A Neurips Checklist Answer Clarification

During submission, we were not aware of the guidance for filling the checklist and thus misunderstood some questions in it. In this section, we want to correct our answers to some of the checklist questions, if allowed:

- Broader Impacts: n/a
- Experiments: yes, and the code can be found at: `https://github.com/elden-neurips2023/ELDEN`

B ELDEN details

Assumptions We summarize our assumptions on the MDP as follows:

1. The state space can be factored as $\mathcal{S} = \mathcal{S}^1 \times \cdots \times \mathcal{S}^N$.
2. The transition of each state factor is independent, i.e., the dynamics can be represented $\mathcal{P}(s_{t+1}|s_t, a_t) = \prod_{n=1}^{N} p(s^n_{t+1}|\text{Pa}(\mathcal{S}^n_{t+1}))$.
3. There is no instantaneous dependency between state factors at the same time step t, i.e., no dependency such as $s^i_t \to s^j_t$ for any i, j.

For assumption 1, factored state space is commonly employed in causality literature and is applicable to many simulated or robotics environments. In cases where low-level observations or partial observability are present, disentangled representation or causal representation methods can be utilized to learn a factored state space [16]. When a factored state space is available, assumptions 2 and 3 generally hold.

Network Architecture In Figure 4(a), the architecture of ELDEN for predicting each state factor s^j_{t+1} is illustrated. The process consists of the following steps:

1. Feature Extraction: For each input state factor s^i_t, ELDEN utilizes a separate multi-layer perception (MLP) to extract its corresponding feature g^i.
2. Entity Interaction: ELDEN employs a multi-head self-attention module to model entity interactions and generates a set of transformed features h^i that incorporate information from other state factors.
3. Prediction using Multi-Head Attention: With h^j as the query, ELDEN utilizes a multi-head attention module to compute the prediction $\hat{p}(s^j_{t+1}|s_t, a_t)$ for each state factor. For continuous state factor, $\hat{p}(s^j_{t+1})$ is modeled as a normal distribution with the mean computed by the network and a fixed variance equal to 1. For discrete factor, $\hat{p}(s^j_{t+1})$ is a categorical distribution with network outputs as class probabilities.

Throughout the prediction process, there are a total of N such networks in ELDEN, with each network responsible for predicting a separate state factor s^j_{t+1}.

The training loss for the dynamics model is:

$$L = -\log \prod_{j=1}^{N} \hat{p}(s^j_{t+1}|s_t, a_t) + \lambda \sum_{i,j} \left| \frac{\partial \hat{p}(s^j_{t+1})}{\partial s^i_t} \right|, \tag{3}$$

where λ is the coefficient for partial derivative regularization.

C Environment Details

In this section, we provide a detailed description of the environment, including its semantic stages representing internal progress toward task completion, state space, and action space. We also highlight that while each task consists of multiple semantic stages, agents do not have access to this information. The learning signal for agents is solely based on a sparse reward of 0 or 1, indicating whether the task has been completed or not. Additionally, in each environment, the poses of all environment entities are randomly initialized for each episode.

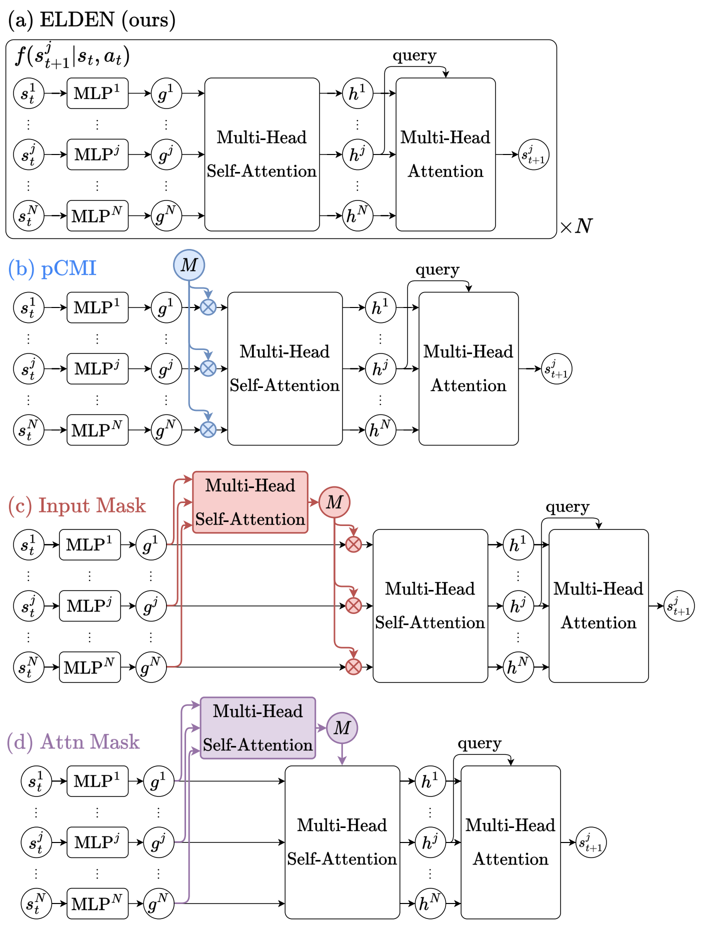
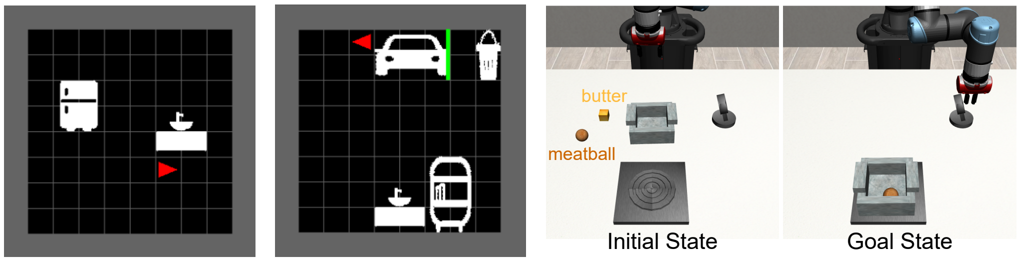


Figure 4: The dynamics model of each local dependency detection method. **(a)** The dynamics model of ELDEN for predicting s^j_{t+1}. Notice that each network predicts s^j_{t+1} only, and there are N such networks in total, each responsible for predicting one state factor in s_{t+1}. For visual simplicity, the "$\times N$" symbol is only shown in (a). **(b)** pCMI computes $p(s^j_{t+1}|s_t, a_t)$ and $p(s^j_{t+1}|s_t \setminus s^i_t, a_t)$ by manually setting the binary mask M to different values, where $\otimes$ represents element-wise multiplication. **(c)** For Input Mask, M is learned to condition on (s_t, a_t) and is regularized to use as few inputs as possible. **(d)** For Attn Mask, M also conditions on (s_t, a_t) but is applied to the attention score in the self-attention module.

Meanwhile, as ELDEN focuses on exploring novel local dependencies between environment entities, in all environments, the action space consists of hard-coded skills to increase the probability of entity interactions and bypass navigation challenges under sparse rewards. Extending ELDEN to explore local dependency and learn such skills simultaneously would be an important direction for future work.

Thawing As shown in Fig. 5(a), the Thawing environment consists of a sink, a refrigerator, and a frozen fish. The task requires the agent to complete the following **stages**: (1) open the refrigerator, (2) take the frozen fish out of the refrigerator, and (3) put the fish into the sink to thaw it. The discrete state space consists of (i) the agent's position and direction, (ii) the positions of all environment entities, (iii) the thawing status of the fish, and (iv) whether the refrigerator door is opened. The discrete action space consists of (i) moving to a specified environment entity, (ii) picking up / dropping down the fish, and (iii) opening / closing the refrigerator door.

CarWash As shown in Fig. 5(b), the CarWash environment consists of a car, a sink, a bucket, a shelf, a rag, and a piece of soap. The task requires the agent to complete the following **stages**: (1) take the rag off the shelf, (2) put it in the sink, (3) toggle the sink to soak the rag up, (4) clean the



(a) Thawing (b) CarWash (c) Kitchen

Figure 5: Environments.

car with the soaked rag, (5) take the soap off the self, and (6) clean the rag with the soap inside the bucket. The discrete state space consists of (i) the agent's position and direction, (ii) the positions of all environment entities, (iii) the soak status of the rag, (iv) the cleanness of the rag and the car, and (iv) whether the sink is toggled. The discrete action space consists of (i) moving to a specified environment entity, (ii) picking up / dropping down the rag, (iii) toggling the sink, and (iii) picking up / dropping down the soap.

Kitchen As shown in Fig. 5(c), in the kitchen environment, there are a robot arm (i.e., the agent), a piece of butter, a meatball, a pot, and a stove with its switch. The task requires the agent to complete the following **stages**: (1) pick and place the butter into the pot, (2) pick and place the pot onto the stove, (3) turn on the stove to melt the butter in the pot, (4) pick and place the meatball into the pot to cook it, and (5) turn off the stove. Notice that melting the butter is a prerequisite for cooking the meatball, otherwise, it will result in the meatball being overcooked and the task failing. The state space is continuous, consisting of the pose of all objects, the melting status of the butter, and the cooking status of the meatball (whether it is raw, cooked, or overcooked). The action space is discrete, consisting of hard-coded skills: moving to [butter, meatball, pot, pot handle, stove, stove switch], grasping, dropping, and toggling the switch. Grasping and toggling are only applicable when the end-effector is close to the corresponding environment entities.

D Evaluating the Detection of Local Dependencies

D.1 Implementation

Baselines We give a detailed description of each baseline as follows:

- **pCMI** (point-wise conditional mutual information): it considers that the local dependency $s_t^i \to s_{t+1}^j$ exists if their point-wise conditional mutual information is greater than a predefined threshold, i.e., $\text{pCMI}^{i,j} := \log \frac{p(s_{t+1}^j|s_t,a_t)}{p(s_{t+1}^j|s_t \setminus s_t^i,a_t)} \geq \epsilon$. As shown in Fig. 4(b), to compute $\text{pCMI}^{i,j}$, Wang et al. [32] uses a manually defined binary mask $M \in [0,1]^N$ to ignore some inputs when predicting s_{t+1}^j: (1) to compute $p(s_{t+1}^j|s_t,a_t)$, M uses all inputs (all its entries are set to 1), and (2) to compute $p(s_{t+1}^j|s_t \setminus s_t^i, a_t)$, the entry for g^i is set to 0. When evaluating the local dependency, pCMI needs to compute $p(s_{t+1}^j|s_t \setminus s_t^i, a_t)$ for every i, and thus its computation cost is N times larger than ELDEN. We also computes pCMI following Seitzer et al. [27], which yields similar performance but is even more computationally expensive compared to the method proposed by Wang et al. [32].
- **Attn** (attention): it uses the same architecture as ELDEN that is shown in Fig. 4(a). When computing the overall attention score, it averages the attention score across all heads in each module, then computes the likelihood of dependency $s_t^i \to s_{t+1}^j$ as $\sum_{k=1}^N c^{g^i,h^k} \cdot c^{h^k,s_{t+1}^j}$ where $c^{a,b}$ is the averaged score between the input a and the output b.
- **Input Mask**: as shown in Fig. 4(c), it also uses a binary mask M except that M is computed from (s_t, a_t). During training, to only use necessary inputs for s_{t+1}^j prediction, M is regularized with the L1 norm on its number of non-zero entries. The Gumbel reparameterization is used to compute the gradient for the binary M [11].

Table 2: Parameters of the dynamics model training for local dependency detection experiments. Parameters shared if not specified.

		Name		**Tasks**	
			Thawing	CarWash	Kitchen
environment		episode length	20	100	100
		grid size	10	10	N/A
training		optimizer		Adam	
		learning rate		3×10^{-4}	
		batch size		32	
		# of training batches		500k	
		# of random seeds		3	
		mixup Beta parameter	1	1	N/A
ELDEN		activation functions		ReLU	
		$\{\text{MLP}\}_{i=1}^{N}$	[64, 64]	[64, 64]	[128, 128]
		λ annealing starts	50k	50k	100k
		λ annealing ends	100k	100k	200k
	attention	# of heads		4	
		use bias		False	
		key, query, value size	16	16	32
		output size	64	64	128
		post attn MLP	[64, 64]	[64, 64]	[128, 128]
Input Mask		attention parameters		same as ELDEN	
		M regularization coefficient		1×10^{-2}	
		M regularization annealing starts	50k	50k	100k
		M regularization annealing ends	100k	100k	200k
Attn Mask		attention parameters		same as ELDEN	
	SKPMD	signature size		64	
		learnable bandwidth		True	
		bandwidth initialization		1	

- **Attn Mask**: as shown in Fig. 4(d), similar to Input Mask, a mask M of size $N \times N$ is computed from (s_t, a_t), but it is applied to the attention score. The mask is regularized with Stochastic Kernel Modulated Dot-Product (SKMDP) proposed by Weiss et al. [33].

For modules that are shared by all methods, we use the same architecture for a fair comparison.

Data For a fair comparison, when training each method, we use the same dataset collected by a scripted policy, rather than let each method collect its own data, to avoid potential performance differences caused by data discrepancies. Specifically, we use a scripted policy to expose all potential local dependencies and collect 500K transitions in each environment.

Notice that, in exploration with sparse reward experiments, the dynamics models are still trained online, using the transition data collected on its own.

Hyperparameters The hyperparameters used for evaluating local dependency detection of each method are provided in Table 2. Unless specified otherwise, the parameters are shared across all environments.

D.2 Ablation of Mixup and Partial Derivative Regularization

In our ablation study on ELDEN for local dependency detection, we investigate the impact of each component with the following variations:

- No Mixup & No Reg: We disable the use of Mixup for discrete space prediction, and no partial derivative regularization is applied in this case.
- Different partial derivative regularization coefficients: we test with different λ values in $\{0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$.

Table 3: Ablation of ELDEN on local dependency detection

	THAWING		CARWASH		KITCHEN	
	ROC AUC	F1	ROC AUC	F1	ROC AUC	F1
no Mixup & no Reg	0.48 ± 0.01	0.42 ± 0.01	0.44 ± 0.00	0.27 ± 0.01	N/A	N/A
no Reg, i.e., $\lambda = 0$	0.57 ± 0.01	0.52 ± 0.01	0.54 ± 0.01	0.42 ± 0.01	0.64 ± 0.01	0.24 ± 0.01
$\lambda = 10^{-1}$	0.68 ± 0.00	**0.57** ± 0.00	0.73 ± 0.01	0.58 ± 0.02	0.55 ± 0.00	0.14 ± 0.00
$\lambda = 10^{-2}$	**0.71** ± 0.01	**0.57** ± 0.00	0.76 ± 0.01	0.60 ± 0.00	0.60 ± 0.01	0.21 ± 0.01
$\lambda = 10^{-3}$	0.64 ± 0.01	0.55 ± 0.01	**0.78** ± 0.02	**0.66** ± 0.02	0.65 ± 0.00	0.24 ± 0.01
$\lambda = 10^{-4}$	0.65 ± 0.02	0.55 ± 0.01	0.75 ± 0.01	0.60 ± 0.01	**0.66** ± 0.00	**0.25** ± 0.01
$\lambda = 10^{-5}$	0.63 ± 0.00	0.53 ± 0.01	0.72 ± 0.00	0.57 ± 0.00	0.65 ± 0.01	0.24 ± 0.01

Table 4: Parameters of the Policy Learning. Parameters shared if not specified.

	Name	**Tasks**		
		Thawing	CarWash	Kitchen
PPO	optimizer		Adam	
	activation functions		Tanh	
	learning rate		1×10^{-4}	
	batch size		32	
	clip ratio		0.1	
	MLP size		[128, 128]	
	GAE λ		0.98	
	target steps		250	
	n steps	60	600	100
	# of environments	20	20	80
training	# of random seeds		3	
	intrinsic reward coefficient		1	
	# of dynamics update per policy step		1	
	dynamics learning rate		1×10^{-5}	
	ensemble size		5	
	PER level of prioritization	N/A	N/A	0.5
	mixup Beta parameter	0.1	0.1	N/A
	partial derivative threshold ϵ		3×10^{-4}	

As shown in Table. 3, in Thawing and CarWash environments, partial derivative regularization with appropriate coefficients significantly improves ELDEN's detection on local dependencies, compared to no regularization (i.e., $\lambda = 0$) or inappropriate λ values. Furthermore, in these discrete-state environments, the use of Mixup is crucial — even when compared to using Mixup without any regularization, not using Mixup leads to a noticeable degradation in the prediction performance.

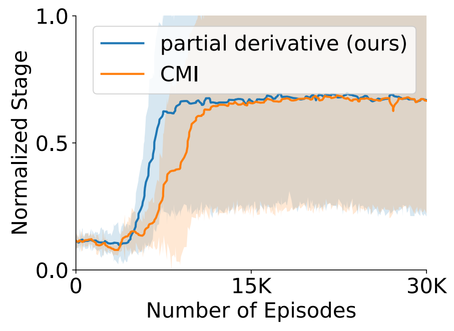
E Evaluating Exploration in Sparse-Reward RL Tasks

E.1 Implementation

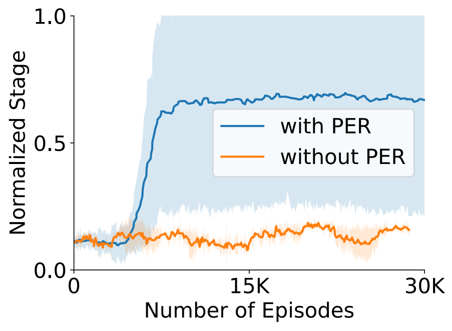
During policy learning, all methods share the same PPO and training hyperparameters, provided in Table 4. The hyperparameters for dynamics model setup during policy learning are the same as in Table 2 unless specified otherwise.

E.2 Ablation of Local Dependency Metrics

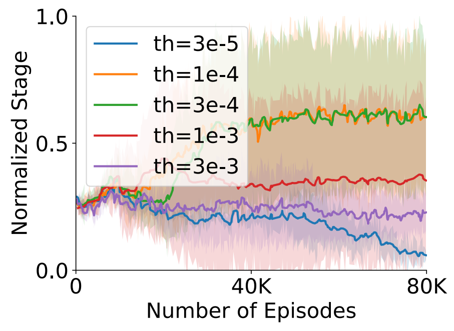
In this section, we compare the exploration performance when using different local dependency detection methods. Specifically, we compare with pCMI as it achieves best local dependency detection in Sec. 4.1. We present the comparison results between ELDEN and pCMI in the Kitchen environment in Fig. 6(a) where both methods successfully learn to solve the task. However, it is important to



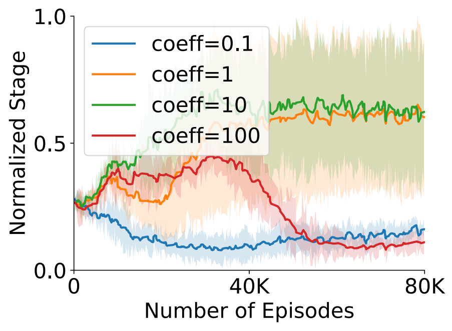
(a) Different local dependency detection methods



(b) Using PER or not



(c) Different thresholds (th) for local dependencies



(d) Different intrinsic reward coefficients

Figure 6: Ablation of ELDEN on task learning.

notice that the computation cost of pCMI is N times more than ELDEN, and thus may not scale to environments with a large number of entities.

E.3 Ablation of Prioritized Experience Replay

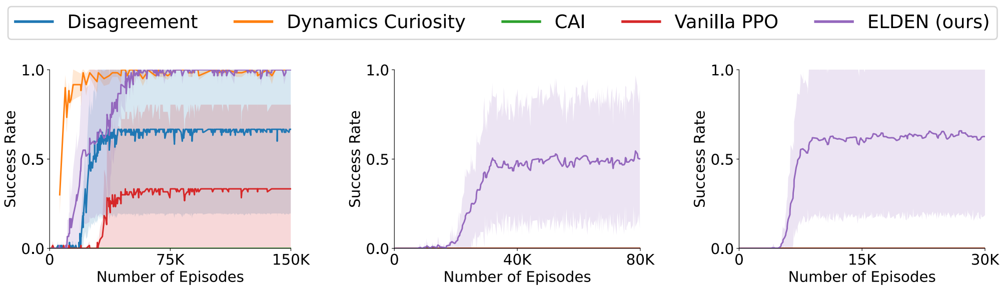
We study the effectiveness of Prioritized Experience Replay (PER) on task learning. Specifically, we test ELDEN with and without PER in the Kitchen environment, and show the result in Fig. 6(b). We can see that ELDEN without PER fails to learn useful policy. The reason is some key entity interactions occur rather rarely before the agent masters them, e.g., there is a small chance for the agent to cook the meatball with random actions. Hence, PER helps the dynamics model learn such infrequent dependencies quickly, enabling it to bias the exploration toward reproducing such dependencies.

E.4 Ablation of Partial Derivative Threshold

The partial derivative threshold ϵ determines the dependencies predictions. A threshold that is too large / too small will make all dependency predictions negative / positive respectively, leading to deteriorated performance. In this section, we examine whether our method is sensitive to the choice of threshold in the CarWash environment, where the results are presented in Fig. 6(c). We observe that our method is relatively sensitive to the choice of threshold, and an inappropriate threshold could cause catastrophic failure. A potential next step for ELDEN is to automatically determine the partial derivative threshold.

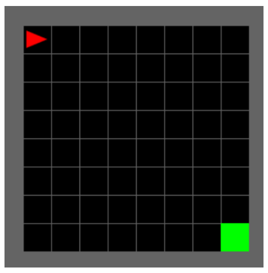
E.5 Ablation of Intrinsic Reward Coefficient

The intrinsic reward coefficient controls the scale of the intrinsic reward relative to the task reward. We examine the effect of this coefficient by experimenting with different values in the CarWash environment, where the results are presented in Fig. 6(d). We find that our methods work well in a

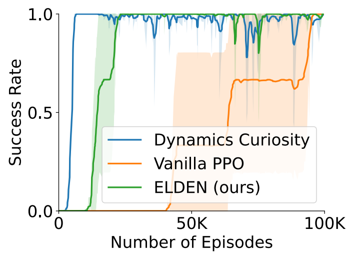


(a) Thawing (b) CarWash (c) Kitchen

Figure 7: Learning curve of ELDEN (ours) compared to baseline approaches (mean ± std dev of success rate across three random seeds). For both CarWash and Kitchen, the success rates for all baselines are zero throughout the training, overlapping with the x axis.



(a) a navigation task with the goal in green



(b) ELDEN performs worse than Dynamics Curiosity

Figure 8: We demonstrate a failure mode of our method on a navigation task.

large range of the intrinsic reward coefficients (1 - 10), since the task only gives sparse rewards and the intrinsic rewards are the only learning signal most of the time. The only exceptions are (1) when the intrinsic reward coefficient is too large (e.g., 100), the intrinsic reward significantly surpasses the task reward, and (2) when the coefficient is too small (e.g., 0.1), the episode intrinsic reward will also be too small (e.g., 0.03) for PPO to learn any useful policy.

E.6 Success Rate Plots

As a supplementary to the normalized stage metric used in the main paper, we provide the success rate as an additional metric. The success rate learning curves of all methods in the three environments are shown in in Fig. 7. Again, ELDEN outperforms and performs comparably with all baselines. Notice that, in the CarWash and Kitchen environments, all baselines never succeed throughout the training (i.e., success rate = 0 for all episodes), leading to training curves that overlap with the x axis.

F Failure Modes of ELDEN

As mentioned in the main paper, ELDEN may have limited advantages for tasks that require precise control of a specific environment entity. One such example is navigation, where the agent needs to reach a very specific point in space that has no particular semantic meaning. We empirically examine this statement in the Minigrid environment [7], where the agent needs to navigate to the green goal point in an empty room through primitive actions (turn left, turn right, and move forward), as shown in Fig. 8(a). We compare ELDEN against Dynamics Curiosity and Vanilla PPO, and present the result in Fig. 8(b). Since this environment is relatively simple, all three methods are eventually able to solve the task. However, the Dynamics Curiosity converges faster than ELDEN, showing that ELDEN is indeed not as capable as curiosity-driven explorations in tasks that focus on precise control rather than exploring dependencies between environment entities. The Vanilla PPO converges slowest, indicating that even in the Empty environment, ELDEN still has advantages over purely random exploration.

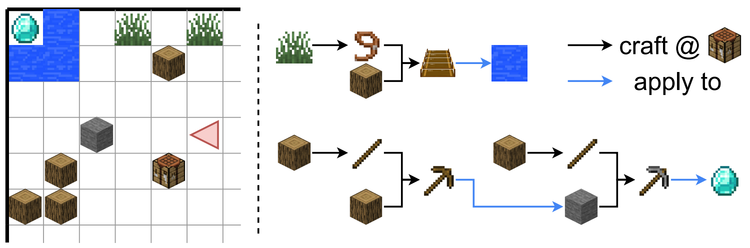


Figure 9: **(Left)** A visualization of the 2-d Minecraft environment. **(Right)** To get the gem, the agent needs to (top) craft a bridge to get across the rive, and (bottom) craft a stone pickaxe that is required to collect the gem.

G Minecraft 2D

We also evaluate all exploration methods in a discrete 2-d Minecraft environment that exhibits more objects and more complex entity dependencies than Thawing and CarWash. The environment is modified from the one used by Andreas et al. [1]. Due to limited space in the paper, we chose to show the results in the appendix.

G.1 Environment Details

As shown in Fig. 9, the environment has complex chained dependencies — to get the gem, the agent needs to

1. get across the river to reach the gem by
   (a) collecting a unit of grass and crafting a rope,
   (b) collecting a unit of wood and crafting a bridge with the rope,
   (c) building the bridge on top of the river;
2. collect the gem by
   (a) collecting a unit of wood to craft a wood stick
   (b) collecting another unit of wood and combining it with the stick to craft a wood pickaxe that is required for collecting the stone,
   (c) collecting a unit of wood and a unit of stone to craft a stick and then a stone pickaxe that is required for collecting the gem,
   (d) collecting the gem with the stone pickaxe.

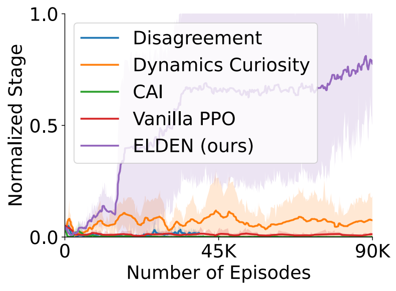


Figure 10: Learning curve of ELDEN (ours) compared to baseline approaches (mean ± std dev of the number of stages completed across three random seeds) in the 2-d Minecraft environment.

Notice that all crafting must be conducted at the crafting table. The discrete state space consists of (i) the agent's position and direction, (ii) an inventory tracking the number of materials and tools that the agent has, and (iii) the positions of all environment entities. The discrete action consists of (i) picking up / applying tools to (only effective when the agent faces an environment entity and has the necessary tools to interact with it), (ii) crafting a specified tool (only effective when the agent has enough materials and faces the crafting table), and (iii) moving to a specified environment entity.

G.2 Evaluating Exploration in 2-d Minecraft with Sparse Rewards

As shown in Fig. 10, even though the task requires that the agent follows the complex craft procedure with complex chained dependencies, ELDEN still learns to finish the task. In contrast, other exploration method fails to finish the task, only ending up with crafting one or two useful tools. This result again demonstrates ELDEN's advantage in exploring complex interactions between a large number of environment entities.

H Compute Architecture

The experiments were conducted on machines of the following configurations:

- Nvidia 2080 Ti GPU; AMD Ryzen Threadripper 3970X 32-Core Processor
- Nvidia A40 GPU; Intel(R) Xeon(R) Gold 6342 CPU @2.80GHz
- Nvidia A100 GPU; Intel(R) Xeon(R) Gold 6342 CPU @2.80GHz