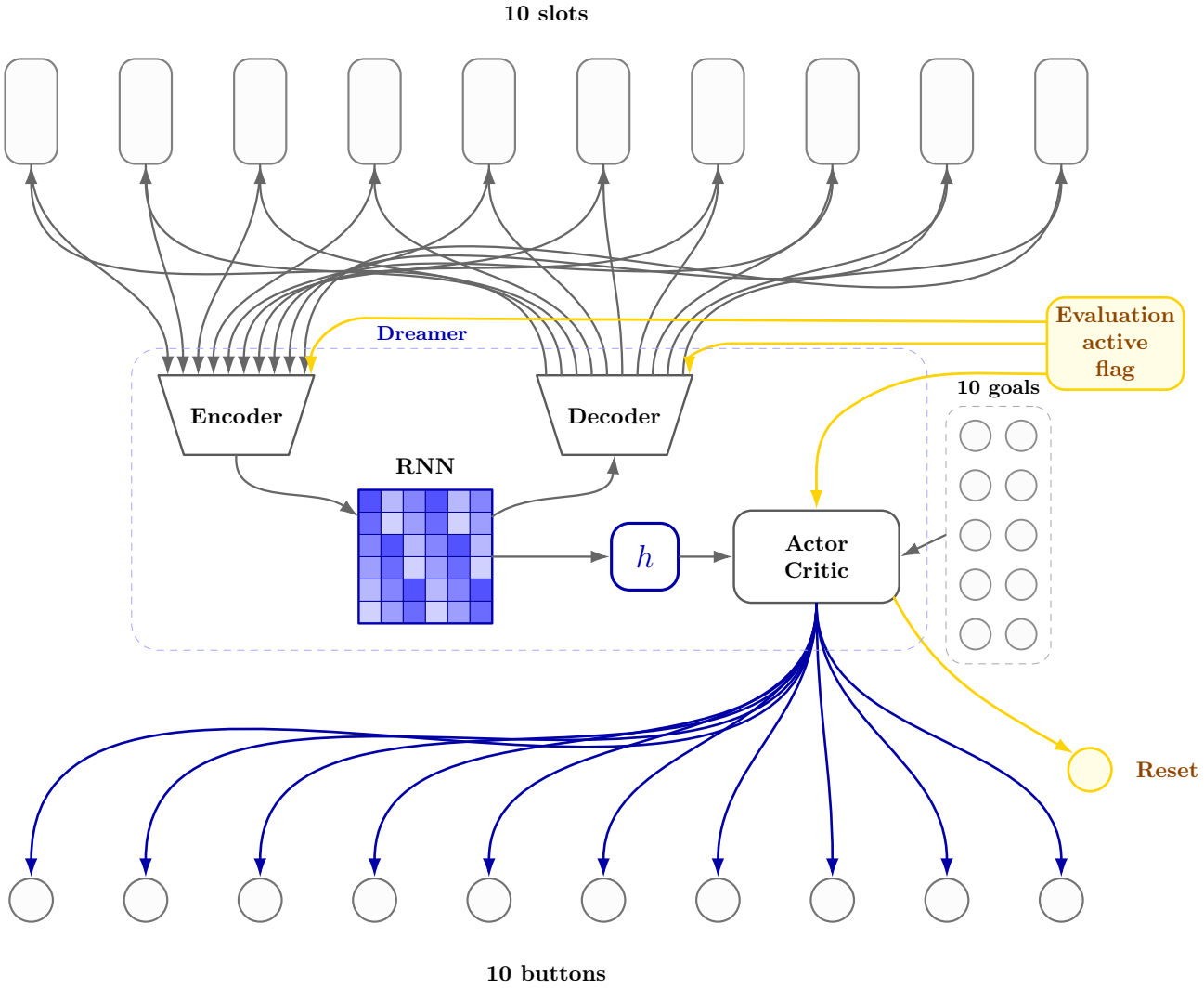

Figure 1: Dreamer-style architecture with slot observations, latent recurrent state, hidden state h, and an actor-critic conditioned on goals to produce button actions. All yellow elements indicate additions for adapting Dreamer to self-learning.

To adapt the Dreamer architecture to the **Switchboard** environment, several changes are necessary. Firstly, the Encoder and Decoder networks need to be replaced. The provided Dreamer implementation uses Convolutional Neural Networks (CNNs) [14] for encoding the image inputs from the **Shapes** environment, and for decoding and comparing predictions to the expected observations. As the Switchboard environment operates exclusively on vector-based states, the Encoder and Decoder networks are replaced with simple feedforward networks, whereby the input and output dimension match the state shape, projecting up in steps to the hidden state dimension of the RSSM, and down-projecting back into a state representation. Through this, the world model is taught to learn the environment's expected future state based on its current state.

Additionally, to allow the agent to act in the environment, an AC component is required, which learns exclusively on imagined rollouts of the RSSM state, following the standard Dreamer training procedure [11, 14]. The policy chosen is a latent goal-conditioned Bernoulli policy because the task is goal-directed and the action space is naturally binary, making Bernoulli outputs a direct fit for modeling switch activations. The policy consists of an input layer, two hidden layers with 256 units each, and an output layer. The Critic network has the same topology. The full modified Dreamer adaptation for the Switchboard environment is illustrated in Figure 1.

Through these modifications, the Dreamer implementation can observe and represent the environment in its hidden world-model state, and act based on its internal representation to achieve the goals it is given.

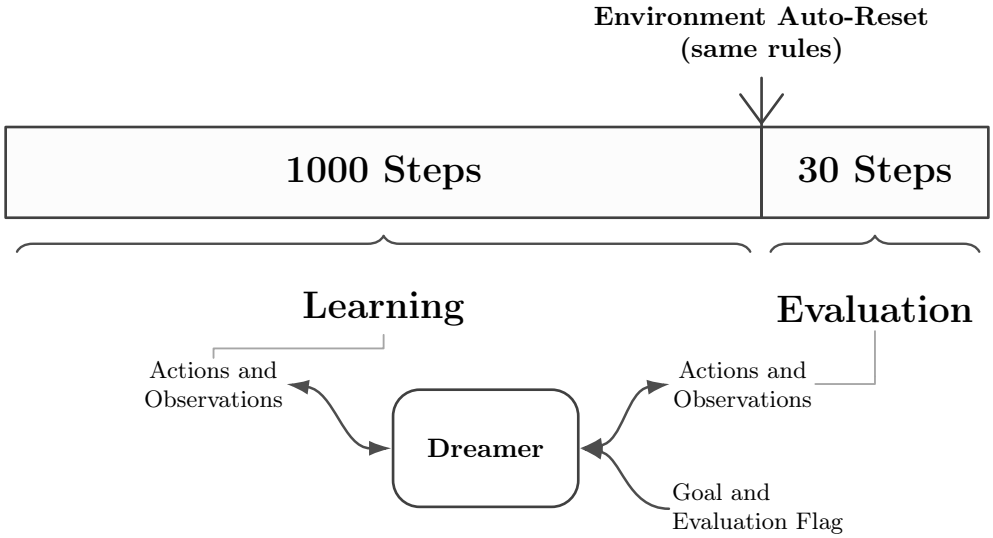

Figure 2: Training and evaluation schedule for Dreamer. During the learning phase, Dreamer interacts with the environment through actions and observations only. After an environment reset with unchanged rules, the evaluation phase additionally provides the goal and an evaluation flag.

3.5.1 Learning with Fixed Rules

The Dreamer Implementation was first used in training on the predefined rulesets given by the lecturers, specifically the **Direct Rules** and **Hard Rules**. The first ruleset contains exclusively button-to-slot mappings, while the second contains complex rules consisting of combined button presses, with several solution options, and temporally separated events necessary for triggering a slot's activation. The results related to these experiments can be seen in Section 4.1.

3.5.2 Self-Learning Dreamer

We adapt Dreamer to a meta-RL setting in which the agent must infer a newly sampled ruleset in each training block and then use that knowledge to solve goal-conditioned tasks. For this, each block is split into two distinct phases. The **Learn Phase** allows the model to experiment in its environment and learn the rules for this turn. This phase consists of significantly more steps than the **Answer Phase**, in which a goal is supplied, and goal attainment is evaluated. To signal these phases to the Dreamer, both the world model and the policy are supplied with an additional input denoting whether it is currently the evaluation phase. Additionally, as some rules require preconditions that can only be fulfilled in a new block (e.g. `Button 5, only if Button 2 was never pressed`), the policy is given the capability to reset the environment via a new output. These additions are illustrated in yellow in Figure 1.
During a single training run, the amount of steps in the **Learn Phase** and **Answer Phase** remain constant.
The training procedure remains largely the same. The World Model is trained to predict the state of the slots in the next step, given the current state and inputs in the current step. The policy is, as before, exclusively trained on hidden-state rollouts by the World Model.
The timeline of a single block of the self-learning process, as it is used for the experiments referenced in Section 4.2, is illustrated in Figure 2.

3.6 Setup Curriculum learning

Confronting the self-learner directly with complex rules requiring multiple distinct concepts simultaneously such as boolean and temporal logic may overwhelm the algorithm. To address this, we implement a curriculum learning strategy that structures training into a series of levels, each incrementally introducing new concepts — from basic boolean logic to more advanced compositional structures. The agent advances to the next level only after reaching a defined performance threshold, with the aim of building foundational knowledge before exposing it to greater complexity. Whether this staged approach outperforms direct training on the full task remains an open question. A direct comparison was not conducted due to compute constraints, though the small scale experiments presented in Section 5.3 provide some meaningful insight into the learning dynamics of the curriculum itself.

3.6.1 Extension of Existing Components

The curriculum learner's implementation builds primarily on the existing random rule generation and complexity scorer components, with minor adjustments and additions. The random rule generation was extended to support filtering of individual rules that fall outside a specified complexity score range. Since this filtering occurs within the generation loop, the number of rules per scenario remains consistent. This mechanism is used for outlier filtering to improve training stability. Support for serializing and loading rules from a JSON file was also added, reducing overhead during training, where SAT-checking makes repeated generation computationally expensive.

3.6.2 Definition of the Curriculum Levels

The random rule generation is heavily parameterizable, which allows for controlled construction of the curriculum levels. The key parameters are the proportion of different node types (such as logical and temporal nodes), the maximum depth of the rule trees, and a probability for early stopping of branch generation. The curriculum levels are designed around the progressive introduction of new concepts. The intention is to enable the model to gradually learn to handle increasingly complex rules, which may not be achievable when confronting it with full complexity from the start.
The levels first introduce boolean logic, then temporal rules, and finally a combination of both (see Table 1). Before generating the full training dataset, a smaller dataset was generated for analysis. The structural and action based complexity scores were computed per level. In the final datasets for training, rules deviating from the level mean by more than two standard deviations were filtered out to improve training stability.

Table 1: Definition of Curriculum Levels (Progressive complexity from logical to temporal sequences)

			Early Stop
Group	Level / Node Types	Max. Depth	Probability (p)
Logic	1: Direct Mappings	0	–
	2: AND	1	–
	3: AND, OR	2	0.05
	4: AND, OR	4	0.10
Temporal	5: hold, delay	1	–
	6: toggle	1	–
	7: sequences	1	–
Combination	8: all types	2	–
	9: all types	4	0.10

3.6.3 Curriculum Manager

The curriculum manager internally tracks the current level and supplies scenarios to the switchboard. Scenarios are sampled from a probability distribution that mixes the current level with earlier levels, providing replay to counteract forgetting. The probability distribution is individually parameterizable for each level. For example, when introducing a new concept such as the temporal rules, a high replay rate of the logical rules can be implemented to counteract catastrophic forgetting.
Level progression occurs when the average single slot success rate over a defined number of training steps exceeds a configurable threshold. A secondary condition allows progression if a step budget is exhausted and a lower threshold is met. Training stops entirely if the step budget is exhausted without meeting the lower threshold.

4 Experiments

This section evaluates Dreamer-based variants in the Switchboard environment through (i) baseline comparisons against AC, Goal-Conditioned Supervised Learning (GCSL), and CRL across rulesets of increasing complexity (Section 4.1), (ii) self-learning capabilities with scaled model architectures on direct-mapping rules (Section 4.2), and (iii) curriculum learning with progressive rule complexity and knowledge transfer analysis (Section 4.4).

4.1 Baseline-Comparison

In this section, we compare the performance of the two Dreamer variants, *VD* and *ED*, against three baseline models: *AC*, *GCSL*, and *CRL*. The evaluation focuses on how each model generalizes across different rulesets and varying levels of task difficulty.

Table 2: Comparison of rulesets (second run in parentheses)

				Mean Complexity Score		
Ruleset	Type of (possible) Nodes	Max. Depth	No. Rules	Structural	Action-based	Combined
Direct Rules	AND	1	6	2.8	2.1	2.5
Hard Rules	+ sequence, toggle	2	10	20.6	11.8	16.2
Random Rules	+ OR, NOT, hold, delay	3	10	18.7 (36.7)	50.1 (42.9)	34.4 (39.8)

4.1.1 Setup

Training was conducted on three distinct rulesets of increasing difficulty: *direct rules*, *hard rules*, and *random rules* (see Table 2). The direct and hard rules represent the easiest and most complex predefined rulesets provided with the switchboard environment and the baseline models by the lecturers. To compare them to the random rules, we converted all rulesets into our tree-based rule representation, as the complexity scorers require this structure as input. Direct rules implement single actions or AND combinations of actions corresponding to a single observation. Hard rules introduce sequences, toggles and observations as inputs. Random rules form the most challenging configuration, additionally including the temporal nodes *hold* and *delay*. The corresponding rule trees were generated randomly using default parameters (see Section 3.2), except for the tree depth, which was set to 3.

4.1.2 Evaluation

Model performance was evaluated using two primary metrics: the *success rate* and the *mean trajectories* over 50 runs enabling a detailed comparison. To account for the inherent stochasticity in the learning process, each model–ruleset combination was trained twice under identical conditions. This approach ensures that the results reflect consistency rather than randomness. For the random ruleset, we generated new rules using the same parameters.

Table 3: Success rates per model and rule: The success rate denotes the rolling average over the final 100 blocks of each training run. Results show the mean and first/second run values in parentheses. *Note:* Direct rules capped at 0.67 (8/12 achievable observations).

	Ruleset		
Model	Direct Rules	Hard Rules	Random Rules
Actor-Critic	0.50 (0.48, 0.52)	0.50 (0.44, 0.55)	0.38 (0.37, 0.38)
Goal-Conditioned Supervised Learning	0.52 (0.53, 0.50)	0.41 (0.38, 0.43)	0.38 (0.40, 0.36)
Contrastive Reinforcement Learning	0.59 (0.59, 0.59)	0.57 (0.57, 0.56)	0.44 (0.50, 0.38)
Vanilla Dreamer	0.62 (0.60, 0.64)	0.71 (0.73, 0.69)	0.39 (0.48, 0.29)
Energy Dreamer	0.57 (0.54, 0.60)	0.69 (0.63, 0.75)	0.42 (0.46, 0.38)

4.1.3 Analysis

The subsequent analysis presents the results for each model, highlighting where particular approaches outperformed others. Table 3 provides an overview of the success rates achieved by each model across the different rulesets. In Appendix C, all trajectories of each model, shown for every ruleset and every run, are presented.

Direct Rules: Since direct rules were capped at 0.67, most models approached this upper limit, indicating that they reliably mastered the directly observable tasks. The Dreamer-based models and CRL achieved the highest performance, followed by GCSL and AC.

Random Rules: Performance on random rules was consistently the lowest across all models, with success rates rarely exceeding 0.50. This decline indicates that models struggled to extract meaningful patterns when the underlying rule structures where to complex or lacked coherence.

Hard Rules: Hard rules produced the largest spread in performance, providing a more detailed view of model differences. Advanced models such as VD and ED achieved notably higher success rates (up to 0.73 and 0.75), demonstrating stronger generalization and rule inference capabilities. Simpler models such as AC and GCSL showed more limited progress, underlining the increased difficulty of these tasks. Trajectory analysis (see Figure 3) reveals that the Dreamer-based models manage sequences with three actions, while sequences of four or five actions are more challenging yet still occasionally solvable. Executing multiple button presses is handled well, but omitting specific actions is only partially learned. VD in run 1 and ED in run 2 succeed in this respect, explaining their swapped performance between runs. Observing multiple slots proved particularly difficult: managing one slot is feasible, whereas handling two or three becomes increasingly challenging. In contrast, CRL partly succeeded in observing slots 1 and 2 in combination with an additional action, outperforming the Dreamer models in this specific task.

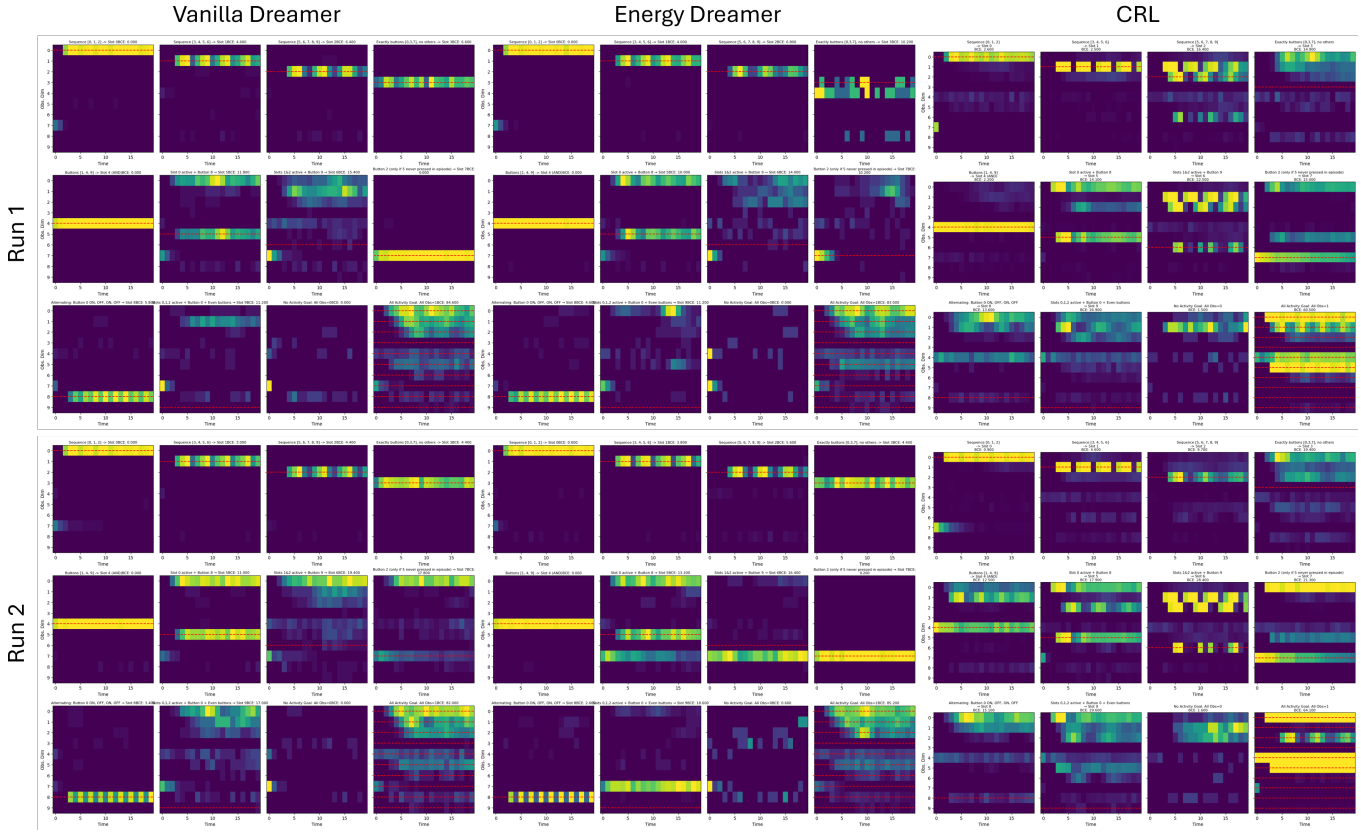

Figure 3: Trajectories of Vanilla Dreamer, Energy Dreamer and Contrastive Reinforcement Learning (CRL) for hard rules

In summary, VD achieved the best performance on solvable rules, followed by ED, whereas CRL provided the most consistent results and emerged as the best baseline model. In Section 5.1, we discuss potential reasons for the differences in performance.

4.2 Self-Learning Dreamer

The following experiments evaluate Dreamer as a self-learning Meta-RL Agent in the Switchboard environment.

4.2.1 Setup

To assess the capability of Dreamer as a self-learning Meta-RL Agent, we conducted several training runs with three different Dreamer model sizes. All experiments described in this section were run with randomly generated rules, but limited in complexity to exclusively direct-connection rules, namely rules that require only a single button press to turn on a certain slot, but the assignment of buttons to slots is unknown to the dreamer before the learning phase. The used architecture was somewhat modified from the one used in Section 4.1. The architectural changes taken are described in Section 3.5.2, and illustrated in yellow in Figure 1.
The model sizes and training hyperparameters used in these experiments are shown in Table 4.

Table 4: Model configurations used in self-learn experiments. Learn steps, answer steps, and unroll steps are identical across all variants.

	Architecture					Training		
Model	lat	cat	det	Params	Energy WM	Learn steps	Answer steps	Unroll steps
Vanilla-Tiny	16	4	256	400K	×	1000	30	16
Vanilla-Medium	256	16	2048	12M	×	1000	30	16
Vanilla-Large	768	48	6144	100M	×	1000	30	16
Energy-Tiny	16	4	256	400K	✓	1000	30	16
Energy-Medium	256	16	2048	12M	✓	1000	30	16
Energy-Large	768	48	6144	100M	✓	1000	30	16

Each experiment consisted of training the Dreamer model on 5000 Blocks with ten randomized direct-mapping rules per block, one per slot. During training, several metrics were collected, including reward, loss and goal attainment success rate.

4.2.2 Evaluation

After each experiment, an evaluation of the final checkpoint on 128 randomly generated rulesets was performed. These evaluations consistently showed results coinciding with the latest metrics reported during training. Several after-training evaluation graphs can be seen in Appendix D. The metrics shown in this section and discussed are metrics gained during training, as these correspond closely to the evaluation metrics at the end of the experiment, but offer insight on how performance on these metrics changed over the course of the training run.

4.3 Assumptions and Reasoning

The core assumption for this project is that Dreamer's world model is able to learn abstract rule types, and then encode the rules of the current turn in its hidden state. The policy can then use the encoded information about the rules from the World Model's hidden state to choose the right actions to achieve the requested goal.
This reasoning is based on Dreamer managing to achieve complex long-horizon tasks such as finding diamonds in Minecraft, which requires different sets of actions depending on the world it is in. In essence, Dreamer learns different rulesets for different situations in the game, such as being able to swim in water, or jump when confronted with an obstacle.

4.3.1 Limitations

The experiments are limited to only three Dreamer model sizes. Additionally, all experiments in this section were performed exclusively with simple single-button-to-slot-mapping rules.

4.4 Curriculum Learning

The following experiments evaluate the curriculum learning pipeline introduced in Section 3.6, studying how the agent adapts as it is exposed to rules of increasing complexity across curriculum levels and whether knowledge acquired at earlier levels transfers to more complex ones.

4.4.1 Setup

The dataset was split into training, validation, and test sets. Rule evaluation on the switchboard is computationally expensive, as it requires substantial CPU resources despite GPU optimization, which constrained the scope of the experiments. As a result, only one full training run and no hyperparameter optimization was conducted.
The training run covered the first four curriculum levels, corresponding to basic boolean logic with increasing tree depth. Due to compute constraints, higher levels were not explored. Progression to the next level occurred either when the agent's average success rate exceeded a 70% threshold, or automatically after a budget of 5000 training blocks was exhausted for a given level. To prevent catastrophic forgetting, training scenarios were sampled according to a dynamic replay scheme: 70% from the current level, 15% from the previous level, and the remaining 15% distributed uniformly across all earlier levels. At level 1 and 2, where fewer prior levels exist, the probabilities were adjusted accordingly.

Throughout training, the following metrics were logged: single-slot success, none-goal success and all-goal success on the training data, as well as world model and policy metrics. At each level transition, a checkpoint of the world model and policy weights was saved for later evaluation. All hyperparameters are listed in Table 5.

Figure 4: Training progress during the curriculum learning. The 4 curriculum levels are indicated by the background coloring. Level 1 passed the upper threshold after about 642 Blocks. The other levels didn't meet the upper threshold and advanced automatically after 5000 blocks.

4.4.2 Evaluation

Evaluation followed a zero-shot protocol with frozen model weights. For each unseen scenario, the agent first executes random actions for a fixed number of steps to build an internal world model state representing the environment's rules. Subsequently, the learned policy is evaluated on its ability to reach a standard set of goals within a time limit. All intermediate checkpoints were tested across all four levels using 20 test scenarios per level, none of which were seen during training. Each intermediate checkpoint corresponds to a model trained on all levels up to and including that point — the level 1 checkpoint on level 1 only, the level 2 checkpoint on levels 1 and 2, and so on. Performance is measured as the mean goal success rate. Results are presented below and discussed in detail in Section 5.3.

4.4.3 Limitations

The evaluation is limited to boolean logic, covering only the first four curriculum levels. No comparison to direct training on higher complexity levels without curriculum warm-up was conducted, which would be necessary to assess the actual benefit of the curriculum approach. Additionally, no hyperparameter optimization was performed, neither for the self-learning Dreamer model nor for the curriculum manager.
Despite these limitations, the results still provide meaningful insight. The training run allows observation of how the model adapts to environment switches across curriculum levels, whether it retains previously acquired knowledge when transitioning to new levels, how it handles increasingly complex switchboard configurations beyond simple direct mappings, and how performance evolves over the course of training.

5 Discussion

In this section, we discuss (i) the baseline comparison results (Section 5.1), (ii) the performance of the self-learning Dreamer (Section 5.2), and (iii) the effectiveness of curriculum learning (Section 5.3). Finally, we relate these findings to concepts from the AGI lecture (Section 5.4).

5.1 Baseline Comparison

The baseline comparison reveals that the Dreamer-based models VD and ED consistently outperform the baselines AC, GCSL, and CRL on the more complex *hard rules*, achieving success rates up to 0.75. The result underscores the value of the world model in Dreamer architectures, which enables better learning of complex rule compositions and temporal dependencies. Consequently, Dreamer variants demonstrate superior generalization to rule structures

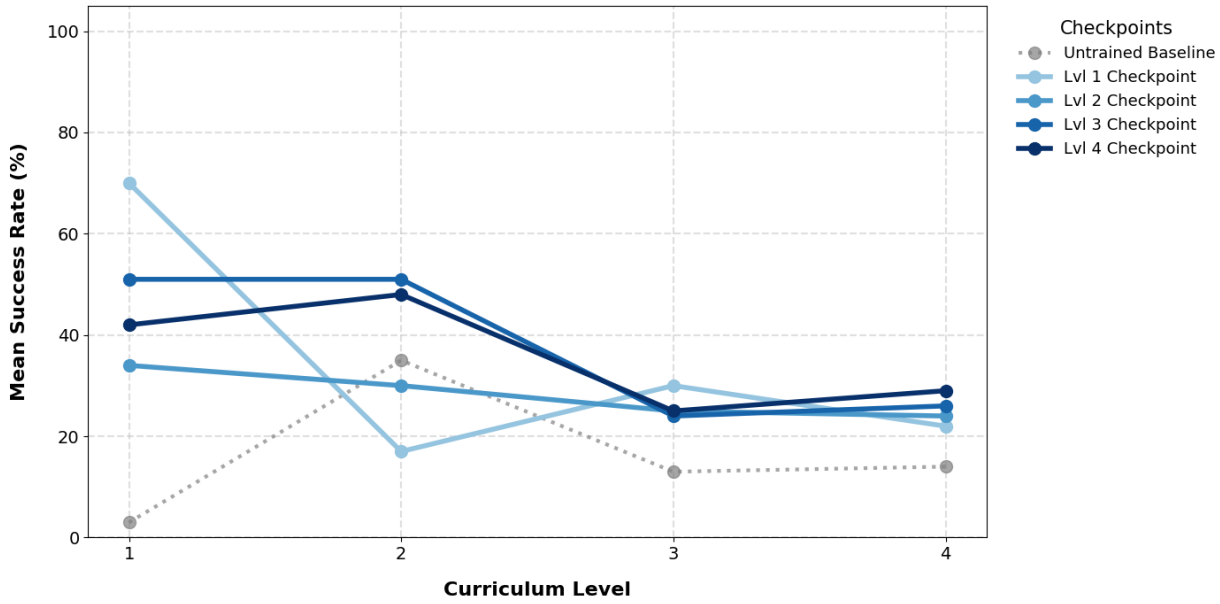

Figure 5: Mean success rate across the four curriculum checkpoints. Each checkpoint was trained on a progressively larger subset of curriculum levels: the Level 1 checkpoint on the first level only, Level 2 on the first two levels, and so on.

requiring internal planning and diverse action exploration, as evidenced by their ability to handle sequences of up to five actions and partial omission of irrelevant actions.

Performance patterns across rulesets provide deeper insights into model strengths. On *direct rules*, all models approach the theoretical cap of 0.67, reflecting straightforward action-observation mappings leaving little room for differences. In contrast, *random rules* yield uniformly low success rates across models. They challenge rule inference, due to their parameterization allowing deep nesting and diverse logical/temporal nodes resulting in sparse feasible solutions. The *hard rules* strike an optimal balance: sufficiently complex (sequences, toggles, observations) to differentiate models without overwhelming them, allowing Dreamer models to leverage latent planning for higher performance.

Several limitations temper the strength of these findings and suggest avenues for future work. First, the observed run-to-run variability indicates that two seeds per configuration capture only tendencies, not definitive superiority, compounded by incomplete convergence due to resource constraints. Regenerating random rules for the second run, while demonstrating generation variance under fixed parameters, further limits direct comparability. Additionally, intermediate rulesets between *hard* and *random rules* could identify the exact complexity threshold where Dreamer advantages diminish. Finally, relying solely on success rates and trajectories overlooks complementary metrics, which could provide a more holistic evaluation.

Overall, these results affirm the potential of world-model-based approaches for rule inference in structured environments like Switchboard, while emphasizing the need for extended training, more seeds, and diverse metrics to solidify claims.

5.2 Self Learning Dreamer

Training the Self-learning Dreamer on Direct Rules shows improvement in goal attainment after the explore phase. Notably, almost all model sizes have high success rates for None- and All-Goal attainment as can be seen in Figure 7. Several models, especially the vanilla- and energy-tiny models, attain single-slot success rates of approximately 60% after training. As can be seen in Figure 6, all model sizes increase the reward they receive significantly compared to the initial reward value.

A rather unexpected observation is that Dreamer model size seems inversely correlated with peak and average success rate for this self-learning exercise. Figure 8 shows a significant decrease in success rate with increasing model size, which is even more pronounced for VD models. Over the training, the single-slot success rate for bigger models, specifically Vanilla medium and large models, fluctuates significantly, often reaching up to 50% or more, but steeply decreasing to about 0.1 again after. Similar instability can be observed in the None-Goal Success evaluation.

An interesting contrast is that the two models that attain the highest reward over much of their training, Vanilla Medium and Vanilla Large, which both fluctuate around a reward mean of 0.7 (Figure 6), have lower or significantly lower average success rates than Vanilla Tiny over their training, with average success rates of 38% for single-slot

attainment in the tiny model, and only 24% in the medium and 18% in the large model in the same task (Figure 8). A potential explanation for this can be seen in the reconstruction loss graphs in Figure 6, as the tiny vanilla model outperforms both bigger models both during learning and in the answer phase.
A clear problem with the training setup can be seen in the lacking decrease, or partially increase in Reconstruction Loss from the Learn phase to the Answer phase visible in Figure 6. If the world model would learn the existing rule types, and then represent the current ruleset in its hidden state, the expectation would be for the answer reconstruction loss to be lower than the Learn Reconstruction Loss, as the average over the Learn Reconstruction Loss also includes the period of time in which the model must explore the different rules, and cannot have knowledge about the rules yet. The lacking decrease (and even increase in the case of the energy large model) hints that the world model does not manage to encode the rules in its hidden state in sufficient quality to predict the outcomes of actions over its learning time.

5.2.1 Interpretation

The high reward attained combined with the low performance in goal attainment shown by the larger models combined with the lacking decrease in reconstruction loss during the answer phase could both be explained by one common factor, namely the world model failing to learn the environment in any significant way. This is, to an extent, unexpected, as the environment is changing with each block. However, the assumption this experiment is based on is that the world model could learn not just the concrete rules used by the current environment, but instead pick up stable meta-structure in the form of recurring rule types and rule semantics. In principle, the world model could therefore learn these shared regularities and encode the currently active ruleset in its hidden state during the Learn Phase. If this were successful, one would expect the Answer Phase to benefit from that inferred structure, for example through lower reconstruction loss. The lack of such a pattern suggests that the model does not reliably capture the current ruleset in a form that is useful for downstream control.
This interpretation also helps explain the observed increase in reward despite limited improvement in goal attainment. Since the policy is trained exclusively on trajectories imagined by the world model rather than directly on the real environment, errors in the learned dynamics can induce a mismatch between optimization target and actual task success. In that case, the policy may improve reward under the learned model without becoming better at solving the true environment. In other words, the more 'confused' the world model is about how the world looks like, the less the policy can learn from it, as the policy reward in the learned model may drift away from real task success.
The inverse correlation of model size and performance is harder to pinpoint. One possible explanation is that the larger models were harder to optimize under the fixed training budget and hyperparameter setting. While possible, this contrasts with the occasional performance spikes observed, and the lack of a consistent trend of performance increase. A second explanation is that bigger world models prefer complex hidden state transitions for more complex environments instead of simply representing the current ruleset internally. Smaller world models might therefore forward more information to the policy by virtue of mangling the world state less strongly, allowing the policy to learn with inputs that correspond more closely to the current ruleset.
The gains the models achieved over the training are more likely than not driven by the policy rather than the world model. As all rules used in these experiments are direct ones, achieving the All-Goal is as simple as pressing all the buttons, and achieving the None-Goal is as simple as pressing none. These goals can be achieved independent of the information supplied by the world model. This might explain the almost 100% success rate for both of these goal

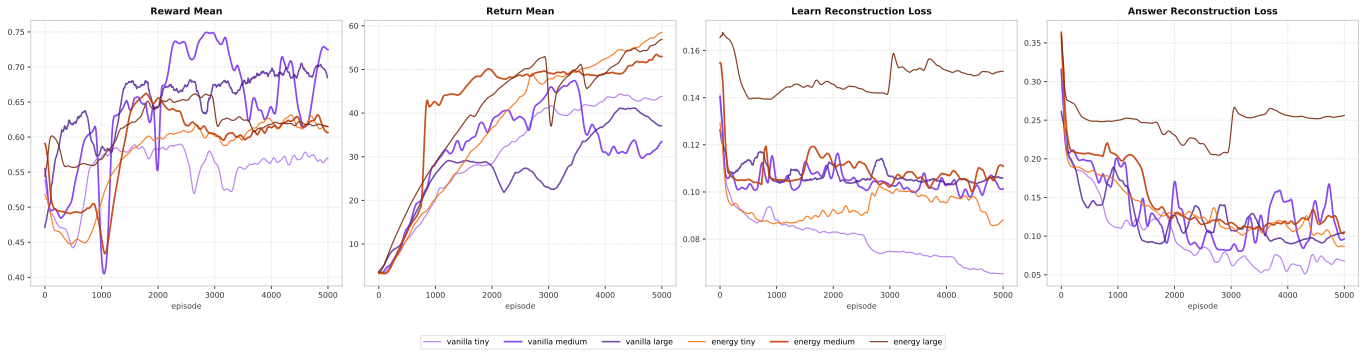

Figure 6: Training Metrics of Self-Learning Dreamer Models during training. Reward and Return metrics concern the policy, while Explore and Answer Rec Loss describe the reconstruction loss of the world model during the Learn and Answer phases

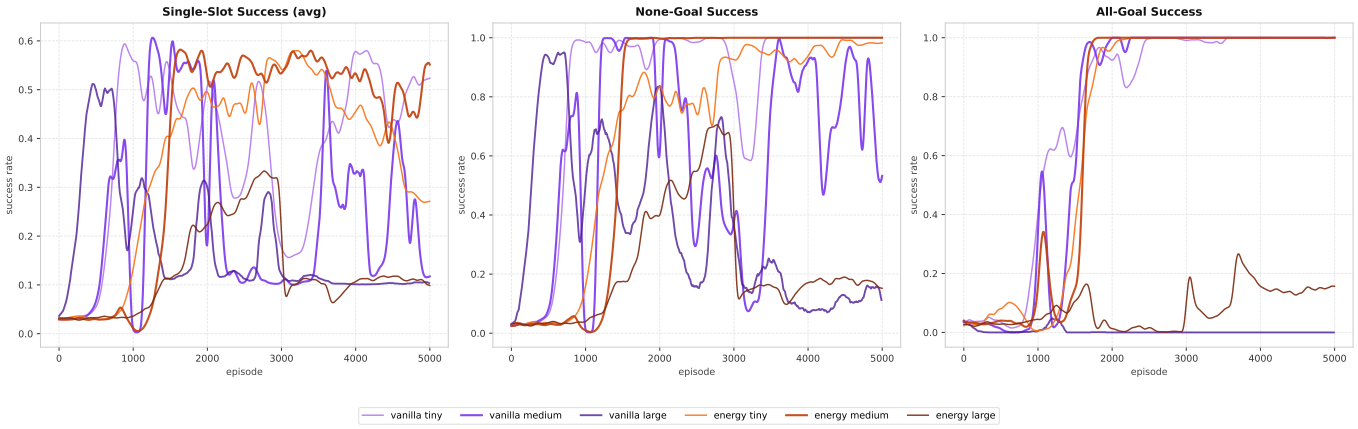

Figure 7: Task Achievement Success for Self-learning Dreamer models. Shown is the average per-slot success rate, as well as the success rate for the goal `No slots are active` and `All slots are active`

.

types.

Addressing these issues likely requires changes to the world-model training procedure. One possible starting point would be to delay world-model updates during the initial part of the Learn Phase. This would reduce the extent to which the model is penalized for failing to predict dynamics before it has had any opportunity to infer the rules of the current block.

5.3 Curriculum Learning

An initial finding is evidence of negative transfer between levels 1 and 2. The level 1 checkpoint achieves 70% on level 1 but only 17% on level 2, below the untrained baseline of 35%. This is not surprising given the conflicting objectives: level 1 rewards pressing individual buttons, while level 2 requires simultaneous actions to satisfy AND rules. A randomly exploring baseline is actually more likely to accidentally satisfy an AND rule than a policy that has learned to act conservatively.

Introducing level 2 training triggers catastrophic forgetting. Level 2 performance improves slightly (17% $\rightarrow$ 30%), but level 1 drops sharply (70% $\rightarrow$ 34%). Despite curriculum replay, the policy fails to reconcile the two conflicting strategies and converges to a suboptimal compromise, performing worse on both than on the more complex levels it has not yet seen.

The more telling result comes from the level 3 and 4 checkpoints. Despite the apparent conflict between levels 1 and 2, the level 3 checkpoint recovers strongly on both (51% each). This suggests the model did not simply find a better compromise but learned a more flexible policy. Level 3 introduces OR rules, which like level 1 can be satisfied by a single action. Exposure to a mix of AND and OR structures may prevent the policy from overspecializing in either

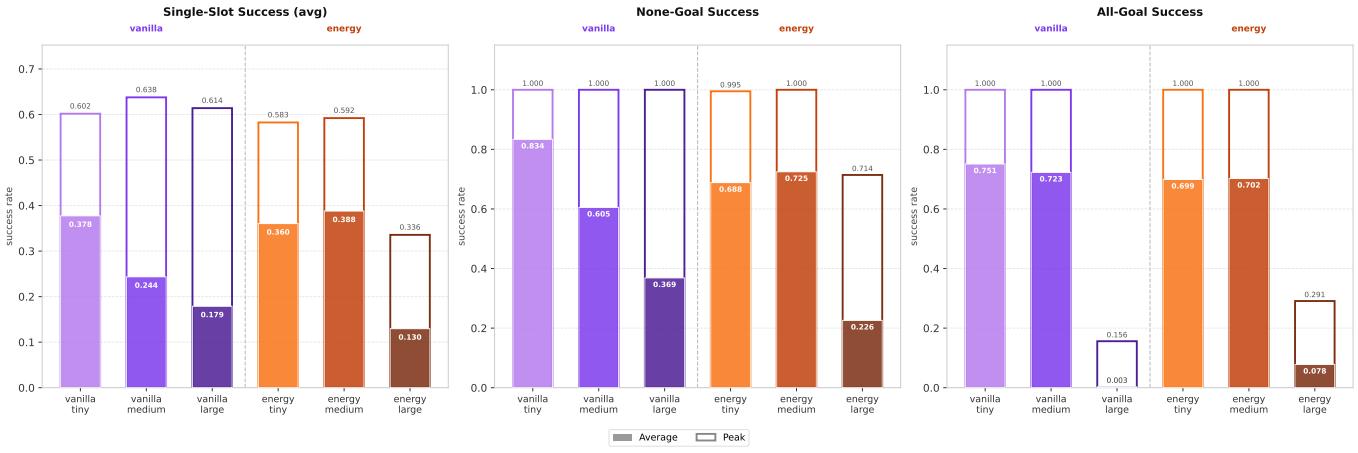

Figure 8: Peak Success Rate per model for Single-Slot Average, None and All goals over the training

direction, forcing context-dependent behavior.
Performance on levels 3 and 4 remains below 30% across all checkpoints, though every trained model outperforms the baseline there. This suggests that even when the policy cannot deduce the specific rules, it has learned a more structured exploration strategy that provides a consistent if modest advantage.
These results should be interpreted with the caveat that the model is small and no hyperparameter tuning was performed. The observed dynamics are informative about curriculum learning behavior, but the absolute performance figures are unlikely to represent the ceiling of this approach. Also as mentioned earlier no baseline for a model trained only on level 3 or 4, without the prior curriculum learning is given, which makes an assessment of the impact of curriculum learning difficult. The experiments rather give insights on how the addition of rules of increasing complexity affects performance.

5.4 Relation to the AGI lecture

A central problem in the pursuit of AGI lies in enabling agents to generalize across a wide range of tasks rather than achieving high performance on fixed benchmarks. This motivates studying settings where memorization is insufficient. Procedurally generated environments, such as the one used in our work, address this by exposing agents to a diverse set of tasks. This can be loosely compared to how humans often learn through play-pretend and practice in varied, low-risk settings.
Another key idea is that effective exploration may be fundamental to intelligent behavior. In many real-world scenarios, rewards are sparse or delayed, requiring agents to gather information without immediate feedback. This suggests that exploration should itself be treated as a learnable capability. Our work follows this perspective by studying whether exploration strategies can be learned and transferred across tasks. Humans and animals naturally explore unfamiliar environments, and curiosity-driven behavior is often linked to discovery and learning, suggesting that similar mechanisms may be important for artificial agents.
It has also been argued that internal world models are an important basis for reasoning, prediction, and planning. Model-based approaches, such as Dreamer, allow agents to simulate potential outcomes and act more deliberately. Humans rely on internal representations of the world to anticipate consequences of actions, which motivates the use of learned predictive models in artificial systems.
Finally, ideas such as curriculum learning and continual learning are often discussed as mechanisms for scaling intelligence. Gradually increasing task difficulty and exposure to new situations encourage the development of reusable skills rather than task-specific solutions. Human learning similarly builds on previously acquired knowledge, supporting the hypothesis that such mechanisms are important for more general agents.
Overall, the relevance of our work lies in probing a set of core hypotheses: that generalization, learned exploration, world models, and continual adaptation are important ingredients for more generally intelligent systems.

6 Conclusion

In this paper, we investigated the feasibility of applying the Dreamer MBRL architecture for meta reinforcement learning to explore the capability for world models to generalize beyond previously seen environments. We adapted various Dreamer-Models — Vanilla Dreamer (VD), Energy Dreamer (ED), and a meta-reinforcement learning dreamer — and studied key aspects in the Switchboard environment: (i) the performance of VD and ED compared to model-free reinforcement learning baselines (AC, GCSL, CRL) across distinct rulesets; (ii) a continual training where the self-learning dreamer is repeatedly exposed to new environment configurations while retaining learned parameters; (iii) curriculum learning effects on progressively complex rules.
Our results show that: (i) Dreamer-based models outperformed baselines on solvable rulesets by up to 50%, with VD reaching 0.71 compared to GCSL 0.41 success rate on hard rules, indicating latent planning and diverse exploration foster superior generalization to complex rules; (ii) continual training of the self-learning dreamer enabled adaptation to new environment configurations without full retraining, retaining prior knowledge and reducing switching costs across rule changes to a limited extent, while finding limitations in the world model picking up stable meta-structure in the form of recurring rule types and rule semantics. We propose potential recommendations for improving performance in future work. (iii) curriculum learning revealed initial negative transfer and catastrophic forgetting between levels 1–2 (70% $\rightarrow$ 34% on level 1; 17% on level 2), but level 3–4 checkpoints recovered to 51% on prior levels via more flexible policies. Trained models consistently beat a random initialized baseline on complex levels through structured exploration, though absolute performance remains modest without tuning.
These findings partially achieve our goal of demonstrating transferable exploration in a limited fashion, as Dreamer models developed more generalist policies despite initial forgetting, providing a foundation for future work on world model generalization with regards to AGI applications.

For future work, we suggest exploring training optimizations to bias the world model into capturing more meta-structure of the environment. Additionally, extending transferable exploration by adapting the Dreamer to unseen environments with varying inputs/outputs, incorporating hyperparameter tuning, and larger models to assess full benefits, are further avenues of research.

References

[1] Yuri Burda et al. "Large-scale study of curiosity-driven learning". In: *arXiv preprint arXiv:1808.04355* (2018).

[2] Eddy Keming Chen et al. "Does AI Already Have Human-Level Intelligence? The Evidence Is Clear". In: *Nature* 650.8100 (Feb. 2026), pp. 36–40. ISSN: 1476-4687. DOI: 10.1038/d41586-026-00285-6. URL: https://www.nature.com/articles/d41586-026-00285-6 (visited on 03/30/2026).

[3] François Chollet. "On the measure of intelligence". In: *arXiv preprint arXiv:1911.01547* (2019).

[4] Karl Cobbe et al. "Leveraging procedural generation to benchmark reinforcement learning". In: *International conference on machine learning*. PMLR. 2020, pp. 2048–2056.

[5] Karl Cobbe et al. "Quantifying generalization in reinforcement learning". In: *International conference on machine learning*. PMLR. 2019, pp. 1282–1289.

[6] Cédric Colas et al. "Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey". In: *Journal of Artificial Intelligence Research* 74 (2022), pp. 1159–1199.

[7] Leonardo De Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2008, pp. 337–340.

[8] Adrien Ecoffet et al. "Go-explore: a new approach for hard-exploration problems". In: *arXiv preprint arXiv:1901.10995* (2019).

[9] Sébastien Forestier et al. "Intrinsically motivated goal exploration processes with automatic curriculum learning". In: *Journal of Machine Learning Research* 23.152 (2022), pp. 1–41.

[10] David Ha and Jürgen Schmidhuber. "World models". In: *arXiv preprint arXiv:1803.10122* 2.3 (2018), p. 440.

[11] Danijar Hafner et al. "Dream to control: Learning behaviors by latent imagination". In: *arXiv preprint arXiv:1912.01603* (2019).

[12] Danijar Hafner et al. "Learning latent dynamics for planning from pixels". In: *International conference on machine learning*. PMLR. 2019, pp. 2555–2565.

[13] Danijar Hafner et al. "Mastering atari with discrete world models". In: *arXiv preprint arXiv:2010.02193* (2020).

[14] Danijar Hafner et al. "Mastering diverse domains through world models". In: *arXiv preprint arXiv:2301.04104* (2023).

[15] Nicklas Hansen, Hao Su, and Xiaolong Wang. "Td-mpc2: Scalable, robust world models for continuous control". In: *arXiv preprint arXiv:2310.16828* (2023).

[16] Isaac Kauvar et al. "Curious replay for model-based adaptation". In: *arXiv preprint arXiv:2306.15934* (2023).

[17] Yann LeCun et al. "A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27". In: *Open Review* 62.1 (2022), pp. 1–62.

[18] Shane Legg and Marcus Hutter. "Universal intelligence: A definition of machine intelligence". In: *Minds and machines* 17.4 (2007), pp. 391–444.

[19] Meredith Ringel Morris et al. *Levels of AGI for Operationalizing Progress on the Path to AGI*. arXiv.org. Nov. 4, 2023. URL: https://arxiv.org/abs/2311.02462v5 (visited on 03/30/2026).

[20] Pierre-Yves Oudeyer and Frederic Kaplan. "What is intrinsic motivation? A typology of computational approaches". In: *Frontiers in neurorobotics* 1 (2007), p. 108.

[21] Deepak Pathak et al. "Curiosity-driven exploration by self-supervised prediction". In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787.

[22] Stuart Russell and Peter Norvig. *Artificial Intelligence, Global Edition A Modern Approach*. Pearson Deutschland, 2021, p. 1168. ISBN: 9781292401133. URL: https://elibrary.pearson.de/book/99.150005/9781292401171.

[23] Bernhard Schölkopf et al. *Towards Causal Representation Learning*. Feb. 22, 2021. DOI: 10.48550/arXiv.2102.11107. arXiv: 2102.11107 [cs]. URL: http://arxiv.org/abs/2102.11107 (visited on 03/30/2026). Pre-published.

[24] Julian Schrittwieser et al. “Mastering atari, go, chess and shogi by planning with a learned model”. In: *Nature* 588.7839 (2020), pp. 604–609.

[25] Ramanan Sekar et al. “Planning to explore via self-supervised world models”. In: *International conference on machine learning*. PMLR. 2020, pp. 8583–8592.

[26] Ashish Sundar, Chunbo Luo, and Xiaoyang Wang. “Enter the Void-Planning to Seek Entropy When Reward is Scarce”. In: *arXiv preprint arXiv:2505.16787* (2025).

[27] Rich Sutton. *The Bitter Lesson — incompleteideas.net*. `http://www.incompleteideas.net/{I}nc{I}deas/{B}itter{L}esson.html`. [Accessed 24-03-2026]. 2019.

A Usage of AI Assistants

AI assistants were used in this work to refine language, such as grammar correction, punctuation, and sentence structure. Furthermore, AI coding assistants were used in the development of the project. All research ideas, methodologies, and analyses are the original contributions of the authors. Thus, the use of AI is confined to editorial support and did not influence the originality or intellectual contributions of the work.

B Complexity Scorer

```
======================================================================
Target Observation Index: 5
Rule Structure:
OR
  Action 7
  Action 6
Status: VALID
Required Actions: 22 steps
======================================================================

Step  0: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  1: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  2: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  3: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  4: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  5: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  6: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  7: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  8: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  9: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 10: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 11: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 12: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 13: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 14: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 15: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 16: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 17: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 18: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 19: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 20: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step 21: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0] -> Actions: [7]

======================================================================

Complexity Score of rule: 1.5
Complexity Score of output: 22.6
```

Figure 9: Simple rule with long action sequence: structural complexity = 1.5, action-based complexity = 22.6

```
======================================================================
Target Observation Index: 1
Rule Structure:
AND
  Action 7
  NOT (
    DELAY 5
      Action 2
  )
Status: VALID
Required Actions: 6 steps
======================================================================

Step  0: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  1: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  2: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  3: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  4: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] -> Actions: None
Step  5: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0] -> Actions: [7]

======================================================================

Complexity Score of rule: 28.18
Complexity Score of output: 6.6
```

Figure 10: Complex rule with simple action sequence: structural complexity = 22.18, action-based complexity = 6.6

C Baseline Comparison

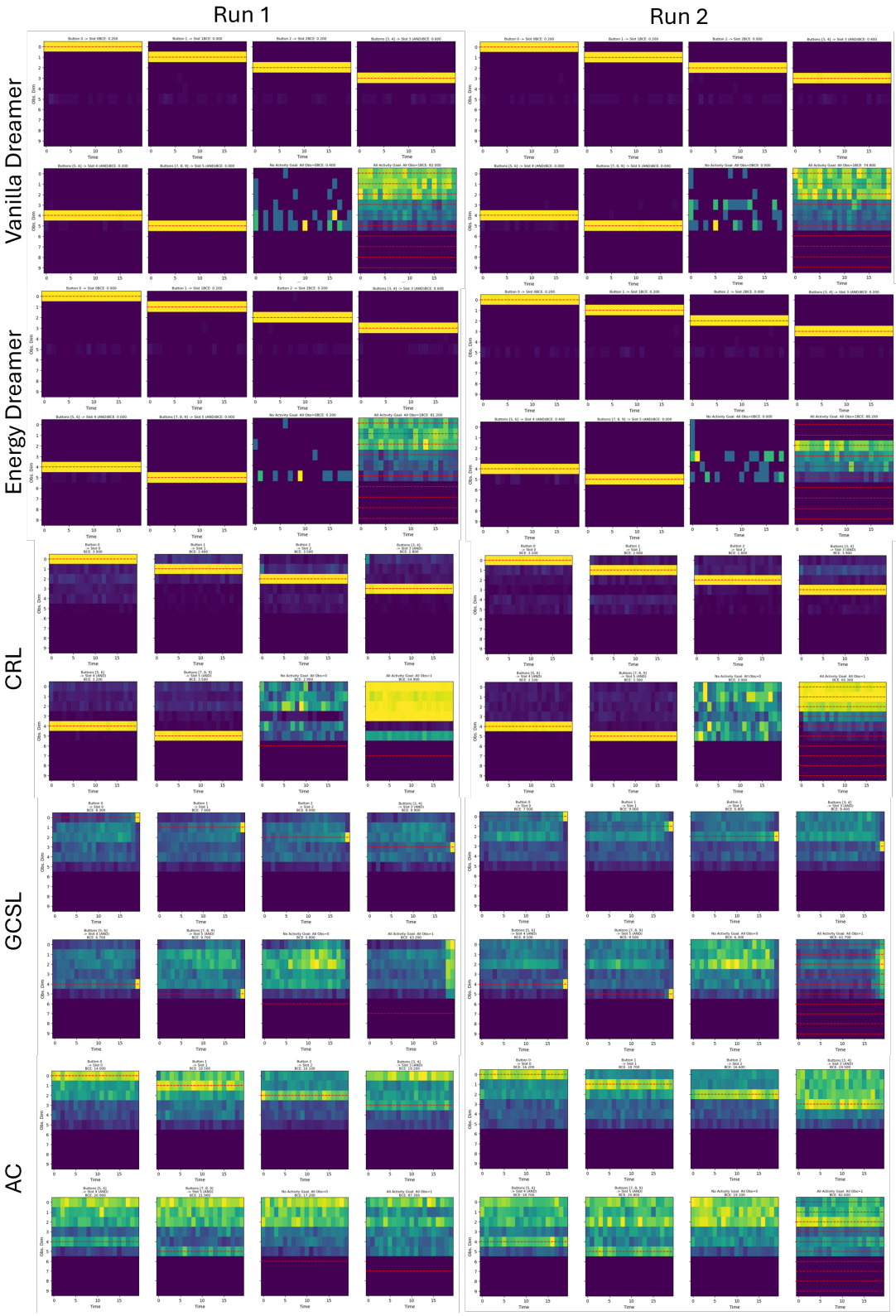

Figure 11: Trajectories of Direct Rules

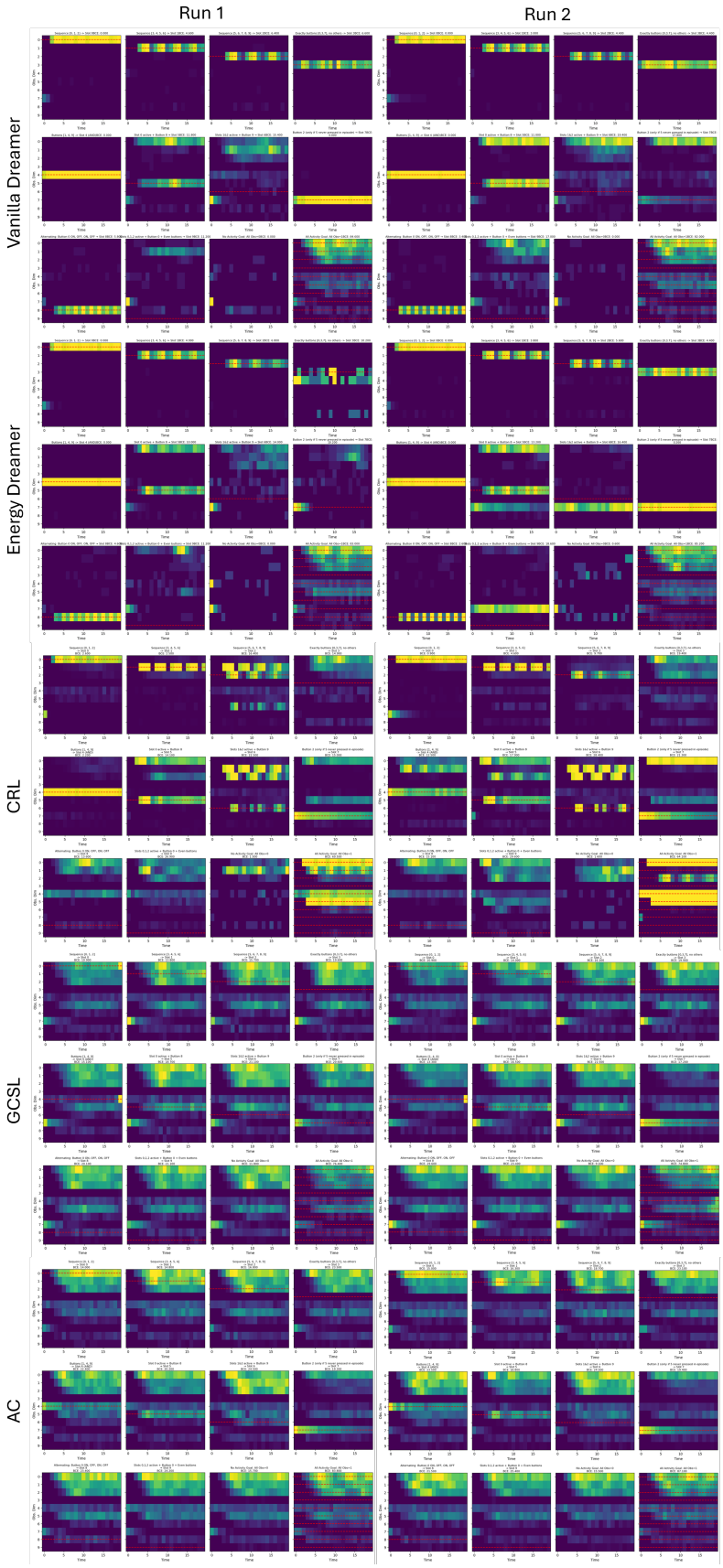

Figure 12: Trajectories of Hard Rules

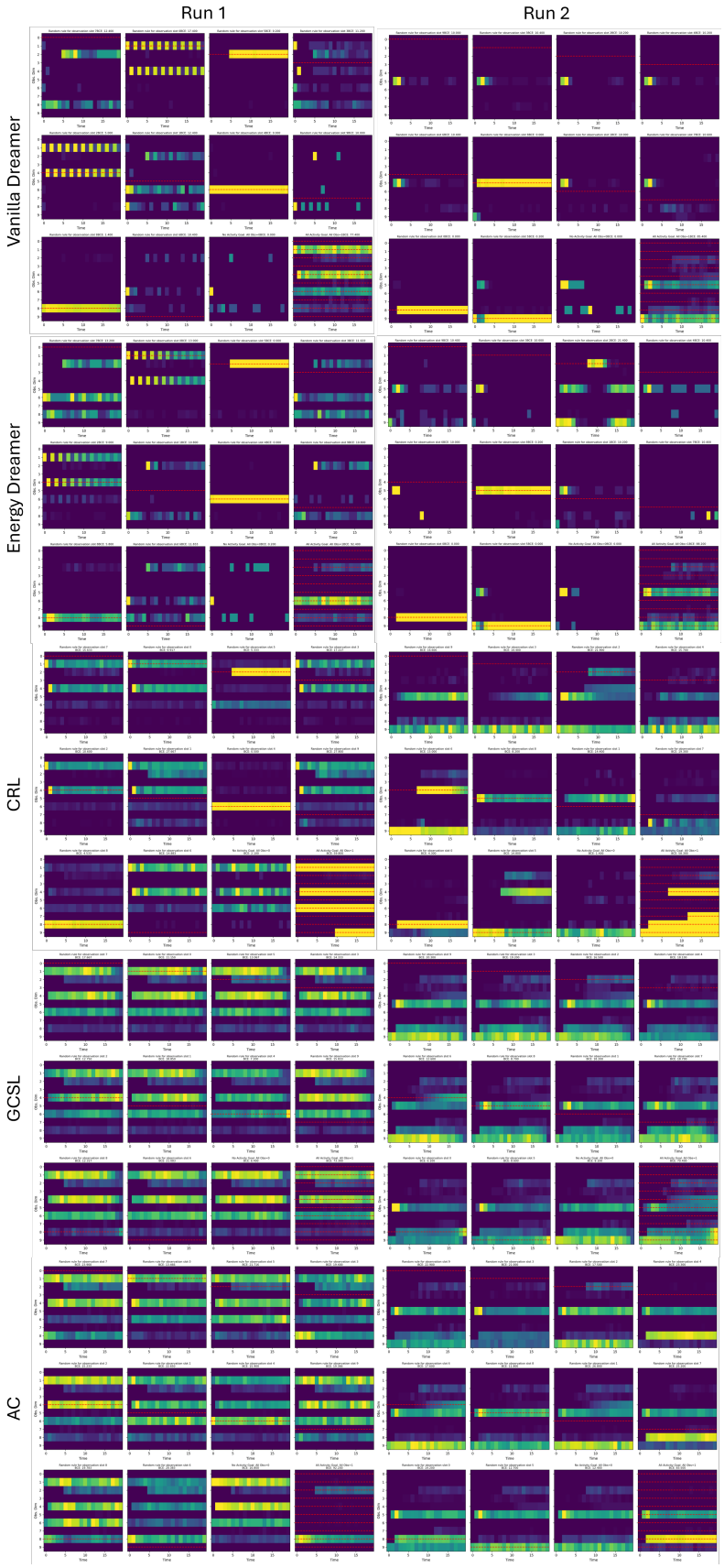

Figure 13: Trajectories of Random Rules

D Dreamer Self-Learning

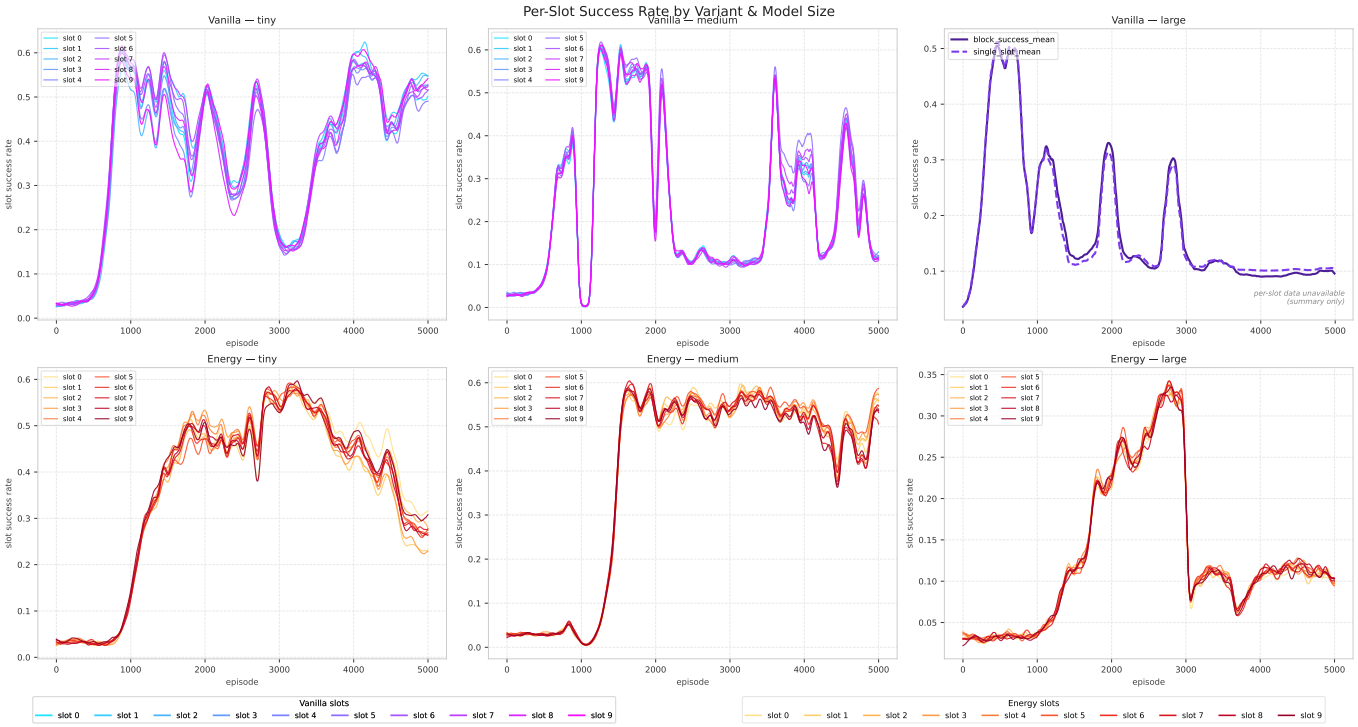

Figure 14: Per-Slot Success Rate of Self-Learning Dreamer Models during training. Interestingly, larger models tend to perform worse.

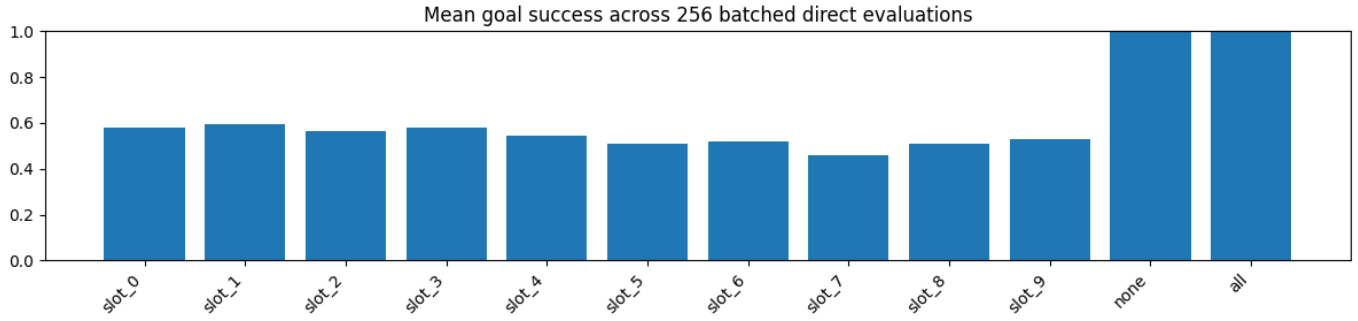

Figure 15: Mean Goal Success Rate for the vanilla-tiny model in evaluation

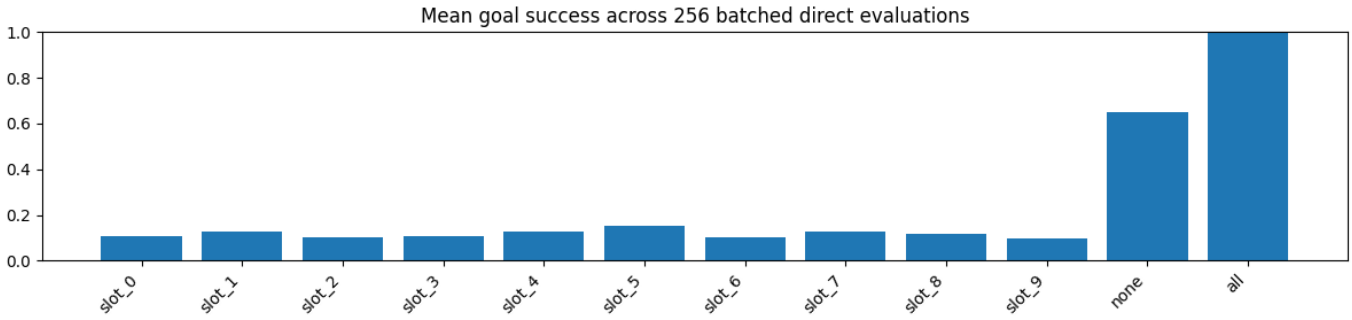

Figure 16: Mean Goal Success Rate for the vanilla-medium model in evaluation

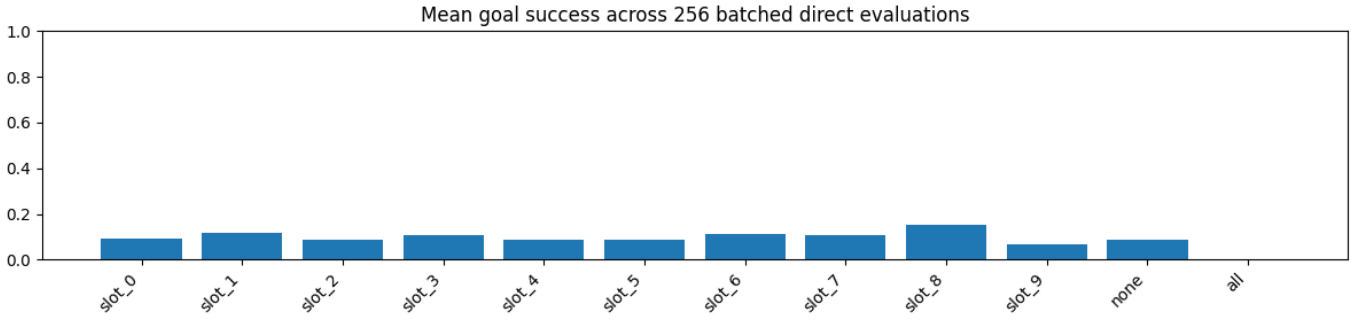

Figure 17: Mean Goal Success Rate for the vanilla-large model in evaluation

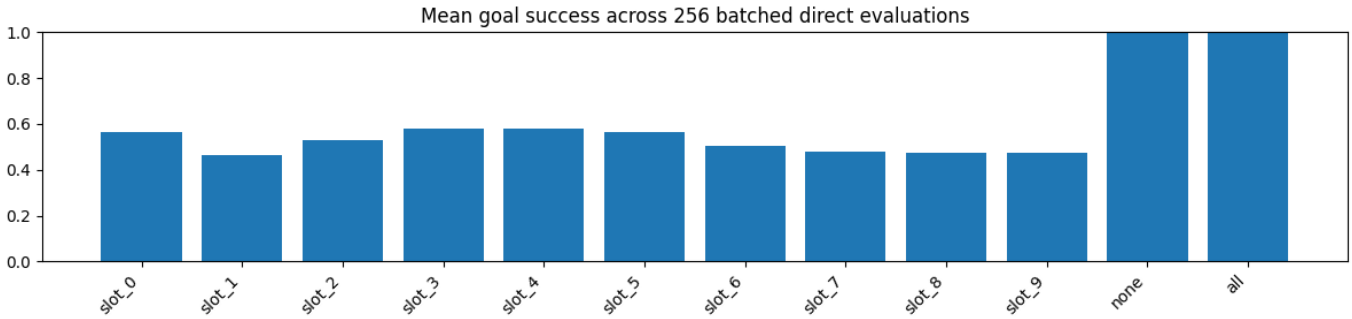

Figure 18: Mean Goal Success Rate for the energy-medium model in evaluation

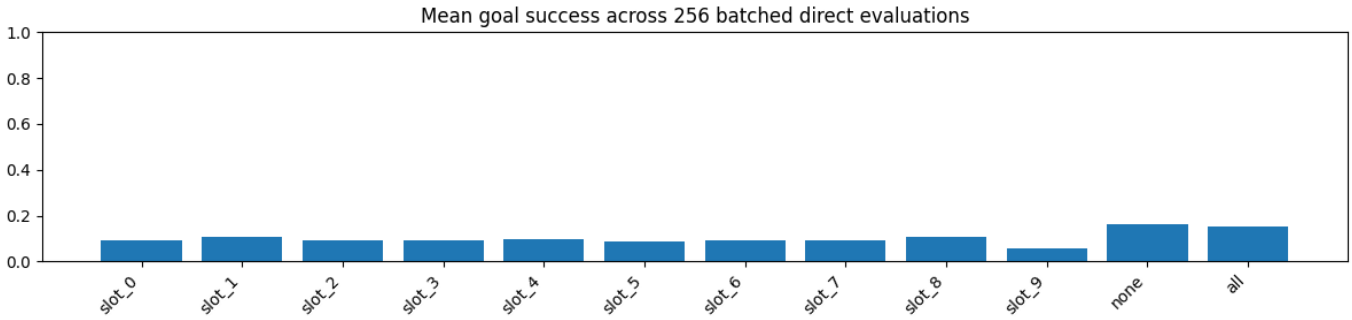

Figure 19: Mean Goal Success Rate for the energy-large model in evaluation

E Hyperparameters

E.1 Curriculum Learning

Table 5: Hyperparameter configurations for the self-learning model used in curriculum learning. The curriculum level advances when the average single-slot success rate over the most recent steps — as defined by the advance window — exceeds the upper threshold, or when the curriculum budget is exhausted and the lower threshold has been met. For experimental purposes, the lower threshold was set to zero throughout all experiments.

Hyperparameter	Value
Architecture	
lat	32
cat	4
det	512
Total Parameters	1M
Training	
Learn steps	256
Answer steps	40
Unroll steps	32
Curriculum Learning	
Curriculum upper threshold	0.7
Curriculum lower threshold	0.0
Advance window	30
Curriculum budget	5000