Joint Learning of Hierarchical Neural Options and Abstract World Model

Wasu Top Piriyakulkij[†1] **Wolfgang Lehrach**[2]
Kevin Ellis[*1] **Kevin Murphy**[*2]
Cornell University[1] Google DeepMind[2]

Abstract

Building agents that can perform new skills by composing existing skills is a long-standing goal of AI agent research. Towards this end, we investigate how to efficiently acquire a sequence of skills, formalized as hierarchical neural options. However, existing model-free hierarchical reinforcement algorithms need a lot of data. We propose a novel method, which we call AgentOWL (**O**ption and **W**orld model **L**earning Agent), that jointly learns—in a sample efficient way—an abstract world model (abstracting across both states and time) and a set of hierarchical neural options. We show, on a subset of Object-Centric Atari games and a simulated robotic block stacking environment, that our method can learn more skills using less data than baseline methods and possesses learning and generalization capabilities that the baselines do not have.

1 Introduction

For decision-making agents, an important goal is the cumulative acquisition of new skills, in tandem with an ever-expanding knowledge of how those skills affect the outside world. For example, we want our agents to first learn to pick up objects, then to pour drinks, and eventually to make a cup of coffee, while also learning how each skill affects the outside world, so that the agent can plan to achieve new goals such as getting coffee for a room full of people.

We formalize this compositional skill learning using the options framework [1]: an agent learns a sequence of options $o_{1:n}$ that achieve increasingly difficult goals, $g_{1:n}$. Each option contains a policy that achieves its specific goal, and which can use options learned earlier, forming a deep hierarchy of skills (Figure 1 right).

But hierarchical options are challenging to learn, because as we acquire more options, we effectively expand our action space, making policy learning less tractable whenever we face with a new goal. This introduces a tradeoff between learning new skills quickly, and how many skills we have acquired. Applications of standard model-free RL to learning option hierarchies therefore require increasingly more samples as learning progresses [3, 4].

To resolve this tradeoff, we instead turn to model-based reinforcement learning [6, 7]. By modeling the effects of options and planning in that world model, we can rule out many options before trying them out in the real world, effectively using the world model to improve sample efficiency. Moreover, modeling option effects produces temporally abstract world models, overcoming the "one-step trap" [8, 9] and promising more tractable planning than low-level world models. However, for this approach to actually improve sample efficiency, we also need a world modeling approach that is data-efficient.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

[†]Work done during an internship at Google DeepMind.
[*]Equal advising.

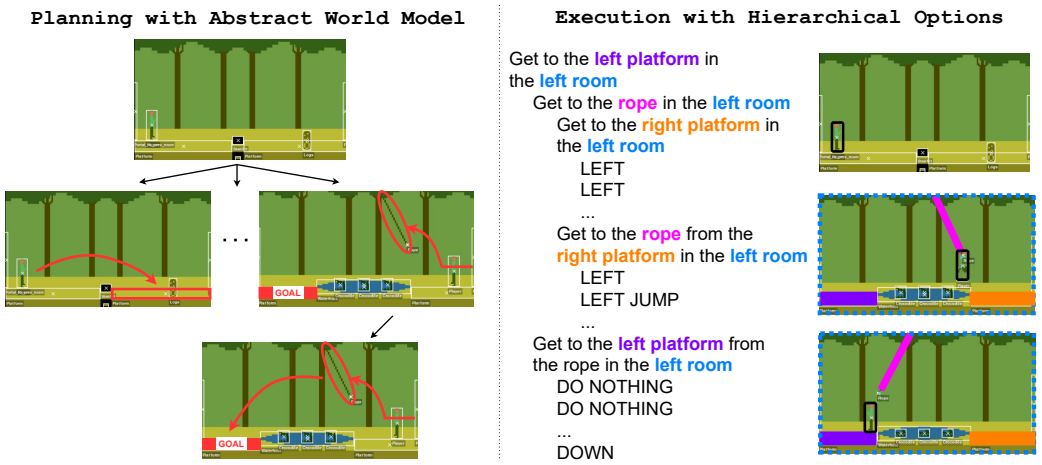

Figure 1: Illustration of hierarchical planning and execution in AgentOWL where the goal is to go to the left platform in the left room. Left: AgentOWL goes through possible plans and successfully makes a short plan (two high level steps) in its abstract world model to reach the goal. Right: AgentOWL executes a hierarchical sequence of options. (The hierarchical structure is represented by the indentation.)

In this work, we propose a novel world model whose representation combines symbolic code with non-parametric distributions, which allows learning the world model from little data. The world model abstracts over states and time. We combine this with a method to learn hierarchical options. We call the resulting system AgentOWL, which stands for **O**ption and **W**orld model **L**earning Agent.

We apply our method to 3 hard object-centric Atari (OCAtari) games, namely Montezuma's Revenge, Pitfall, and Private Eye and also a simulated robotic block stacking environment. We show that AgentOWL acquires the highest number of skills compared to other baselines. We also show that AgentOWL has unique capabilities that the baselines lack, namely implicit, hierarchical learning of sub-options and zero-shot generalization to novel situations.

2 Background

Problem Setting. An environment can be described as a goal-conditioned MDP $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{G}, \gamma)$. We assume states can be broken down into primitive features $\mathcal{S} = S_1 \times ... \times S_n$, meaning the state is symbolic, allowing us to focus on the skill learning problem without mixing in the well-known challenges of representation learning of pixel inputs. The set $\mathcal{A}$ enumerates primitive actions, e.g., LEFT, RIGHT, UP, etc. in 2d video games. The goals $\mathcal{G} = (g_1, g_2, \cdots, g_m)$ are an ordered sequence of goal predicates, $g_i : \mathcal{S} \to \{0, 1\}$, each of which defines a reward function $R_{g_i}(\boldsymbol{s}, a, \boldsymbol{s}') = g_i(\boldsymbol{s}')$. We use the same discount factor γ for all goals, and assume that the transition function $\mathcal{T}$ does not depend on the goal. Episodes end when an agent reaches the goal or timeouts.

Options (Figure 1 right). An option is a learned skill. Formally, option o_i comprises a tuple (π_i, g_i) of a policy π_i which executes until its goal, g_i, is satisfied. The goal g_i serves as the *termination condition* of the option. We follow the call-and-return paradigm [1]; an option executes until its goal is satisfied, or it timeouts. Options may form a hierarchy: Option o_i has policy $\pi_i : \mathcal{S} \to \mathcal{A} \cup \{o_j\}_{j<i}$, meaning it can output either primitive actions in $\mathcal{A}$ or a previously learned option o_j (for $j < i$). We write Ω for a set of options. Adding options to the action space of an MDP forms a Semi-MDP [10].

State abstraction (Figure 1 left). An option can change the state in complex ways, and for the agent to plan, it must predict those changes. To make this prediction problem tractable, we consider *state abstractions*, which are functions of the state that elide unpredictable or irrelevant features that would be hard to predict [11, 12]. Formally, a state abstraction $\boldsymbol{f}(\boldsymbol{s})$ is a function of the state $\boldsymbol{s}$. When the state is clear from context, we abuse notation by writing $\boldsymbol{f}$ to mean $\boldsymbol{f}(\boldsymbol{s})$.

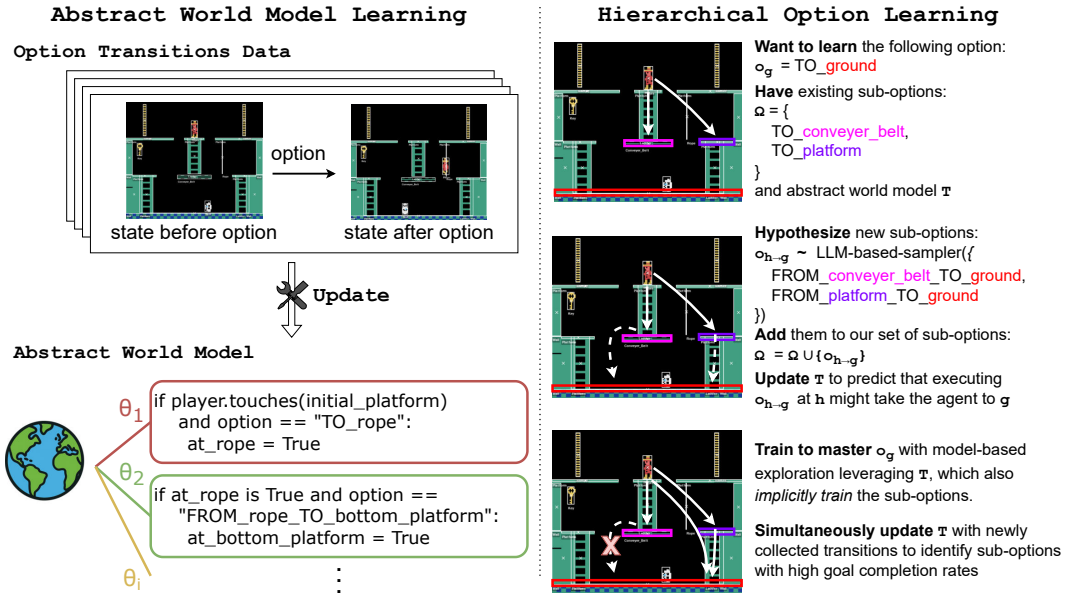

Figure 2: Illustration of how AgentOWL learns its abstract world model (left) and hierarchical options (right). Left: given a dataset of option transitions, we learn an abstract world model using (an extension of) the method of [2]. Specifically, each expert is generated using LLM code synthesis, and the weight for each expert (denoted θ_i) is learned using gradient descent on the likelihood objective. Right: AgentOWL learns a new option to achieve a new goal by leveraging previously acquired sub-options. Specifically, given the goal, we ask an LLM to hypothesize new sub-options (building on the already learned ones) that might help achieve the goal. The agent trains to master the highest-level option, o_g, which implicitly trains these new sub-options, and simultaneously updates our abstract world model to identify sub-options with high goal completion rates. Eventually, the agent masters the target option o_g by composing good sub-options. Detailed pseudocode can be found in Algorithm 1.

Abstract world models (Figure 1 left). Within the context of this work, an abstract world model predicts future abstract states, given the current state, and the current option. This implements temporal abstraction *and* state abstraction, because rather than predicting the immediate next state, we instead predict only its abstract features, and only at the time that the current option terminates. This prediction is written $p_o(\boldsymbol{f}' \mid \boldsymbol{s})$. This conditions the abstract world model on the full state but predicts only the future abstract state.

PoE-World (Figure 2 left). Piriyakulkij et al. [2] introduces PoE-World, a framework for learning structured world models from little data. World models are represented using a product-of-experts, where each expert is a short symbolic program. Intuitively, each program models an independent causal mechanism in the world, and by encoding each program as a snippet of Python, they become learnable using LLMs. Given current state $\boldsymbol{s}$ and action a, the next state $\boldsymbol{s}'$ follows

$$p_{\boldsymbol{\theta}}(\boldsymbol{s}'|\boldsymbol{s},a) = \prod_j p(s'_j|\boldsymbol{s},a) \quad (1)$$

$$p(s'_j|\boldsymbol{s},a) = \frac{1}{Z_j}\prod_{i:j(i)=j} p_i(s'_j|\boldsymbol{s},a)^{\theta_i}. \quad (2)$$

where $j(i)$ is the target feature dimension modeled by expert i. Learning with PoE-World means generating experts p_i with LLMs and estimating weights $\boldsymbol{\theta}$, which requires little data (few $(\boldsymbol{s},a,\boldsymbol{s}')$ triples) because it is not learning a fully parametric model. The model assumes the state features are conditionally independent; this makes it tractable to compute the partition function, Z_j, and hence we can perform maximum likelihood estimation (MLE) of the weights through gradient descent. On object-centric Atari (OCAtari) [13], PoE-World takes only a few minutes of gameplay to assemble a working world model. We use PoE-World to learn an abstract world model.

Algorithm 1 AgentOWL learning algorithm

Given a target goal to achieve g, a list of sub-options Ω and corresponding policies $\pi_\Omega = \{\pi_o\}_{o \in \Omega}$, a current abstract world model $T = \{p_o\}_{o \in \Omega}$, a set of seen option transitions $D_\Omega = \{D_o\}_{o \in \Omega}$, an environment E, and stabilization thresholds n (number of samples) and δ (goal completion rate).
$\pi_g^{real} \leftarrow InitPolicy(E, \Omega)$ ▷ Initialize policy
$\epsilon \leftarrow 1$ ▷ Initialize ϵ which will anneal over time to 0
while o_g for goal g is not mastered **do**
  $\Omega_g \leftarrow \{o : g_o = g \mid o \in \Omega\}$ ▷ Set of options whose corresponding goals match the target goal
  $\Omega_{stable-g} \leftarrow \{o : n_o > n \text{ or } \delta_o > \delta \mid o \in \Omega_g\}$ ▷ Subset of Ω_g with only "stable" sub-options
  $\Omega_{good-g} \leftarrow \{o : \delta_o > \delta \mid o \in \Omega_g\}$ ▷ Subset of Ω_g with only good, high-performing options
  # If all relevant options are stable but none are good enough, then generate new sub-goal
  if $|\Omega_g| = |\Omega_{stable-g}|$ and $|\Omega_{good-g}| = 0$ **then**
    $h \leftarrow LLM(g, \Omega, D_\Omega)$ ▷ Sample a precondition with LLM
    $\Omega \leftarrow \Omega \cup \{o_{h \to g}\}$ ▷ Add a new sub-option $o_{h \to g}$ to Ω
    $T \leftarrow T \cup \{p_{o_{h \to g}}\}$ ▷ Add the corresponding hypothetical option model $p_{o_{h \to g}}$ to T
    $\pi_g^{real} \leftarrow InitPolicy(E, \Omega)$ ▷ Reinitialize the policy as Ω changes
  end if
  $\pi_g^{wm} \leftarrow DQN(T, \Omega, R_g)$ ▷ Learn π_g^{wm} from scratch in the abstract world model T with Ω.
  # Train hierarchically to master an option for goal g in E.
  # This may change any $\pi \in \pi_\Omega$, in addition to π_g.
  $((\pi_g^{real}, \pi_g^{wm}, \epsilon), \pi_\Omega, D_\Omega) \leftarrow HierarchicalDQN(E, \Omega, D_\Omega, \pi_\Omega, g, n, \delta, \pi_g = (\pi_g^{real}, \pi_g^{wm}, \epsilon))$
  $T \leftarrow PoEWorld(T, D_\Omega)$ ▷ Fit T with the updated D_Ω using PoE-World
end while
$o_g \leftarrow ((\pi_g^{real}, \pi_g^{wm}, \epsilon), g)$
return o_g, Ω, T, D_Ω ▷ Return the target option o_g and the updated Ω, T, D_Ω

3 Method

We propose AgentOWL (**O**ption and **W**orld model **L**earning Agent), an agent that sample-efficiently learns a sequence of options $\{o_i\}$, given a goal-conditioned MDP and a sequence of goals $\{g_i\}$. We describe our abstract world modeling approach embedded in AgentOWL in Section 3.1 and then the full AgentOWL in Section 3.2.

3.1 Abstract World Modeling

How should we abstract the state in order to reason about the effects of options? Each goal predicate must be in the state abstraction in order to successfully capture how each option transforms the state. In principle, further predicates may be important to include so that the abstract state is sufficiently informative, but recall that the abstract world model conditions on the current state $\boldsymbol{s}$, so any further features can still be extracted from the current state. We therefore define a state abstraction using just the goal predicates:

$$\boldsymbol{f}(\boldsymbol{s}) = (g_1(\boldsymbol{s}), g_2(\boldsymbol{s}), g_3(\boldsymbol{s}), \ldots) \tag{3}$$

Next, we need an abstract world model that can be used for model-based lookahead. We learn $p_o(\mathbf{f}'|\mathbf{s})$ using PoE-World [2], because by using symbolic programs to represent the abstract dynamics of the world, we can generalize more strongly from fewer examples. Indeed, symbolic rules have long been an attractive representation for modeling coarse-grained world dynamics [14, 15].

But even using symbolic programs, abstract states contain many abstract features, requiring many samples to learn. To maintain sample efficiency, we impose a "frame axiom prior" on the abstract world model, which biases it toward believing that option o_i tends to change f_i (from achieving g_i), but does not usually change f_j for $j \neq i$. The frame axiom prior is implemented by incorporating $p(\theta_i)$ into Equation (2), turning weight optimization into a maximum a posteriori estimation (MAP) instead of MLE. We use Gaussian priors $p(\theta_i) = \mathcal{N}(\mu, \sigma^2)$ with $\sigma = 0.1$ and $\mu = 0.5$ for experts that do not change f_j, and $\mu = 0.001$ for the ones that do. This "frame prior" is commonly used in the planning community, as it is employed, in a much stronger form, in PDDL [15].

PoE-World yields $p_o(\boldsymbol{f}' \mid \boldsymbol{s})$, but learning only this conditional distribution is insufficient because it cannot chain together several options: After running the first option in state $\boldsymbol{s}$, we arrive in $\boldsymbol{f}'$, but predicting the effect of a second option might need the full state $\boldsymbol{s}'$. We heuristically predict $\boldsymbol{s}'$ from

$\boldsymbol{f}'$ using a kernel density estimator $w(\boldsymbol{s}' \mid \boldsymbol{f}')$ that samples full states $\boldsymbol{s}'$ given an abstract state $\boldsymbol{f}'$:

$$p_o(\boldsymbol{s}' \mid \boldsymbol{s}) \approx \mathbb{E}_{\boldsymbol{f}' \sim p_o(\cdot \mid \boldsymbol{s})} \left[w(\boldsymbol{s}' \mid \boldsymbol{f}') \right] \tag{4}$$

Note this is approximate: $\boldsymbol{s}'$ generally depends on $\boldsymbol{s}$ and o, even conditional on $\boldsymbol{f}'$. This approximation is common in the hierarchical decision-making literature, where w is called a *weighting function* [16, 12]. Weighting functions allow sampling states from an abstract state without learning the MDP transition function, and without training a parametric generative model over the raw state space, both of which would require enormous data. To the extent that the environment can be accurately modeled using $p_o(\boldsymbol{f}' \mid \boldsymbol{s})$ where p_o only depends on the state abstraction $\boldsymbol{f}(\boldsymbol{s})$, this approximation becomes exact. Section A.4 contains implementation details of our abstract world modeling approach.

3.2 Joint Learning of Hierarchical Neural Options and Abstract World Model

Using this world modeling setup, we now introduce the full AgentOWL. It iteratively trains the next option to achieve the next goal (and in its world model) by calling Algorithm 1, whose three main ideas are described below.

Model-based exploration. Intuitively, planning in our world model should offer good guidance to a model-free policy; we can weigh trajectories in imagination before deciding what to try in the real world. Concretely, we run RL (specifically, deep Q-learning (DQN)) in the abstract world model yielding a policy π^{wm} (Algorithm 1 line 15). Note that this is computationally cheap, since the abstract world model T takes large steps, and is defined on a fairly low-dimensional symbolic state space, which allows us to use simple MLPs to represent the policy.[1]

The resulting policy, π^{wm}, serves as an exploration policy for training a policy in the real world, π^{real}, that learns to achieve a goal. More precisely, each option comprises a policy and goal, $o = (\pi, g)$, and we further decompose the policy into $\pi = (\pi^{real}, \pi^{wm}, \epsilon)$, where ϵ is the probability of taking exploratory actions (actions the world model predicts):

$$\pi(a \mid \boldsymbol{s}) = (1 - \epsilon)\ \pi^{real}(a \mid \boldsymbol{s}) + \epsilon\ \pi^{wm}(a \mid \boldsymbol{s}) \tag{5}$$

The decomposition ensures we can still learn a good policy even with imperfect world model. By annealing ϵ from 1 to 0, the agent eventually stops relying on π^{wm} and falls back on fully model-free RL learning of π^{real}. The reason we do this is that model-based learning is known to be sensitive to model inaccuracy [17, 18].

We note that for AgentOWL, each policy has its own set of weights; there is no weight sharing between the policies.

Hypothesizing sub-options to achieve a target goal. Planning (or RL) to achieve a new goal is challenging unless we *already* have an option which reaches that goal, which therefore could serve as a sub-option. Absent such sub-options, the agent would need to reason about how its low level actions could be used to reach the new goal, defeating the whole point of a temporally abstract world model. For example, if we have a sub-option to "pick up the cup" and a target goal of "fill the cup with water", a successful plan might first "pick up the cup" followed by a long sequence of low level actions.

To shorten our abstract plans, and help "plan in the now" [19], we let the agent hypothesize new sub-options that aim to achieve the target goal given that certain preconditions are satisfied (see Figure 2 (right) and Algorithm 1 line 10-13). We use LLMs to propose the preconditions, h, of a new option $o_{h \to g}$. This new option, and its corresponding hypothetical option model $p_{o_{h \to g}}$, is then added to the set of options Ω and the abstract world model T respectively.

For this work, we restrict preconditions to the form $h(\boldsymbol{s}) = \boldsymbol{f}(\boldsymbol{s})_i$, representing the completion of the sub-goal with index i. Concretely, we prompt an LLM to pick, among the sub-goals that the agent can already achieve with existing sub-options Ω, a sub-goal that would be useful towards the target goal. In the prompt, we include a sampled state from our set of seen transitions D_Ω to be included as part of the prompt. If the game has multiple rooms, we sample one state for each room (each state contains a "room number" object, so this can be easily done). The exact prompt used can be found in Section A.7.

[1] Note π^{wm} could be computed using a different strategy, such as planning in the world model, rather than RL in the world model. We leave exploring these alternatives for π^{wm} for future work.

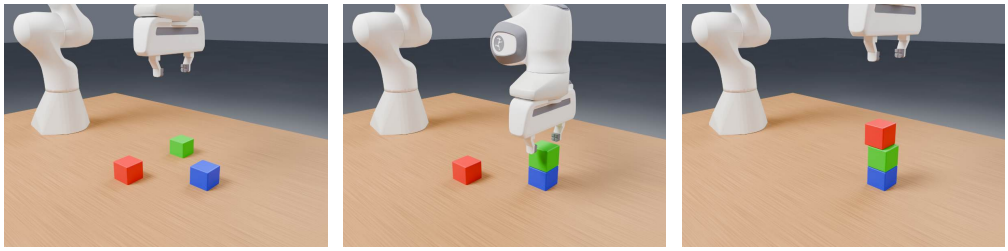

Figure 3: Screenshots from the simulated robotic block stacking domain, showing the initial state and states that achieve a two-block and a three-block stacking goal.

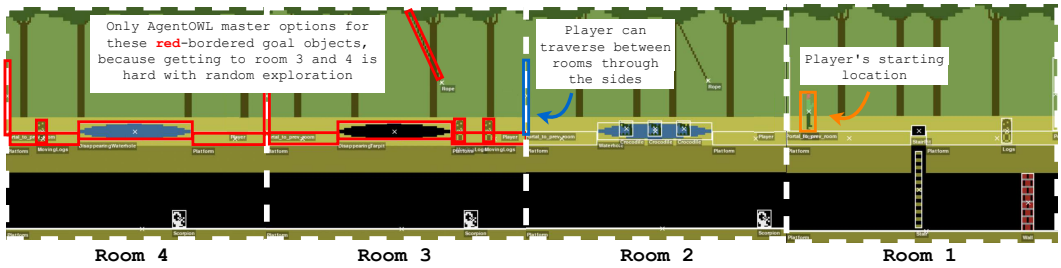

Figure 4: Screenshots of 4 rooms of Pitfall stitched together. Player starts in Room 1 (rightmost) and can traverse to other rooms through the sides of the screen. Goals that only AgentOWL masters within 5M environment steps (Figure 5 middle) are shown in red borders.

Stable training of hierarchical options. Hierarchical option training is done in Algorithm 1 line 18 using a hierarchical version of DQN. It proceeds similarly to typical DQN: executing the policy to collect data in the replay buffer and optimizing the policy using samples from the replay buffer. However, in hierarchical DQN, the execution is hierarchical (Figure 1 right); the agent executes the root-level option, o_g, which then recursively calls sub-options until a primitive action is executed. We also assign each option its own replay buffer to keep the data it collects with its own policy.[2] Each time an option has collected enough new datapoints, the agent optimizes the option's policy weights for a fixed number of steps. Because of hierarchical execution, any sub-option may collect data and have its weights updated.

Nevertheless, hierarchical DQN can be unstable, because each higher-level option faces a non-stationary environment: Training lower level options changes the transition dynamics as seen by higher level options [20]. To mitigate this instability, note that an option's policy stabilizes once it has been trained with enough samples, or it reliably achieves its goals. We therefore disregard episode data for option training which contains an execution of at least one sub-option o with $n_o < n_{threshold}$ and $\delta_o < \delta_{threshold}$, where n_o is the number of samples the option has been trained with, δ_o is the option's goal completion rate over the 100 most recent episodes, and $n_{threshold}, \delta_{threshold}$ are hyperparameters. More details on stable Hierarchical DQN can be found in Section A.1.

4 Experimental Results

Experimental setup. Each agent will be given an ordered sequence of target goals $(g_1, ..., g_n)$. The task for each agent is to "master" a set of neural options that correspond to the target goals, $(o_1, ..., o_n)$. We consider an option "mastered" when the goal completion rate of that option surpasses a threshold $\delta_{threshold}$. For practicality, we approximate the goal completion rate by averaging goal completions over the 100 most recent execution of the option and set $\delta_{threshold} = 0.5$. We evaluate the number of environment steps each agent uses to master this set of neural options.

We conduct our experiments on a subset of object-centric Atari games and a simulated robotic block stacking environment. In both domains, a state is a set of objects with primitive features, and goals

[2]Note that we could be more sample-efficient if we maintain a single shared replay buffer. We leave this for future work.

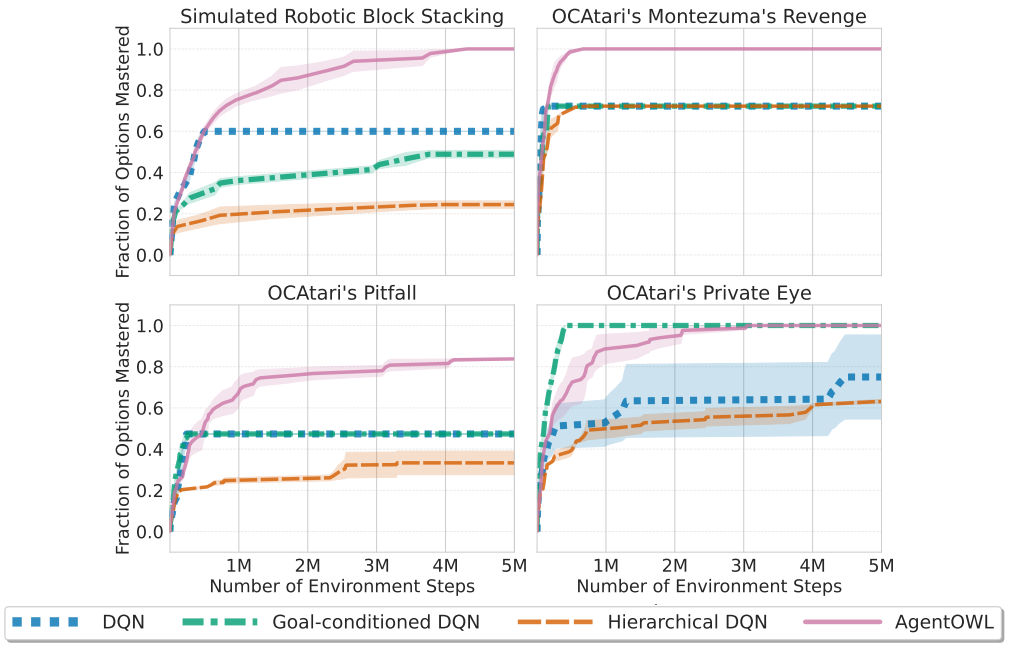

Figure 5: Fraction of options mastered vs number of environment steps for the block stacking domain and the three OCAtari's games we test on: Montezuma's Revenge, Pitfall, and Private Eye. The shaded area represents standard error computed over 3 seeds. Option is acquired once its success rate for the recent episodes reaches threshold $\delta = 0.5$.

are given as a sequence ordered by difficulty (for reasons discussed in Section 6), so that earlier goals serve as stepping stones for later ones.

Domain: Object-centric Atari (OCAtari) [13] provides an object parser on top of Atari games [21], transforming the inputs from pixels to sets of objects. Each object is described by object type (player, platform, ladder, etc.) and bounding box coordinates; we treat these values as primitive features.

We evaluate on the subset of games commonly used to study hard exploration in RL [22, 23, 24], specifically Montezuma's Revenge, Pitfall and Private Eye, selected based on available computational resources.

For each game, we construct a small sequence of goals with the following procedure: We manually select a few rooms in each game, as each may have many rooms, e.g., Pitfall contains 255 rooms. We then define our list of goals as touching each possible object within these selected rooms. Finally, we manually order these goals by difficulty.

Domain: Simulated robotic block stacking (Figure 3) is a simulated tabletop manipulation domain built in PyBullet [25], featuring a robotic arm with a gripper and three cubes (red, green, and blue) on a table. Similar to OCAtari, a state is a set of objects, namely the end effector and the three blocks, each described by its position, discretized to integer, together with a gripper open/closed feature. The agent has nine discrete actions: a no-op, 6 movement actions along $\pm x$, $\pm y$, and $\pm z$, and two gripper actions (open/close).

The goal sequence consists of 15 goals in three tiers: lifting a block (3 goals), stacking one block on another (6 ordered pairs), and building a three-block tower (6 orderings). Unlike the Atari goal sequences, this goal sequence was produced by an LLM prompted with a description of the domain. All methods attempt the goals in this order, but move a goal to the back of the queue if it fails to master the goal within a training budget.

Full details on the experimental setup and domains can be found in Section A.2 and Section A.3.

Baselines. **Rainbow DQN** [26] is an improved version of DQN (Deep Q-Network) [27], a standard off-policy RL algorithm commonly used in discrete action settings. All DQNs used in the paper are

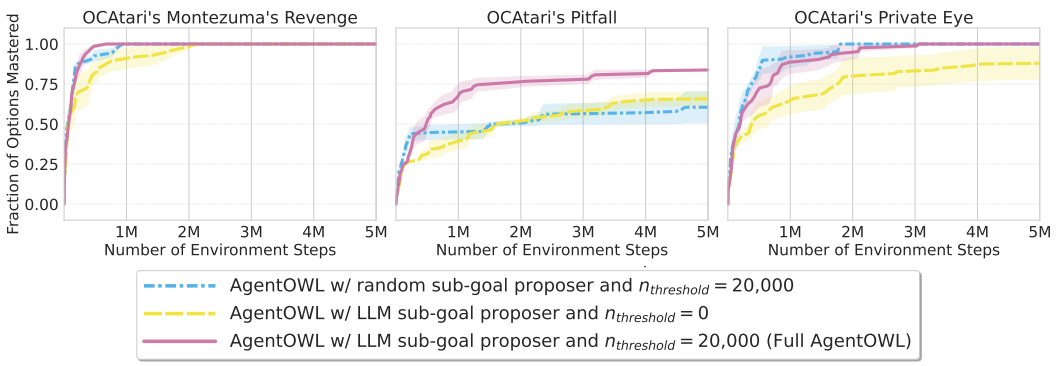

Figure 6: Ablation study: Removing core components of AgentOWL lead to fewer options being acquired and/or more data from the environment being needed. Setting $n_{threshold} = 0$ means stabilization for hierarchical DQN is not implemented.

Rainbow DQNs. **Goal-conditioned DQN** is DQN with weight sharing between the policies of the options. Specifically, instead of learning $\pi_1(a \mid \boldsymbol{s}), ..., \pi_n(a \mid \boldsymbol{s})$, we seek to learn a goal-conditioned policy $\pi(a \mid \boldsymbol{s}, g)$. **Hierarchical DQN** is DQN whose policy has an action space that includes previously learned sub-options. Hierarchical DQN can be seen as AgentOWL without its abstract world model. Implementation details of DQN and Hierarchical DQN can be found in Section A.1.

For the main results, we use Gemini 2.5 Flash as a sub-goal proposer for AgentOWL, but we also rerun AgentOWL with Qwen3.5-9B to show that the method does not require a strong LLM in Section A.5.

Skill acquisition results. As shown in Figure 5, AgentOWL masters the highest number of options for most environment steps. Although the baselines without hierarchical options seem to be better than AgentOWL at lower training sample budget for some environments, their performance plateaus at a much lower number of options mastered compared to that of AgentOWL. Qualitatively, these baselines fail to acquire options for harder goals (Figure 4) because they never discover any action sequence that can achieve the harder goals, as the number of possible action sequences grows exponentially with the number of steps needed to achieve the goals. AgentOWL, on the other hand, manages to accomplish a goal that requires a long, complicated sequence of primitive actions by planning abstractly with higher-level options and executing the options hierarchically. Abstract plans can be very short, allowing them to be found very easily in both the real world and the abstract world model.

We perform an ablation study in Figure 6. Removing techniques introduced in Section 3.2—specifically LLM-based sub-goal proposal and hierarchial training stabilization—degrades AgentOWL's performance across most environment steps. These ablated systems need more data to master the same number of options and in many cases, plateau at fewer mastered options than the full system—with one exception where removing stabilization in Private Eye does not significantly change results. We also show that AgentOWL still outperforms the baselines when the order of goals is randomized in Section A.6.

Implicit, hierarchical learning of sub-options. AgentOWL improves its sub-options through hierarchical DQN even when rewarded only for goals that do not correspond to them. In Figure 7, we perform an experiment to clearly demonstrate this implicit learning. We take an AgentOWL agent trained to achieve a goal sequence in Montezuma's Revenge (Figure 2 left), keep its learned abstract world model, but re-initialize the policy networks of all options with random weights. Then, we solely train this agent to master goal "key", i.e., to touch the key. Figure 7 (right) shows the performances of many re-initialized sub-options increase significantly after AgentOWL is trained to achieve goal "key". We observe that sub-options helpful to the target goal improve, while irrelevant options corresponding to sub-goals outside the successful trajectory (Figure 7 left) do not.

Intuitively, the agent leverages the learned world model to help decide a sequence of sub-goals to pursue to eventually achieve the target goal. This sequence of goals acts as a curriculum for the agent to follow. The agent refines these options to help it eventually achieve the target goal.

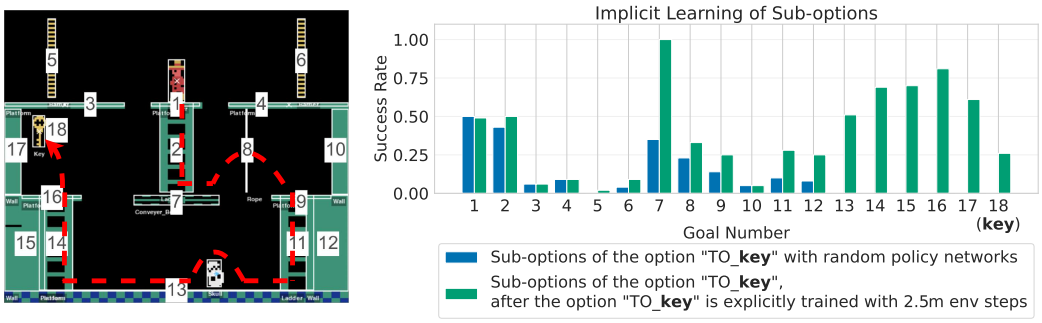

Figure 7: Left: Illustration of the goal labels (where all goals are of the form "touch this specific object") and the path the player needs to take to get the key from the starting position. Right: The success rates of the sub-options implicitly trained while the agent is training for the goal "key", compared to those of sub-options with randomly initialized policy network. The sub-options for the sub-goals within the path to retrieve the key, such as goal #13-17, has high success rates, because they are useful to its parent-option "TO_key". On the other hand, unhelpful goals, such as goal #3-6, remain poorly learned.

Zero-shot generalization to novel situations. Another benefit of having a world model is zero-shot generalization to novel situations. Looking forward to more complex domains, we want agents that can flexibly complete goals in novel situations, such as picking up a cup in a novel kitchen, or clearing a randomly generated level in Minecraft or Nethack [28, 29].

Thus, we design two experiments[3] to showcase AgentOWL's zero-shot generalization capabilities. First, we test zero-shot generalization to a novel starting state in Private Eye (Table 1a). In OCAtari, the starting state of each game is always exactly the same. We show that after learning an option to travel from a new starting state back to the game's original starting state, AgentOWL can compose existing options to achieve target goals zero-shot without any additional training data. Second, we test zero-shot compositional generalization to an unseen environment layout in the block stacking domain. Each agent has to stack the red block on the green block. The red and green blocks each start at one of four positions, giving 16 starting layouts, and we hold out one layout from training. AgentOWL learns an option that lifts the red block from every red position and another option that places it on the green block at every green position. Composing the two, AgentOWL succeeds in all 100 evaluation episodes on the held-out layout.

Method	Goal 1	Goal 2	Goal 3
Random	0.06	0.00	0.00
Goal-conditioned DQN	0.00	0.00	0.00
AgentOWL w/o new option	0.02	0.01	0.26
AgentOWL w/ new option	**1.00**	**1.00**	**1.00**

(a) Private Eye

Method	Seen layouts	Unseen layout
DQN	0.96	0.73
AgentOWL	0.98	**1.00**

(b) Block stacking

Table 1: Zero-shot generalization results. (a) Goal completion rates of previously mastered goals over 100 episodes in Private Eye, attempted from a new starting state. We then give a new option to navigate from the new starting state to the initial starting state to AgentOWL, allowing it to achieve the goals from the new starting position without any additional training. Figure 8 displays the initial state, the new starting state, and the goals in this experiment. (b) Goal completion rate over 100 episodes for stacking the red block on the green block. Each method is trained on 15 starting layouts (seen) and tested on one held-out layout (unseen).

[3]We only ran Private Eye and block stacking experiments due to time constraints, but we believe the results for Montezuma's Revenge and Pitfall would be qualitatively similar.

5 Related Work

Abstract and symbolic world models. The advent of LLMs has sparked interest in symbolic world models. WorldCoder and GIF-MCTS [30, 31] directly use LLMs to generate and refine a world model as a program. POMDP Coder [32] and CWM [33] tackle the problem of partial observability. PoE-World [2] and OneLife [34] scale up symbolic world modeling with a product-of-experts world representation.

Like our work, several prior works have explored abstract world modeling with symbolic world representations. DECKARD [35] learns a directed acyclic graph as its abstract world model. Ada [36] synthesizes PDDL as its abstract world model. AgentOWL, on the other hand, learns a stochastic, symbolic world model capable of making both abstract and low-level predictions, offering greater representational power.

Option discovery and hierarchical RL. Option critic methods [38, 39, 40] learn both policies and termination functions of options end-to-end through gradient descent. Skill-chaining [41, 42, 43] focuses on discovering chainable options, where the termination set of an option is the initiation set of another. Other hierarchical RL methods tend to have high-level policies and low-level policies, where the high-level ones either set goals or rewards for the low-level ones [44, 45, 46, 47, 48] or directly select which low-level ones to execute [49, 50, 51]. Our work takes inspiration from these prior works but differs in how we learn deeply hierarchical options, as opposed to the common two-level hierarchy of policies.

LLMs for RL. Existing works have explored the use of LLMs in RL agents in many ways, including assisting with reward design [52, 53, 54, 55], serving as policies or policy generators [56, 57, 58], and producing high-level plans [59, 60, 61, 62]. AgentOWL leverages LLMs for world modeling and sub-goal proposal, but can also benefit from LLM-based reward design if reward functions are not provided. Importantly, AgentOWL learns neural policies through environmental interaction rather than relying on LLMs to directly output action sequences or policies.

6 Limitations and Future Direction

While AgentOWL efficiently learns abstract world models and hierarchical neural options in our setting, there are limitations. First, we assume the given sequence of goals is ordered by difficulty. What made AgentOWL effective is the ability to use sub-options to help achieve a hard goal. Consequently, AgentOWL can fail to achieve a challenging goal without first learning to accomplish its prerequisite sub-goals. In the future, we hope to use ideas from curriculum learning [63] to automate this.

Second, we assume the number of goals is relatively small (< 100). While AgentOWL's model-based exploration keeps environment interactions from scaling linearly with the number of goals, training compute still does. Thus, currently, we reduce the number of environment samples at the expense of increasing compute, which is the right tradeoff only when compute is cheaper than data. Incorporating option affordances [64, 65] to reduce the number of applicable options could be a fruitful direction.

Lastly, we assume symbolic input to the abstract world model, using OCAtari instead of pixel-level Atari. Symbolic input permits learning world models like PoE-World, which is much more sample-efficient than learning a pixel-level world model. There is ongoing effort to learn abstract symbolic world models directly from pixels [66, 67, 68], but merging that line of work with option training remains open. Purely neural world models have made great strides [69, 70], but do not learn explicit symbolic abstractions that can be used to reason over long horizons. Leveraging these purely neural models to efficiently learn neurosymbolic world models offers another path forward.

Limitations aside, AgentOWL's successful results emphasize that option and world model learning are deeply intertwined, and that skills are fundamental to an agent's understanding of its environment. We hope this insight draws attention to several underexplored research problems. For example, if we allow goals and corresponding skills to grow over time, our abstract model faces an ever-changing state and action space. How do we perform efficient world modeling in this online learning setting? With large sets of abstract features and skills, how do we reason over only relevant ones to save computation? Answering these questions could make systems like AgentOWL much more powerful.

Acknowledgments

Kevin Ellis and Wasu Top Piriyakulkij acknowledge support from an NSF CAREER grant.

References

[1] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[2] Wasu Top Piriyakulkij, Yichao Liang, Hao Tang, Adrian Weller, Marta Kryven, and Kevin Ellis. Poe-world: Compositional world modeling with products of programmatic experts. *Advances in Neural Information Processing Systems*, 2025.

[3] Anand Kamat and Doina Precup. Diversity-enriched option-critic. *arXiv preprint arXiv:2011.02565*, 2020.

[4] Andrei Nica, Khimya Khetarpal, and Doina Precup. The paradox of choice: Using attention in hierarchical reinforcement learning. *arXiv preprint arXiv:2201.09653*, 2022.

[5] Marwa Abdulhai, Dong-Ki Kim, Matthew Riemer, Miao Liu, Gerald Tesauro, and Jonathan P How. Context-specific representation abstraction for deep option learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 5959–5967, 2022.

[6] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[7] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 16(1):1–118, 2023.

[8] Richard S. Sutton. The oak architecture: A vision of superintelligence from experience. Invited talk at the Conference on Neural Information Processing Systems (NeurIPS), December 2025.

[9] Kavosh Asadi, Dipendra Misra, Seungchan Kim, and Michel L Littman. Combating the compounding-error problem with a multi-step model. *arXiv preprint arXiv:1905.13320*, 2019.

[10] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York, 1994.

[11] Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *AAAI/IAAI*, pages 106–111, 1997.

[12] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. *AI&M*, 1(2):3, 2006.

[13] Quentin Delfosse, Jannis Blüml, Bjarne Gregori, Sebastian Sztwiertnia, and Kristian Kersting. Ocatari: Object-centric atari 2600 reinforcement learning environments. *arXiv preprint arXiv:2306.08649*, 2023.

[14] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[15] Drew M McDermott. The 1998 ai planning systems competition. *AI magazine*, 21(2):35–35, 2000.

[16] Dimitri Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 01 1995.

[17] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International conference on machine learning*, pages 2829–2838. PMLR, 2016.

[18] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

[19] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477. IEEE, 2011.

[20] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[21] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[22] Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando De Freitas. Playing hard exploration games by watching youtube. *Advances in neural information processing systems*, 31, 2018.

[23] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

[24] Ionel-Alexandru Hosu and Traian Rebedea. Playing atari games with deep reinforcement learning and human checkpoint replay. *arXiv preprint arXiv:1607.05077*, 2016.

[25] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

[26] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[28] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *Ijcai*, volume 16, pages 4246–4247, 2016.

[29] Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

[30] Hao Tang, Darren Yan Key, and Kevin Ellis. Worldcoder, a model-based LLM agent: Building world models by writing code and interacting with the environment. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[31] Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[32] Aidan Curtis, George Matheos, Nishad Gothoskar, Vikash Mansinghka, Joshua Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Partially observable task and motion planning with uncertainty and risk awareness. *arXiv preprint arXiv:2403.10454*, 2024.

[33] Wolfgang Lehrach, Daniel Hennes, Miguel Lazaro-Gredilla, Xinghua Lou, Carter Wendelken, Zun Li, Antoine Dedieu, Jordi Grau-Moya, Marc Lanctot, Atil Iscen, et al. Code world models for general game playing. *ICLR*, 2026.

[34] Zaid Khan, Archiki Prasad, Elias Stengel-Eskin, Jaemin Cho, and Mohit Bansal. One life to learn: Inferring symbolic world models for stochastic environments from unguided exploration. *arXiv preprint arXiv:2510.12088*, 2025.

[35] Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling. In *International Conference on Machine Learning*, pages 26311–26325. PMLR, 2023.

[36] Lio Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S. Siegel, Jiahai Feng, Noa Korneev, Joshua B. Tenenbaum, and Jacob Andreas. Learning adaptive planning representations with natural language guidance. In *International Conference on Learning Representations (ICLR)*, 2024.

[37] Zergham Ahmed, Joshua B Tenenbaum, Christopher J Bates, and Samuel J Gershman. Synthesizing world models for bilevel planning. *arXiv preprint arXiv:2503.20124*, 2025.

[38] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[39] Anna Harutyunyan, Will Dabney, Diana Borsa, Nicolas Heess, Remi Munos, and Doina Precup. The termination critic. *arXiv preprint arXiv:1902.09996*, 2019.

[40] Saket Tiwari and Philip S Thomas. Natural option critic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5175–5182, 2019.

[41] George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems*, 22, 2009.

[42] Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2019.

[43] Akhil Bagaria, Jason Senthil, Matthew Slivinski, and George Konidaris. Robustly learning composable options in deep reinforcement learning. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, 2021.

[44] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

[45] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[46] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International conference on machine learning*, pages 3540–3549. PMLR, 2017.

[47] Siyuan Li, Rui Wang, Minxue Tang, and Chongjie Zhang. Hierarchical reinforcement learning with advantage-based auxiliary rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

[48] Danijar Hafner, Kuang-Huei Lee, Ian Fischer, and Pieter Abbeel. Deep hierarchical planning from pixels. *Advances in Neural Information Processing Systems*, 35:26091–26104, 2022.

[49] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

[50] Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

[51] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[52] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. *arXiv preprint arXiv:2303.00001*, 2023.

[53] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[54] Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint arXiv:2310.00166*, 9 2023.

[55] Roger Creus Castanyer, Faisal Mohamed, Pablo Samuel Castro, Cyrus Neary, and Glen Berseth. Arm-fm: Automated reward machines via foundation models for compositional reinforcement learning. *arXiv preprint arXiv:2510.14176*, 2025.

[56] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[57] 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. *arXiv preprint arXiv: Arxiv-2305.16291*, 2023.

[58] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

[59] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[60] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022.

[61] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

[62] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2998–3009, 2023.

[63] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009.

[64] Khimya Khetarpal, Zafarali Ahmed, Gheorghe Comanici, David Abel, and Doina Precup. What can i do here? a theory of affordances in reinforcement learning. In *International Conference on Machine Learning*, pages 5243–5253. PMLR, 2020.

[65] Khimya Khetarpal, Zafarali Ahmed, Gheorghe Comanici, and Doina Precup. Temporally abstract partial models. *Advances in Neural Information Processing Systems*, 34:1979–1991, 2021.

[66] Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B Tenenbaum, Tom Silver, João F Henriques, and Kevin Ellis. Visualpredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. *arXiv preprint arXiv:2410.23156*, 2024.

[67] Yichao Liang, Dat Nguyen, Cambridge Yang, Tianyang Li, Joshua B Tenenbaum, Carl Edward Rasmussen, Adrian Weller, Zenna Tavares, Tom Silver, and Kevin Ellis. Exopredicator: Learning abstract models of dynamic worlds for robot planning. *arXiv preprint arXiv:2509.26255*, 2025.

[68] Ashay Athalye, Nishanth Kumar, Tom Silver, Yichao Liang, Jiuguang Wang, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. From pixels to predicates: Learning symbolic world models via pretrained vision-language models. *arXiv preprint arXiv:2501.00296*, 2024.

[69] Philip J Ball, Jakob Bauer, Frank Belletti, B Brownfield, A Ephrat, S Fruchter, A Gupta, K Holsheimer, A Holynski, J Hron, et al. Genie 3: A new frontier for world models. *Google DeepMind Blog*, pages 253–279, 2025.

[70] Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in atari. In *Thirty-eighth Conference on Neural Information Processing Systems*, 2024.

[71] Ronald Edward Parr. *Hierarchical control and learning for Markov decision processes*. University of California, Berkeley, 1998.

[72] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

[73] Akhil Bagaria, Jason K Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *International conference on machine learning*, pages 521–531. PMLR, 2021.

[74] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[75] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[76] Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In *International Conference on Machine Learning*, pages 23190–23211. PMLR, 2023.

[77] Clare Lyle, Zeyu Zheng, Khimya Khetarpal, Hado van Hasselt, Razvan Pascanu, James Martens, and Will Dabney. Disentangling the causes of plasticity loss in neural networks. *arXiv preprint arXiv:2402.18762*, 2024.

[78] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. In *International Conference on Learning Representations*, 2018.

[79] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[80] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer, 1999.

[81] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[82] Roger Creus Castanyer, Johan Obando-Ceron, Lu Li, Pierre-Luc Bacon, Glen Berseth, Aaron Courville, and Pablo Samuel Castro. Stable gradients for stable learning at scale in deep reinforcement learning. *arXiv preprint arXiv:2506.15544*, 2025.

[83] Xi-Lin Li. Preconditioned stochastic gradient descent. *IEEE transactions on neural networks and learning systems*, 29(5):1454–1466, 2017.

A Technical appendices

A.1 DQN and Hierarchical DQN

Here we describe our implementation of Rainbow DQN [26] which is used in all baselines, including AgentOWL itself.

Neural network architecture. Rainbow DQN uses dueling network architecture [74] We use a 2-layer MLP as our feature extractor module. The hidden feature size is 256 for π^{real} and 128 for π^{wm} for both layers (so always 256 for baseline methods). Each linear layer is followed by a layer normalization (LN) layer [75], as recent studies found incorporating LN in the network of a deep RL algorithm to be highly beneficial [76, 77], and then a ReLU layer. After the feature extractor, the value and advantage layer is also 2-layer MLP with hidden feature size half of what is used in the feature extractor, but the linear layers are replaced with noisy linear layers [78], and LN and ReLU are only applied the first layer, since the last layer produces output.

For goal-conditioned DQN, we also learn an embedding for each goal and concatenate it to the input to the network.

Training. We base our implementation of (Rainbow) DQN and hierarchical DQN off of Stable Baseline's implementation [79]. The pseudocode for DQN is and hierarchical DQN is at Algorithm 2 and Algorithm 3 respectively, with Algorithm 4 and Algorithm 5 describing the helper functions used in hierarchical DQN. The common hyperparameter for all variants of DQN is listed at Table 4. There are extra hyperparameters for hierarchical DQN Table 5 and overridden hyperparameters for training π_g^{wm} Table 6.

Goal heuristics. Additionally, to speed up training, we have a heuristic for each environment, described in detail in later sections. We incorporate this heuristic h in our Q network as follows:

$$Q_\psi^{new}(s,a) = Q_\psi(s,a) + h_g(s) \tag{6}$$

Intuitively, a freshly initialized neural network typically outputs values around 0. Adding $h_g(s)$ to it makes Q_{new} outputs values around $h_g(s)$ instead. Over time, as we train Q^{new} more and more, the neural network learns to correct the heuristics value so that the output is close the true Q value [80].

A.2 Experimental setting and OCAtari details

Note that the Arcade Learning Environment (ALE) [81] code uses GPL-2.0 license, and OCAtari code [13] uses MIT license.

Goal sequences for each game. For each game, we select a few rooms and define goals as touching objects within them. Then, we manually order the goals based on difficulty. Additionally, we give short natural language names to each goal, such as 'top_left_plat', 'mid_right_wall', etc. We display the screenshots of the rooms with goal objects labeled with their order in the goal sequence at Figure 9, Figure 10, and Figure 11. We note that the orders within each room in Pitfall and Private Eye tend not to matter too much as they are roughly the same in difficulty level.

Goal heuristics. For each game, we use a heuristic based on the Manhattan distance between the player object and the goal object. Specifically, we let the heuristic function h be $h(s) = 5 \cdot (1 - \frac{ManhattanDist(s_{player}, s_{g-object})}{400})$. In OCAtari, Manhattan distance between any two objects never exceeds 400. If the goal object is not visible in s, e.g., the goal object is in another room in the game, we let the heuristics value equal to 0.

Vectorizing list of objects. We vectorize each input object list into a fixed-size feature vector for the DQN policy. By using the maximum count for each object type, we assign each object to a unique location in the observation space. For example, with one player and up to two enemies, where each object is vectorized into a 8-dimensional feature vector, the observation space has size 24: the player occupies indices 0-8, the first enemy occupies indices 9-16, and so on.

To vectorize each object, we take its x and y coordinates and encode each value into a 4-dimensional vector using a positional encoder implemented with sine and cosine functions, commonly used in LLMs.

Additionally, we include the goal values, i.e., the abstract state $\boldsymbol{f}(\boldsymbol{s})$, in the feature vector as well, so the feature vector contains both low-level and abstract features. To better match the dimensionality of the abstract features to that of the low-level features, we duplicate each abstract feature by 4.

Observation space optimization. To optimize running time, we choose to disregard static (non-moving) object types, such as platforms, ladders, etc., from the vector discussed above, as all states have the same information for these objects. This speeds up our code significantly, reducing the low-level feature vector size from $248, 344, 408$ to $24, 176, 328$ for Montezuma's Revenge, Pitfall, and Private Eye, respectively.

Episode timeouts. Since stable hierarchical DQN only adds data to the replay buffer once an episode ends, we need to make sure that episodes do not go on forever. Toward that end, we set a maximum time limit for all episodes in a (real) environment to be $1,000$ environment steps. On the other hand, when we treat our abstract world model as a simulated, abstract environment, we set the time limit to be only 4 environment steps, as this simulated environment is very abstract and only requires very short action sequences to achieve the goal.

A.3 Simulated robotic block stacking experiments details

Environment. The blocks are 5 cm cubes. The robot arm is allowed to move within a 60 cm cube workspace in front of the robot base, with a table at the bottom. Each movement action moves the end-effector by 5 cm along one axis.

Goal heuristics. The heuristic is based on distances between the end effector and the relevant blocks and between the blocks themselves, and is scaled to lie in $[0, 0.5]$.

Vectorizing list of objects We vectorize states as in OCAtari (Section A.2), with one change to the positional encoder. With the encoder used for OCAtari, whose coordinates range over hundreds of pixels, some pairs of positions that are far apart in this 60 cm workspace receive more similar encodings than pairs that are close together. We therefore rescale coordinates by 0.1 and use base 5 instead of $10{,}000$ in the sine and cosine functions, which makes the distance between encodings increase with the distance between positions.

Round-robin goal scheduling with exponential give-up mechanism. All methods in the main block stacking experiment (Figure 5) use an exponential give-up mechanism, with the goals queued in the order generated by the LLM. Each attempt at a goal gets a budget of 250K environment steps times a multiplier that starts at 1. If the goal is not mastered within the budget, it moves to the back of the queue. AgentOWL, hierarchical DQN, and goal-conditioned DQN then revert to a snapshot of the agent taken just before the attempt, so a failed attempt cannot damage previously learned options. DQN, which trains an independent policy for each goal, instead restarts the goal from scratch with a new random seed on its next attempt. The multiplier doubles after a full pass over the queue in which no goal is mastered and resets to 1 whenever a goal is mastered. Because the budget keeps growing, the initial budget is not a fragile hyperparameter: it affects how quickly AgentOWL learns but not its asymptotic performance.

Option resets. Two safeguards keep option training from getting stuck. If an option that has not yet been mastered goes 100K steps without a single success, we re-initialize its network and empty its replay buffer. If a mastered option fails 20 episodes in a row, we restore that option from the most recent snapshot saved after a goal was mastered, the last state in which all options were known to work together.

Neural network architecture. The main block stacking experiment uses the same 2-layer MLP feature extractor as OCAtari (Section A.1). The zero-shot compositional generalization experiment (Table 1b) replaces it with a transformer encoder for both AgentOWL and DQN. Each object in each stacked frame becomes one token, and the abstract features become one more token. Each token

is linearly projected to the hidden size of the MLP, and a learned per-token embedding is added. A learned classification token is prepended to the sequence, which then passes through a 2-layer transformer encoder with 4 attention heads and feed-forward size twice the hidden size. The output at the classification token, after layer normalization, goes to the same dueling value and advantage layers as in the MLP version. Pooling through a learned classification token, rather than averaging all tokens, lets the network ignore objects that are irrelevant to the current goal.

Zero-shot compositional generalization setup. In this experiment, the red block starts at one of four positions with $x \in \{20, 25\}$, $y \in \{15, 20\}$, and the green block also starts at one of four positions with $x \in \{20, 25\}$, $y \in \{40, 45\}$, while the blue block stays at its default position. We hold out one layout, $x = 25$, $y = 20$ for red and $x = 25$, $y = 40$ for green, and let the methods train on the other 15. AgentOWL learns the goal sequence $lift_red$ and red_on_green, while DQN learns the goal red_on_green directly. Compared with the main experiment, we raise the mastery threshold to $\delta = 0.95$, shorten real-environment episodes to 50 steps, lower the learning rate to 5×10^{-5}, enlarge the replay buffer to $40{,}000$ transitions, and update the networks every 256 environment steps instead of 64, taking 0.25 instead of 1 gradient step per environment step.

A.4 Abstract world modeling implementation details

Learning $p_o(\boldsymbol{f}'|\boldsymbol{s})$ with PoE-World. We first assume the following structure for $p_o(\boldsymbol{f}'|\boldsymbol{s})$:

$$p_o(\boldsymbol{f}'|\boldsymbol{s}) \triangleq p_o(f'_o|\boldsymbol{s})(\prod_{i \neq o} p_o(f'_i|f'_o, \boldsymbol{s})) \tag{7}$$

Then, each $p_o(f'_o|\boldsymbol{s})$ and $p_o(f'_i|f'_o, \boldsymbol{s})$ can be modeled with a product of experts and learned with PoE-World. And as mentioned in the main text, for $p_o(f'_i|f'_o, \boldsymbol{s})$, we further incorporate Gaussian priors with $\sigma = 0.1$ and $\mu = 0.5$ for experts that do not change f_i, and $\mu = 0.001$ for the rest.

The intuition behind the above structure is the model should first predict f'_o which corresponds to the result of executing option o, whether or not it will achieve its corresponding goal g, with $p_o(f'_o|s)$. Based on the result of the execution, the model can then predict the rest of the abstract features with $\prod_{i \neq i_o} p_o(f'_i|f'_o, s)$.

Expert generation for PoE-World. For $p_o(f'_o|s)$ of each option o, we prompt an LLM to synthesize a set of possible preconditions: Gemini 2.0 Flash-Lite with the prompts in Tables 8 and 9 for OCAtari, and Gemini 2.5 Flash with a block stacking version of these prompts for the block stacking domain. We then add one expert per precondition that sets $f'_o = 1$ if that precondition is true. We also add a "blanket" expert that sets $f'_o = 0$ without any precondition. Both types of experts have Gaussian weight prior with $\mu = 0.5$, but the blanket expert has $\sigma = 0.001$, while the experts with preconditions have $\sigma = 0.1$. We give low σ to the blanket expert because we want its weight stays close to 0.5. If we allow the weight for the blanket expert to change too much, it might become 0 when only positive examples are observed during fitting, causing the model to incorrectly conclude that option o will always succeed without any preconditions.

For $p_o(f'_i|f'_o, s)$ of each option o, we generate three experts: $f'_i = f_i$ (no change), $f'_i = 1$, and $f'_i = 0$. As mentioned in Section 3.1, we give a Gaussian weight prior with $\mu = 0.5, \sigma = 0.1$ for the first type of expert (no change), and $\mu = 0.5, \sigma = 0.001$ for the latter two types.

Weighting function. We implement weighting functions straightforwardly as a lookup table with the keys being the abstract states and the values being the corresponding low-level states. We only keep a single low-level state for each abstract state in the lookup table, so there is no sampling.

Undefined distribution and partial state. It is possible that in the weighting function, we try to sample from $w(\boldsymbol{s} \mid \boldsymbol{f})$ when we have not yet seen a $\boldsymbol{s}$ that corresponds to $\boldsymbol{f}$, meaning our lookup table for the key $\boldsymbol{f}$ is empty. When that happens, we leave $\boldsymbol{s}$ unspecified. Concretely, we assign $None$ values to all primitive features. We call states with $None$ values for some features, partial states. Arriving at partial states in the world-model-simulated environment does not terminate the episode right away. The episode only terminates if the experts in $p_o(\boldsymbol{f}'|\boldsymbol{s})$ try to access the features with $None$ values.

Additionally, we also set $f'_i = None$ for all $i \neq o$ when we sample from $p_o(\boldsymbol{f}'|\boldsymbol{s})$ but get $f'_o = 0$. We implement this mechanism because options can lead to highly unpredictable states when they are not successful in achieving their corresponding goals.

A.5 Experiments with a small language model

AgentOWL relies on an LLM for sub-goal proposal and world model learning, and our main Atari experiments use Gemini 2.5 Flash for the former and Gemini 2.0 Flash-Lite for the latter, which costs around $1 in API calls per run. To test whether AgentOWL needs a strong LLM, we rerun AgentOWL on Montezuma's Revenge with Qwen3.5-9B, a small open-weight model that fits on a single consumer GPU. As shown in Table 2, AgentOWL with Qwen3.5-9B masters options for all 18 goals within 1M environment steps in every seed, matching AgentOWL with Gemini models, while every baseline plateaus at 13 goals.

Method	Options mastered after 1M steps
DQN	13.00 ± 0.00
Goal-conditioned DQN	13.00 ± 0.00
Hierarchical DQN	13.00 ± 0.00
AgentOWL w/ Gemini models (Figure 5)	$\mathbf{18.00 \pm 0.00}$
AgentOWL w/ Qwen3.5-9B	$\mathbf{18.00 \pm 0.00}$

Table 2: Number of options mastered after 1M environment steps in Montezuma's Revenge, mean $\pm$ standard error over 3 seeds. Replacing Gemini models (Gemini 2.0 Flash-Lite + Gemini 2.5 Flash) with the much smaller Qwen3.5-9B does not change AgentOWL's performance.

A.6 Random and LLM-generated goal order

Our main experiments give AgentOWL a sequence of goals ordered by difficulty. To relax this assumption, we run AgentOWL on Montezuma's Revenge with two other goal orders: a random order, and an order proposed by an LLM. Since an agent might be asked to master a goal before it has the sub-options needed to make that goal tractable, we employ the same round-robin goal scheduling as Section A.3.

Table 3 shows that AgentOWL still outperforms all baselines with either goal order, although it masters fewer goals than with the manual order under the same training budget. In domains where the LLM has a strong prior, ordering the goals by difficulty with an LLM is also feasible: in our block stacking domain, the goals, their predicates, and their order are all generated by an LLM agent.

Method	Options mastered after 5M steps
DQN	13.00 ± 0.00
Goal-conditioned DQN	13.00 ± 0.00
Hierarchical DQN	13.00 ± 0.00
AgentOWL w/ manually ordered goals (Figure 5)	$\mathbf{18.00 \pm 0.00}$
AgentOWL w/ LLM-ordered goals	16.33 ± 1.67
AgentOWL w/ randomly ordered goals	16.67 ± 0.67

Table 3: Number of options mastered after 5M environment steps in Montezuma's Revenge with different goal orders, mean $\pm$ standard error over 3 seeds. The baselines use the manual goal order.

A.7 Prompts for LLM-based sub-goal proposer

Table 7 contains the prompt used for LLM-based sub-goal proposer.

A.8 Compute resources and execution time

Compute Resources. Our experiments are run on 1 GPU (NVIDIA RTX A6000 or NVIDIA GeForce RTX 3090) with 4 CPUs (CascadeLake, IceLake, or SaphireRapids) and 48 GB memory. The method involves calling LLMs through OpenRouter API; each experiment costs around $1 in API cost.

Execution time. Our method tends to take around 1 day for each experiment on Montezuma's Revenge and PrivateEye experiments and 2 days for each experiment on Pitfall.

Algorithm 2 DQN learning algorithm (without parallel environments)

Given an environment E, a list of actions/sub-options Ω, , a reward function R, and training hyperparameters in Table 4.
Initialy policy network: $\pi \leftarrow InitPolicy(E, \Omega)$
Initialize replay buffer $D \leftarrow \{\}$
Get initial state: $\boldsymbol{s} \leftarrow E.reset()$
for $t \leftarrow 0$ to $total_env_steps$ step $train_frequency$ **do**
    # Collect rollouts by executing π for $train_frequency$ steps
    for $i \leftarrow 0$ to $train_frequency$ **do**
        Sample action: $a \sim \pi(a \mid \boldsymbol{s})$
        Interact with environment: $\boldsymbol{s}' \leftarrow E.step(a)$
        Update replay buffer: $D \leftarrow D \cup \{(\boldsymbol{s}, a, \boldsymbol{s}', \kappa \cdot R(\boldsymbol{s}, a, \boldsymbol{s}')\}$
        Update current state $\boldsymbol{s} \leftarrow \boldsymbol{s}'$
    end for
    # Then, train for $n_gradient_steps$ steps
    Optimize the policy weights: $DQNUpdate(\pi, D, n_gradient_steps)$
end for
return π

Algorithm 3 Stable Hierarchical DQN learning algorithm (without parallel environments)
Note that ExecuteOneStep and ReceiveObsOneStep are defined in Algorithm 4 and Algorithm 5 respectively

Given $E, \Omega, D_\Omega, \pi_\Omega, \pi_g, g, n, \delta$ and training hyperparameters in Table 4 and Table 5.

*# We define a **mutable** struct to keep track of execution variables of an option*
 # In hierarchical options, each parent option selects a child option to execute, recursing until reaching a primitive action
struct $OptionExecutionState$
    Option: o
    Policy: π
    Goal: g
    Active child option: $child$
    Child start state: u
    Episode data: D
    Current execution time: t
    New data counter: ct
end struct

Initialize (mutable) option execution states:
    $\omega_\Omega \leftarrow \{OptionExecutionState(o, \pi_o, g_o, None, None, [], 0, 0)\}_{o \in \Omega}$
Initialize (mutable) target option execution state:
    $\omega_{target} \leftarrow OptionExecutionState(o_g, \pi_g, g, None, None, [], 0, 0)$
Get initial state: $s \leftarrow E.reset()$
for $i \leftarrow 0$ to $total_env_steps$ **do**
    Execute the target option for one step: $a \leftarrow ExecuteOneStep(s, \omega_{target}, \Omega, \omega_\Omega)$
    Interact with environment: $s' \leftarrow E.step(a)$
    Update option states and replay buffers: $(_, D_\Omega) \leftarrow ReceiveObsOneStep(s', \omega_{target}, \Omega, D_\Omega, n, \delta)$
    for $o \in \Omega$ **do**
        # optimize the policy weights if the option has collected enough new data
        if enough new data collected: $\omega_o.ct > train_frequency$ **then**
            Optimize the policy weights: $DQNUpdate(\omega_o.\pi, D_o, n_steps = n_steps_per_sample \cdot \omega_o.ct)$
            Reset new data counter: $\omega_o.ct \leftarrow 0$
        end if
    end for
end for
Retrieve π_Ω from the mutable execution states: $\pi_\Omega \leftarrow \{\omega_o.\pi\}_{o \in \Omega}$
Retrieve π_g from mutable target execution state: $\pi_g \leftarrow \omega_{target}.\pi$
return $\pi_g, \pi_\Omega, D_\Omega$

Algorithm 4 Helper function $ExecuteOneStep$ used in Stable Hierarchical DQN

function $ExecuteOneStep(s, \omega, \Omega, \omega_\Omega)$
    # keep recursing until we hit the leaf node of the execution trace
    if $\omega.child \neq None$ **then**
        return $ExecuteOneStep(s, \omega.child, \Omega)$
    end if

    Sample a child option to execute: $a \leftarrow \omega.\pi(a \mid s)$
    Set child start state: $\omega.u \leftarrow s$
    Increment the number of sub-options ω has executed: $\omega.t \leftarrow \omega.t + 1$

    # recurse if a is indeed a sub-option, not a primitive action
    if $a \in \Omega$ **then**
        Make sub-option a (along with its execution state) the active child: $\omega.child \leftarrow \omega_a$
        return $ExecuteOneStep(s, \omega.child, \Omega)$
    end if

    # return a if it is a primitive action
    return a
end function

Algorithm 5 Helper function $ReceiveObsOneStep$ used in Stable Hierarchical DQN

function $ReceiveObsOneStep(s', \omega, \Omega, D_\Omega, n, \delta)$
  # keep recursing until we hit the leaf node of the execution trace
  if $\omega.child \neq None$ **then**
    $(is_child_done, D_\Omega) \leftarrow ReceiveObsOneStep(s', \omega.child)$
  else
    $is_child_done \leftarrow True$
  end if

  # Add data to episode buffer if the child option is done
  if is_child_done **then**
    Get child start state: $s \leftarrow \omega.u$
    Get child option: $a \leftarrow \omega.child.o$
    Calculate reward: $r \leftarrow R_{\omega.g}(s, a, s')$
    Add datapoint to $\omega.D \leftarrow \omega.D \cup \{(s, a, s', r)\}$
    Child is no longer active: $\omega.child \leftarrow None$
  end if

  Check if the current option is done: $is_cur_done \leftarrow (\omega.g(s') = 1)$ or $(\omega.t > max_t)$

  # Update execution state and potentially update the option's replay buffer if the option is done executing
  if is_cur_done **then**

    # Check if all executed sub-options in the episode are stable
    $only_stable_in_episode \leftarrow True$
    for $(_, a, _, _) \in \omega.D$ **do**
      # a is not stable if a is a sub-option
      # AND has lower number of samples it has been trained with than n and lower goal completion rate than δ
      if $a \in \Omega$ and $n_a < n$ and $\delta_a < \delta$ **then**
        $only_stable_in_episode \leftarrow False$
      end if
    end for

    # Add episode buffer to the full replay buffer if all executed sub-options in the episode are stable
    if $only_stable_in_episode$ **then**
      Get option index: $o \leftarrow \omega.o$
      Add episode data to the corresponding replay buffer: $D_o \leftarrow D_o \cup \omega.D$
      Add number of new data to the counter: $\omega.ct \leftarrow \omega.ct + |\omega.D|$
      Retrieve current epsilon: $(\pi^{real}, \pi^{wm}, \epsilon) \leftarrow \omega.\pi$
      Anneal ϵ based on the number of new data: $\epsilon \leftarrow Anneal(\epsilon, |\omega.D|)$
      Update the policy with the new ϵ: $\omega.\pi \leftarrow (\pi^{real}, \pi^{wm}, \epsilon)$
    end if

    # Update execution state
    Child is no longer active: $\omega.child \leftarrow None$
    Clear episode buffer: $\omega.D \leftarrow \{\}$
    Execution time resets: $\omega.t \leftarrow 0$
  end if

  return is_cur_done, D_Ω
end function

Hyperparameter	Value
Number of parallel environments	4
Batch size	256
Learning rate	0.0001
Replay buffer size	$5000 \cdot 4$
Multi-step return	10
Discount factor γ	0.99
Priority replay temperature	0.01
Target network update interval	200
Number of gradient steps per training step	64
Training frequency	$16 \cdot 4$
Reward multiplier κ	10
Maximum gradient norm	10
Random exploration rate ϵ	0
Frame stacking	4
Optimizer	Kron [82, 83]

Table 4: Common training hyperparameters for Rainbow DQN.

Hyperparameter	Value
Max option executime time max_t	100
Gradient steps per sample $n_steps_per_sample$	1
Annealing schedule ϵ	$Linear(start = 1, end = 0, n_samples = 10000)$
$n_{threshold}$	20000
$\delta_{threshold}$	0.5

Table 5: Training hyperparameters that are specific to Hierarchical DQN.

Hyperparameter	Value
Multi-step return	1
Random exploration rate ϵ	$Linear(start = 1.0, end = 0.01, fraction = 0.95)$

Table 6: Training hyperparameters for Rainbow DQN that are specific to π^{wm}.

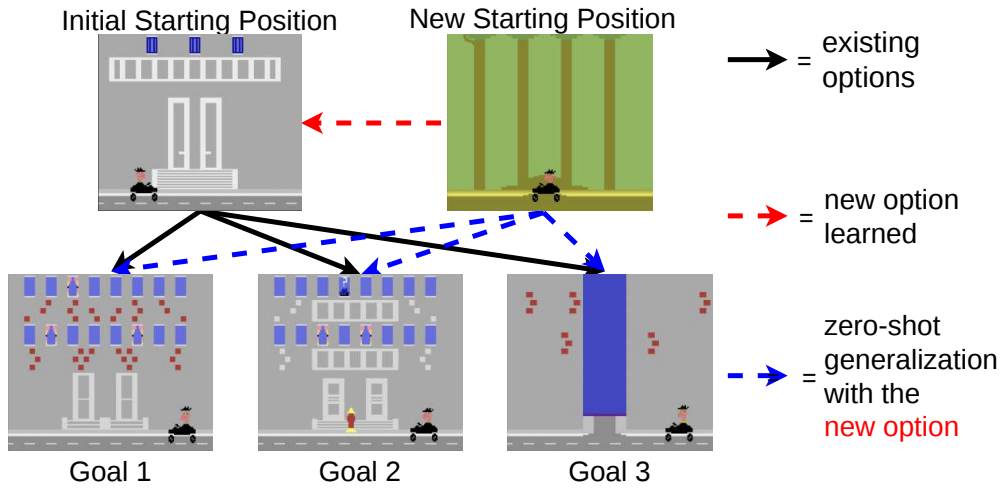

Figure 8: Illustration of the initial state, the new starting state, and the goals in Table 1a.

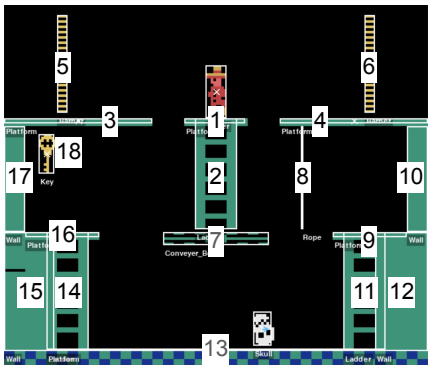

Figure 9: Screenshots of Montezuma's Revenge with goals labeled in order it appears in the ordered list of goals

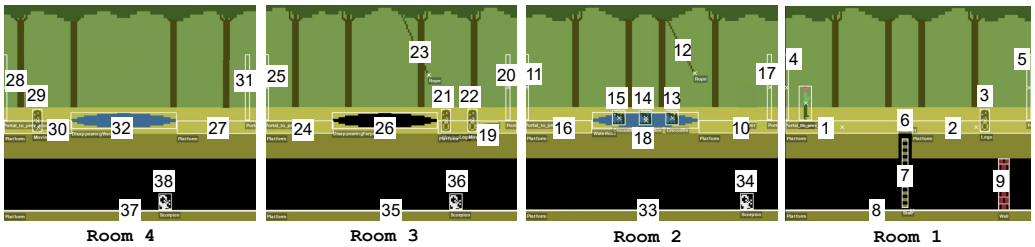

Figure 10: Screenshots of Pitfall with goals labeled in order it appears in the ordered list of goals

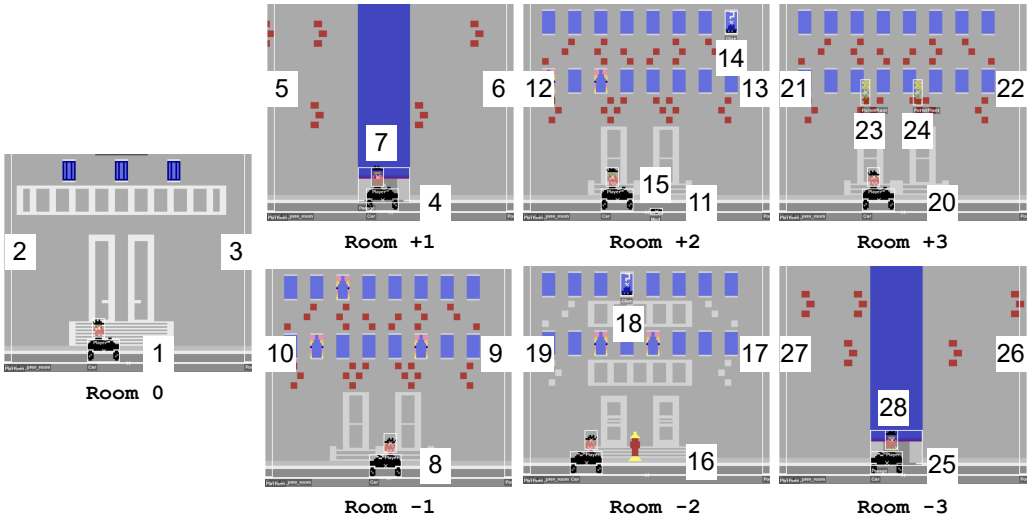

Figure 11: Screenshots of Private Eye with goals labeled in order it appears in the ordered list of goals

```
Here is the current observation of the game {game_name}:

{cur_obs}

And here is a list of goals we know how to achieve:

{achieved_goal_names_and_descriptions}

Your task is to list 1 achieved goals that, on its own, can act as a possible
stepping stone to achieve the target goal of '{target_goal_name}' -- Description:
'{target_goal_description}'.

Required reasoning process:
First, discuss out loud how to achieve the target goal of '{target_goal_name}',
taken into account the current observation.
Then, for each achieved goal, discuss out loud how completing that goal would help
us achieve the target goal of '{target_goal_name}', taken into account the current
observation.
Make sure to go through all achieved goals. But also do not keep repeating the same
achieved goal. The current position of the player is irrelevant.

Final output format:
After you are done reasoning, list the achieved goal in a numbered list with
the following format:
Possible stepping stone 1: <achieved goal>
```

Table 7: Prompt for LLM to propose sub-goals to hypothesize new sub-options. Note that $target_goal_description$ is simply "touch object with id obj_id or "in room #$room_num$, touch object with id obj_id" and $target_goal_name$ is something like "bot_plat" or "rope_in_roomnumber+2". cur_obs is a state sampled from D_{seen} if there is a single room like Montezuma's Revenge and multiple states, one per room, for Pitfall and Private Eye.

```
I'll give you an input list of objects.

I want you to list 4 possible features that the input list of objects has that
allows us to achieve a certain goal.

Here's an example:

Example input list of objects:
player object with at (x=16, y=104, w=8, h=21),
wall object with at (x=136, y=148, w=7, h=32),
logs object with at (x=125, y=118, w=6, h=14),
stairpit object with at (x=76, y=122, w=8, h=6),
stair object with at (x=78, y=136, w=4, h=42),
platform object with at (x=8, y=179, w=152, h=1),
platform object with at (x=8, y=125, w=152, h=1),
playerscore object with at (x=38, y=9, w=30, h=8),
lifecount object with at (x=23, y=22, w=1, h=8),
lifecount object with at (x=21, y=22, w=1, h=8),
timer object with at (x=31, y=22, w=37, h=8),
portal_0 object with at (x=7, y=85, w=1, h=40),
portal_1 object with at (x=155, y=85, w=1, h=40),
Interaction -- player object with at (x=16, y=104, w=8, h=21) is touching platform
object with at (x=8, y=125, w=152, h=1)

Example possible features that allow us to achieve the goal of '{goal}':
1. AnyObjTypeTouching: The player object touches a platform object
2. SpecificObjTouching: The player object touches the platform object located at
(x=8, y=125)
3. AnyObjTypeTouching: ...
4. SpecificObjTouching: ...

Now, I want you to list 4 possible features of the input list of objects has that
allows us to achieve the goal of '{goal}'.

Input list of objects:
{input}

Please follow these rules for your output:
1. Do not explain -- simply list each feature
2. Make the features diverse
3. Do use interactions (what the player is touching), as they usually make good
features
4. Each rule should of type 'AnyObjTypeTouching' or 'SpecificObjTouching'
```

Table 8: Prompt for LLM to propose preconditions for games where the agent controls only the Player object: Montezuma's Revenge and Pitfall.

```
I'll give you an input list of objects.

I want you to list 4 possible features that the input list of objects has that
allows us to achieve a certain goal.

Here's an example:

Example input list of objects:
player object (x=30, y=150, w=8, h=12),
car object (x=27, y=163, w=20, h=14),
score object (x=75, y=8, w=30, h=8),
clock object (x=67, y=19, w=30, h=8),
roomnumber_+0 object (x=0, y=0, w=0, h=0),
portal_to_prev_room object (x=8, y=27, w=5, h=150),
portal_to_next_room object (x=155, y=27, w=5, h=150),
platform object (x=8, y=177, w=152, h=1),

Example possible features that allow us to achieve the goal of '{goal}':
1. RoomNumberExist: An object with type 'roomnumber_+0' exists
2. ObjTouchingAndRoomNumberExist: The car object touches the platform object and an
object with type 'roomnumber_+0' exists

Now, I want you to list 2 possible features of the input list of objects has that
allows us to achieve the goal of '{goal}'.

Input list of objects:
{input}

Please follow these rules for your output:
1. Do not explain -- simply list each feature
2. Each rule should of type 'RoomNumberExist' or 'ObjTouchingAndRoomNumberExist'
3. Make sure to mention the roomnumber in the feature, e.g., 'an object with type
'roomnumber_+0' exists'
```

Table 9: Prompt for LLM to propose preconditions for games where the agent controls several objects: Private Eye (the agent controls Player and Car object)

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the limitations of the work performed by the authors.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have fully described the algorithms used in the paper, with details included in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We currently cannot release code and data, but we have fully described the algorithms used in the paper, with details included in the appendix.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Full experimental details are provided in appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Our main results are accompanied by error bars.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have provided compute resources used for our experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: Our work is foundational research and has no direct societal impact.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The creators of original data are all properly credited, and the license and terms of use of the data are explicitly mentioned and properly respected.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: This paper describes the usage of LLMs in our method.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.