Chain-of-Imagination for Reliable Instruction Following in Decision Making

Enshen Zhou[1,2,*], **Yiran Qin**[1,3,*], **Zhenfei Yin**[1,4], **Yuzhou Huang**[3],

Ruimao Zhang[3,†], **Lu Sheng**[2,†], **Yu Qiao**[1], **Jing Shao**[1,‡]

[1]Shanghai Artificial Intelligence Laboratory [2]Beihang University
[3]The Chinese University of Hong Kong, Shenzhen [4]The University of Sydney
zhouenshen@buaa.edu.cn yiranqin@link.cuhk.edu.cn
ruimao.zhang@ieee.org lsheng@buaa.edu.cn shaojing@pjlab.org.cn

Figure 1: **Comparison between *DecisionDreamer* and previous studies in Minecraft.** For the "`Chop a tree`" task, the agent should break the logs from bottom to top in sequence to collect more logs. *DecisionDreamer* uses the Chain-of-Imagination mechanism to imagine a series of situation-aware visual sub-goals that the logs should be collected one-by-one in a bottom-up manner and predict reliable but short-horizontal actions to achieve them sequentially. Previous methods, driven by the single instruction, struggle with such instructions that require sequentially achieving several intermediate sub-goals and therefore fail to collect more logs. As shown in the figure above, they attempt to break the upper log before successfully breaking the lower log ($\mathcal{O}_t$ to $\mathcal{O}_{t+k}$) and return to try breaking the lower log ($\mathcal{O}_{t+2k}$ to $\mathcal{O}_{t+3k}$).

Abstract

Enabling the embodied agent to imagine step-by-step the future states and sequentially approach these situation-aware states can enhance its capability to make reliable action decisions from textual instructions. In this work, we introduce a simple but effective mechanism called Chain-of-Imagination (CoI), which repeatedly employs a Multimodal Large Language Model (MLLM) equipped diffusion model to facilitate imagining and acting upon the series of intermediate situation-aware visual sub-goals one by one, resulting in more reliable instruction-following capability. Based on the CoI mechanism, we propose an embodied agent *DecisionDreamer* as the low-level controller that can be adapted to different open-world scenarios. Extensive experiments demonstrate that *DecisionDreamer* can achieve more reliable and accurate decision-making and significantly outperform the state-of-the-art generalist agents in the Minecraft and CALVIN sandbox simulators, regarding the instruction-following capability. For more demos, please see https://sites.google.com/view/decisiondreamer.

[*] Equal contribution [†] Corresponding author [‡] Project leader

Preprint. Under review.

1 Introduction

One of the core objectives of current embodied intelligence is to develop a generalist agent that can solve endless open-world tasks [4; 38; 3]. Existing methods [34; 43; 18; 45] have leveraged Large Language Models (LLMs) to decompose challenging long-horizon tasks and create high-level plans that guide agents in making decisions. However, these LLM-based agents' decision-making performance is bottlenecked by their low-level controllers, which may not reliably follow textual instructions as fine-grained sequential control signals (*e.g.*, keyboard and mouse in Minecraft) that depend on the situation (*e.g.*, the current ego-centric observation, the current status of the agent, the next sub-goal, and *etc.*) surrounding the agents.

Recent studies [25; 6; 44] explore more powerful controllers by translating textual instructions into visual goals and employing pretrained visual goal-based decision-making foundation models to convert the visual goals into executable action sequences. These decision-making foundation models are pretrained on vast, task-agnostic datasets of robotic manipulation [23; 32] or human gameplay videos [1; 13], and thus preserve certain generalization capabilities towards diverse action decisions. These methods, such as STEVE-1 [25], usually align a textual instruction with a single visual goal embedding. However, a single visual goal embedding may not well represent a series of situation-aware visual sub-goals that have to be achieved one-by-one. For example, in Fig. 1, STEVE-1 would transfer the instruction "`chop a tree`🌳" into a visual goal that can decide a sequence of actions "`strike`", but the controller may falsely strike the upper trunk of the tree before the bottom trunk has been broken into logs. In this case, if the controller foresees sub-goals that the logs should be collected one-by-one in a bottom-up manner, the instruction would be easier to follow.

To this end, we introduce a simple but effective mechanism called Chain-of-Imagination (CoI), which enables the agent to imagine and act upon the series of intermediate visual sub-goals one-by-one along with the ever-changing situations, resulting in a more reliable instruction-following capability. To be specific, the CoI mechanism is implemented by repeatedly executing an Imaginator and a Goal Generator. In each round of imagination, the Imaginator generates short-horizontal future imaginations based on the current observation and the textual instruction, and the Goal Generator converts these imaginations into visual sub-goal embeddings to better guide the goal-based foundation models for predicting reliable but short-horizontal actions. The chain of such imaginations can thus construct a series of short-horizontal action sequences that will eventually solve the instructed task, as indicated by Fig. 1.

To validate the CoI mechanism, we propose an embodied agent *DecisionDreamer* that is equipped with CoI, as the low-level controller in different open-world scenarios. Specifically, *DecisionDreamer* comprises three modules: (1) an Imaginator based on a Multimodal Large Language Model (MLLM) enhanced diffusion model, (2) a PolicyNet based on scenario-specific pre-trained goal-based decision-making foundation models, and (3) a Goal Generator as the bridge between the Imaginator and the PolicyNet. As indicated above, the multi-turn interaction between the Imaginator and the Goal Generator forms the CoI mechanism. Note that, training an Imaginator to envision visual sub-goals requires extensive data. We employ an efficient *Goal Drift Collection* method to gather a large amount of egocentric data that help the Imaginator to implicitly understand how the instructions are achieved step-by-step but without tedious annotations of true visual sub-goals.

Extensive experiments show that *DecisionDreamer* with the Chain-of-Imagination (CoI) mechanism reliably follows instructions, outperforming the best baseline by nearly 1.5 times in Minecraft according to evaluation metrics. Moreover, this mechanism can be readily adapted to other domains, such as CALVIN, with minimal redesign, where *DecisionDreamer* also achieves SOTA performance.

In summary, the contribution of this work is three-fold:

- We introduce the Chain-of-Imagination mechanism, which allows the agent to imagine and act upon the series of intermediate visual sub-goals one-by-one, enhancing its ability to follow instructions more reliably.
- We employ an efficient *Goal Drift Collection* method to gather data and train the MLLM-enhanced diffusion model to generate egocentric future images continuously.
- Leveraging these methods, we create an agent named *DecisionDreamer* that significantly outperforms the best generalist agent baseline in Minecraft and CALVIN.

2 Related Work

Diffusion models for Image Generation. Recent advances in text-to-image diffusion models [37; 39] have greatly improved instruction-driven image-to-image methods [46; 7] like InstructPix2Pix [5], primarily used for image editing that aims to alter content while keeping the background constant. In the embodied domain, SuSIE [2] effectively uses InstructPix2Pix as a high-level planner to decompose the instructions into image sub-goals of robotic arms, provided the backgrounds are simple and static. The method struggles with egocentric images in dynamic, open-world environments like Minecraft, where the backgrounds change significantly and the images should conform more to physical principles such as perspective. Leveraging the advanced instruction comprehension and visual perception capabilities of multimodal large language models (MLLMs), many studies enhance previous image editing models, like MGIE [14] and SmartEdit [19]. In this work, we train the MLLM-enhanced diffusion model to generate continuous future images for guiding real-time, low-level control in open-world environments, especially from an egocentric perspective, moving beyond the static backdrops common in image editing.

Diffusion models for Image Generation. Recent advances in text-to-image diffusion models [37; 39] have greatly improved instruction-driven image-to-image methods [46; 7] like InstructPix2Pix [5], primarily used for image editing that aims to alter content while keeping the background constant. In the embodied domain, SuSIE [2] effectively uses InstructPix2Pix as a high-level planner to decompose the instructions into image sub-goals of robotic arms, provided the backgrounds are simple and static. The method struggles with egocentric images in dynamic, open-world environments like Minecraft, where the backgrounds change significantly and the images should conform more to physical principles such as perspective. Leveraging the advanced instruction comprehension and visual perception capabilities of multimodal large language models (MLLMs), many studies enhance previous image editing models, like MGIE [14] and SmartEdit [19]. In this work, we train the MLLM-enhanced diffusion model to generate continuous future images for guiding real-time, low-level control in open-world environments, especially from an egocentric perspective, moving beyond the static backdrops common in image editing.

Build Instruction-Following Agents using Pre-trained Goal-based Model. Research on generalist agents in complex and dynamic embodied environments is increasingly popular in AI. There has been a recent explosion of interest in general instruction-to-action models using visual goal-based models pre-trained on large-scale task-agnostic datasets [25; 12; 2] to follow instructions. Since these datasets are not confined to specific tasks, the models preserve certain generalization capabilities toward diverse action decisions. For instance, VPT [1] collects extensive data from human players in the open-world Minecraft, focusing on exploration rather than a specific task, and trains a foundation model. STEVE-1 [25] uses this foundation model to fine-tune and allow visual goals to guide it while preserving the pre-trained skills. However, these works often align a textual instruction with a single visual goal embedding like STEVE-1, struggling to control pre-trained models along with the ever-changing situation. or generate videos like UniPi [12], which are computationally expensive and have physical inconsistencies that confuse pre-trained models. In this paper, We use the Chain-of-Imagination mechanism to enable the agent to imagine and act upon the series of intermediate visual sub-goals one by one, resulting in a more reliable instruction-following capability.

3 Method

We first provide an overview of our *DecisionDreamer* and Chain-of-Imagination (CoI) mechanism. Next, we introduce the advantages of the CoI. Then, we elaborate on the dataset construction method, *i.e. Goal Drift Collection*, for training the Imaginator. Finally, we introduce each part's network specification of *DecisionDreamer*.

3.1 Overview of *DecisionDreamer*

Our *DecisionDreamer* comprises three modules, *i.e.*, Imaginator $\mathcal{F}_\theta$, Goal Generator $\mathcal{G}_\xi$, and PolicyNet π_ϕ. Fig. 2 demonstrates how *DecisionDreamer* works with the help of the Chain-of-Imagination mechanism. First, the Imaginator takes in instructions y and current observations $\mathcal{O}_t$ and imagines a future sub-goal imagination $\widehat{\mathcal{I}}_t \sim \mathcal{F}_\theta(\mathcal{I}_t \mid \mathcal{O}_t, y)$ depicting a moment or likely visual sub-goal within the process of completing the given instruction y. Next, the Goal Generator creates a visual sub-goal

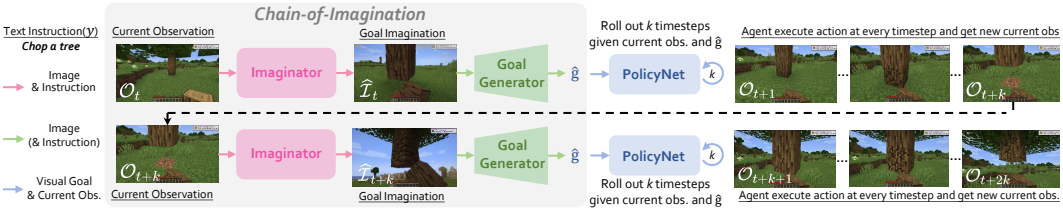


Figure 2: **The Overview of *DecisionDreamer*.** The Imaginator imagines a goal imagination based on the instruction and current observation. The Goal Generator transforms this into a precise visual goal and the PolicyNet progressively rolls out actions for k timesteps conditioned on it. After k timesteps the visual goal is refreshed and the process recycles. This figure shows executing k timesteps based on the previous visual goal, then regenerating a new one to execute for another k timesteps.

embedding $\widehat{g} \sim \mathcal{G}_\xi(g \mid \widehat{\mathcal{I}}_t, \mathcal{O}_t, y)$ in awareness of the current observation $\mathcal{O}_t$, instruction y and future sub-goal imagination $\widehat{\mathcal{I}}_t$, aligning with the visual goal space of the PolicyNet. Notably, if PolicyNet's goal input only comprises an image (*e.g.*, in the case of CALVIN), the $\widehat{g}$ is just the imagination $\widehat{\mathcal{I}}_t$ without the use of the Goal Generator. Then, we progressively roll out the pre-trained goal-based PolicyNet $a_t \sim \pi_\phi(a \mid \widehat{g}, \mathcal{O}_t)$ conditioned on $\widehat{g}$ for k timesteps, where k is the hyperparameter of the action prediction horizon. At every timestep, $\mathcal{O}_t$ is updated to the current observation. After k timesteps, we refresh the visual sub-goal by sampling from the Imaginator and Goal Generator again and then repeat the cycle.

3.2 Chain-of-Imagination

As shown in Fig. 2, repeated execution of the Imaginator and the Goal Generator forms the Chain-of-Imagination mechanism, continuously providing PolicyNet with more precise visual sub-goal embeddings for more reliable decision-making. This mechanism has two advantages:

(1) CoI allows the pre-trained decision-making foundation model to "re-decide" the subsequent actions, enhancing the agent's ability to recover from failures. Our method regenerates a new imagination every k timesteps along with the ever-changing situations, enabling successful task completion as long as subsequent sub-goal imaginations are generated correctly, even if a previous one is incorrect.

(2) CoI is easier to adapt across different embodied scenarios, thanks to the explicit modulation of the Imaginator and the Goal Generator. The Imaginator can be efficiently adapted to a new domain if a small set of video data in the target domain is provided, with off-the-shelf parameter-efficient fine-tuning (PEFT) techniques. The Goal Generator learns how to convert images (*i.e.*, current observations and imaginations) and textual instructions into goal embeddings tailored to the subsequent PolicyNet, whose training does not need the outputs generated by the Imaginator. Thus it is possible to train a powerful Imaginator that works for most embodied scenarios, including Minecraft and CALVIN. And then, solely training a PolicyNet-specific Goal Generator is fairly enough for adaptation into each scenario. Therefore, this mechanism can be efficiently adapted to different domains without a tedious end-to-end training process. We need to mention that one may also apply imaginations in a latent space to generate goal embeddings, whose design would require joint training of the Imaginator and the Goal Generator, thus not efficient when adapting to a new scenario.

3.3 Goal Drift Collection

We train the Imaginator using the Goal Drift Dataset consisting of triplets (current observation $\mathcal{O}_t$, sub-goal imagination $\mathcal{I}_t$, instructions y), collected by *Goal Drift Collection* method without tedious annotations of visual sub-goals. In existing video datasets (*e.g.*, Minecraft VPT Dataset [1], CALVIN Manipulation Dataset [30]), we can obtain the timestamps of the final goal achievement t^* when a given instruction is completed once, such as from the game records in the VPT dataset or the timestamps of the final frames in CALVIN videos.

Yet, directly pairing images $\mathcal{O}_{t^*}$ from these timestamps t^* as future goal imaginations $\mathcal{I}_t$ with images $\mathcal{O}_{t^*-T}$ from a fixed timestep T earlier as current observations $\mathcal{O}_t$, along with instruction y, could lead to two problems (Please see Fig. 3 for details): **(1) Goal Illusion: The Imaginator edits the**

observation to depict the final goal directly. Training the Imaginator on such data may reduce it to an image editor, as it directly generates the final goal without regard to the environment rather than step-by-step sub-goal imagination because all ground truths in the dataset represent the final goal. In Fig. 3 above, given the instruction "`Chop a tree`" while facing the sky, the Imaginator may unrealistically insert a broken log into the sky. **(2) Imagination Stagnation: The Imaginator fails to imagine what to do after the agent has achieved the final goal once.** For many tasks, the Imaginator needs to imagine multiple processes for instruction completion (*e.g.*, "`Chop a tree`" task needs the agent to chop down many trees). The Imaginator trained on such data only imagines the step-by-step completion of instruction once but cannot imagine what to do after achieving it (*e.g.*, find another tree to chop), as all current observations in data occur before the final goal. In Fig. 3 below, after cutting the uppermost wood by looking up, the agent will not look down for more trees, impeding multiple completion of instructions.

Algorithm 1 Backward Drift and Forward Drift

Require: final goal achievement timestamp t^*, backward times b, instruction y, imagination horizon $k_{\min}$, $k_{\max}$.
Ensure: Triplets (current observation $\mathcal{O}_t$, sub-goal imagination $\mathcal{I}_t$, instructions y).

```
Initialize list outputs to empty
t_goal ← t*
for i = 1 to b do
    I_t ← GetObservation(t_goal)
    k* ← Uniformly sample from k_min to k_max
    t_curr ← t_goal − k*
    O_t ← GetObservation(t_curr)
    t_goal ← t_curr
    Add (O_t, I_t, y) to outputs
end for
O_t ← GetObservation(t*)
k* ← Uniformly sample from k_min to k_max
t_goal ← t* + k*
I_t ← GetObservation(t_goal)
Add (O_t, I_t, y) to outputs
return outputs
```

To address the above issues, we propose the *Goal Drift Collection* method to gather Goal Drift Dataset. We form many triplets (current observation, sub-goal imagination, instruction) at each timestamp t^*, all associated with the same instructions y. Our approach has both **Backward Drift**, which helps the Imaginator implicitly understand how the instructions are achieved step-by-step to mitigate Goal Illusion, and **Forward Drift**, which enables the Imaginator to learn how to accomplish instructions repeatedly to reduce Imagination Stagnation. Below are the method details: **(1) Backward Drift**: As shown in Algorithm 1, we select PolicyNet-dependent(or dataset-dependent) imagination horizon hyperparameters k_{min} and k_{max} to control the range of the imagined distances. Here, k_{min} ensures that PolicyNet can reach the imagined state appropriately, while k_{max} aims to reduce the number of imaginations needed to complete the task. **(2) Forward Drift**: Compared to Backward Drift, Forward Drift is a reverse process. In Algorithm 1, the number of forward times is typically 1 compared to the backward times, as the subsequent processes and instructions may not be related.

In practice, for computational efficiency, we set prediction horizon hyperparameter k for inference to be similar to the corresponding k_{max} and find this sufficient for obtaining good performance. Meanwhile, we manually select k_{min} and k_{max} by observing the PolicyNet's ability and lengths of

Figure 3: **Imagination without Backward/Forward Drift.**

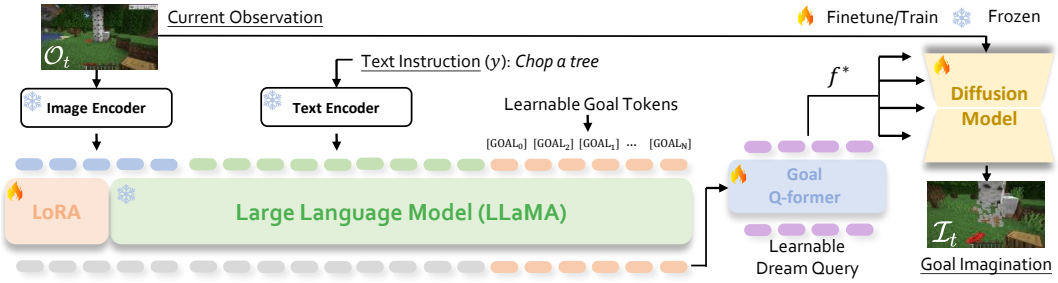


Figure 4: **The Framework of Imaginator.** We add N [GOAL] tokens to the end of the instruction and input them with current observation into LLaVA [26]. Then LLaVA generates hidden states for the [GOAL] tokens, which the Q-Former processes to produce the f^*. Next, the image encoder combines its output with f^* in the diffusion models for future sub-goal image generation.

trajectories in datasets, like SuSIE [2]. For instance, in the CALVIN dataset, where most demos span 65 steps, the pre-trained PolicyNet predicts 20 timesteps, so we set k_{min} to 20. This straightforward approach delivers reliable results without expensive re-training and evaluation.

3.4 Network Specification

Imaginator. Our Imaginator is an MLLM-enhanced diffusion model that imagines step by step based on the current state and instruction, especially from an egocentric perspective. The training data uses the Goal Drift Dataset mentioned above, consisting of (current observation $\mathcal{O}_t$, sub-goal imagination $\mathcal{I}_t$, instructions y) triplets. Below are the specific details: (1) Fine-grained Goal Understanding: In Fig. 4, the Imaginator receives current observation $\mathcal{O}_t$, encoded by a frozen image encoder, and a tokenized textual instruction y. However, the LLM's output is constrained to the language modality. Inspired by GILL [22], we bridge the language-vision modalities gap by extending the LLM's vocabulary with N Learnable Goal Tokens $[\text{GOAL}_1], \ldots, [\text{GOAL}_N]$, appending them to instruction y. We aim to minimize the negative log-likelihood of predicting the next [GOAL] token given previously generated [GOAL] tokens. We add LoRA [17] parameters into the LLM's self-attention projection layers. Only the LoRA parameters and the Learnable Goal Tokens are updated during training. The hidden states $h_{[\text{GOAL}]}$ are used to generate imaginations in the following module.

(2) Fine-grained Imagination Generation: To bridge the gap between the LLM's hidden states and the CLIP text encoder's spaces, we transform the LLM's goal tokens into semantically relevant representations f^* using a Goal Q-Former with Learnable Dream Queries inspired by BLIP2 [24] and InstructBLIP [10]. We utilize a latent diffusion model with a VAE for latent space denoising diffusion conditioned on f^*. The MLLM-enhanced diffusion architecture and the Goal Drift Dataset allow Imaginator to generate fine-grained future sub-goal imagination continuously.

Goal Generator. Goal Generator will transform sub-goal imagination into the visual embedding the PolicyNet needs. Notably, if PolicyNet's goal input only comprises an image (e.g., in the case of CALVIN), There is no need for the Goal Generator. Since PolicyNet across different domains requires diverse goals, the Goal Generator can be a variety of network structures (*e.g.*, MLP, CVAE) and is domain-specific, *i.e.*, needs to be trained on corresponding datasets.

PolicyNet. PolicyNet is a goal-based model pre-trained on large datasets, often task-agnostic, and can be not only a common Transformer-based but also Diffusion-based policy [9] or other architecture. To validate the generalizability, we employ a **video** embedding goal-based **transformer policy** in Minecraft [25] named STEVE-1(visual) and a pixel-level **image** goal-based **diffusion policy** used by SuSIE [2] in CALVIN, which are all pre-trained on their domain datasets.

4 Experiments

4.1 Experimental Setup

Training Process. The Imaginator training process is divided into three stages: (1) we train the Q-Former to align the MLLM output space with the CLIP text space. (2) we fintune the InstructPix2Pix's pre-trained weights. (3) we optimize Imaginator in an end-to-end manner. Specifically, for MLLM, only LoRA [17] weights are trained for efficient fine-tuning. For the diffusion model, we use the

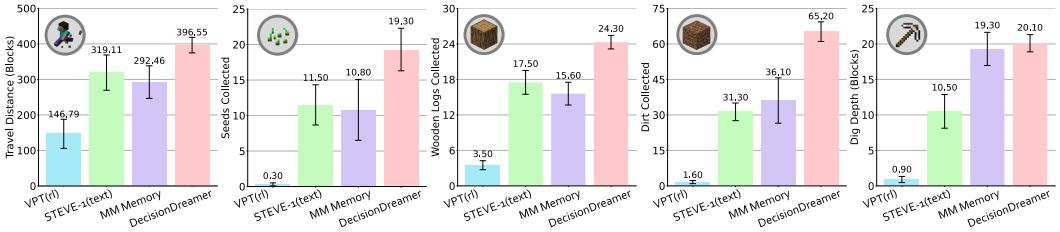


Figure 5: **Performance on Programmatic Evaluation.**

weights pre-trained in the second stage as the initial weights in Imaginator. Goal Generator for Minecraft's PolicyNet is a CVAE with three-layer MLPs, similar to the architecture of STEVE-1's prior. PolicyNet for CALVIN needs images as visual sub-goals, so no need for a Goal Generator.

Training Datasets. Our Goal Drift Dataset comprises 520k triplets (current observation, goal imagination, instruction) collected from the Minecraft VPT Dataset [1] and the CALVIN Dataset [30]. In the first stage of Imaginator, we use the CC12M dataset [8], and the Goal Drift Dataset is used in the second and third stages. For the Goal Generator in Minecraft, we follow the STEVE-1, selecting 10k quadruplets (current observation, sub-goal imagination, instruction, video sub-goal embedding) for our test tasks. The first three components are from the Goal Drift Dataset, and for the latter, we use the MineCLIP [13] video encoder to transform the sub-goal imagination and the previous 15 frames into a visual sub-goal embedding as the ground truth.

Environment Setting. We employ MineRL [16] as the Minecraft simulator. The observation space is limited to RGB images, and the action space is confined to keyboard and mouse controls. We also employ CALVIN [30] as the robot manipulation simulator. We only use RGB images as observation space, and the action space is confined to the 7-DOF of the robot arm.

Baseline. We compare three baselines in Minecraft: (1) VPT [1], a foundation model pretrained on 70k hours gameplay. Here, we select the VPT(rl), which is finetuned by reinforcement learning on the original VPT but cannot follow instructions. (2) STEVE-1 [25]. STEVE-1(visual) is a pre-trained video embedding goal-based model finetuned from VPT(rl). Here, we select STEVE-1(text), which is instruction-following by using a simple prior to map the text into the visual goal space of STEVE-1(visual) directly, without considering the current observation. (3) Multi-Modal Memory, efficiently searches through instruction-video pairs to find the most relevant video and input it into the pre-trained STEVE-1(visual), whose visual goal effectively leverages the current observation. For robot manipulation, we compare to previous methods tested on CALVIN. This includes multi-context imitation (MCIL) [28], hierarchical universal language-conditioned policy (HULC) [29], and improved variants of HULC. We also compare to other SOTA methods from [15] that employ an identical training and evaluation protocol as our experiments, namely MdetrLC [20] and AugLC [33]. Additionally, we compare to generative model methods like Unipi [12] and SuSIE [2].

Evaluation. We have three evaluations for different purposes: **(1) Programmatic Evaluation**, a quantitative evaluation in Minecraft used to evaluate an agent's ability to follow one instruction accurately. We follow the evaluation protocol from [25]. **(2) Prompt-Chaining Evaluation**, a quantitative evaluation in Minecraft designed to assess whether to execute a new instruction immediately after switching the instruction. We use the success rate as the metric for evaluation. **(3) Manipulation Evaluation**, a quantitative evaluation in CALVIN used to evaluate the method's adaptability in other domains. We study the most challenging **zero-shot multi-environment scenario**: training on A, B, and C environments, and testing on D. We follow the evaluation protocol from [30].

4.2 Performance on Textul Instructions Control

Programmatic Evaluation. We evaluate all agents on 5 tasks and plot the programmatic metric performances(mean and 95% confidence intervals). Each task runs 10 trials with distinct environment seeds, limiting 3,000 frames, following the evaluation protocol from STEVE-1[25].

We compare our *DecisionDreamer* with the previous best generalist agent in Fig. 5 and find it significantly outperforms STEVE-1(text) and Multi-Modal Memory, which all use the same PolicyNet, *i.e.*, STEVE-1(visual), with an average improvement of 1.7 $\times$ and 1.5 $\times$ in the programmatic

Table 1: **Performance on Prompt-Chaining Evaluation and Ablation on Generating Latent Goal.**

(a) Performance on Prompt-Chaining Evaluation.

Method	collect wood then craft planks	gather dirt then build a tower	dig down then mine diamonds
VPT	0.00	0.00	0.00
STEVE-1(text)	0.48	0.18	0.00
MM Memory	0.52	0.64	0.02
DecisionDreamer	**0.60**	**0.81**	**0.10**

(b) Ablation on Generating Latent Goal.

Instruction	Latent Goal directly	**Future Imaginatio (Ours)**
"Chop a tree"	20.90 ± 2.01	$\mathbf{24.30 \pm 2.59}$
"Collect dirt"	49.30 ± 7.74	$\mathbf{65.20 \pm 9.39}$
"collect seeds"	12.30 ± 6.69	$\mathbf{19.30 \pm 6.78}$

Table 2: **Performance on Manipulation Evaluation.**

Method	No. of Instructions Chained				
	1	2	3	4	5
HULC[29]	0.43	0.14	0.04	0.01	0.00
MCIL[28]	0.20	0.00	0.00	0.00	0.00
MdetrLC[20]	0.69	0.38	0.20	0.07	0.04
AugLC[33]	0.69	0.43	0.22	0.09	0.05
UniPi[12]	0.56	0.16	0.08	0.08	0.04
SuSIE[2]	0.87	0.69	0.49	**0.38**	0.26
***DecisionDreamer*(Ours)**	**0.89**	**0.69**	**0.53**	0.37	**0.29**

metric, respectively. we draw the following conclusions: (1) *DecisionDreamer* with CoI, which can break down instructions into a series of situation-aware visual sub-goals embedding, outperforms STEVE-1(text), which directly converts the instruction into a single visual goal embedding without considering the situation. (2) *DecisionDreamer* with CoI can generate future sub-goal imaginations along with the ever-changing sit- uations compared to MM Memory's retrieved result, providing more precise visual sub-goal embeddings.

Prompt-Chaining Evaluation. We also explore agents' ability to execute a new instruction immediately after switching the instruction, including (1) collect wood and then craft planks, (2) gather dirt and then build a tower and (3) dig down and then mine horizontally for diamonds, each with 50 trials. Tasks 1 and 2 limit 3,000 frames, with instructions changing to 1,500 and 2,000 frames. Task 3 limits 12,000 frames, switching instructions upon reaching the 13th floor, as diamonds are commonly found between the 7th and 14th floors.

In Tab. 1, *DecisionDreamer* surpasses STEVE-1(text) and Multi-Modal Memory baselines, which all use the same PolicyNet,*i.e.*, STEVE-1(visual), with an average improvement of 2.3 $\times$ and 1.3 $\times$ in the successful rate, respectively. This means that *DecisionDreamer* with CoI can provide a new visual sub-goal that aligns with a new instruction immediately after switching, unaffected by the previous instruction, thus enabling the execution of a new instruction right away.

Manipulation Evaluation. To show the adaptability of our method, we also test *DecisionDreamer* in the **zero-shot setting** of CALVIN (train A, B, C $\rightarrow$ test D) 100 times and calculate the average success rate. During evaluation, agents have 360 timesteps to execute a chain of 5 instructions and each environment includes 34 language-specific tasks.

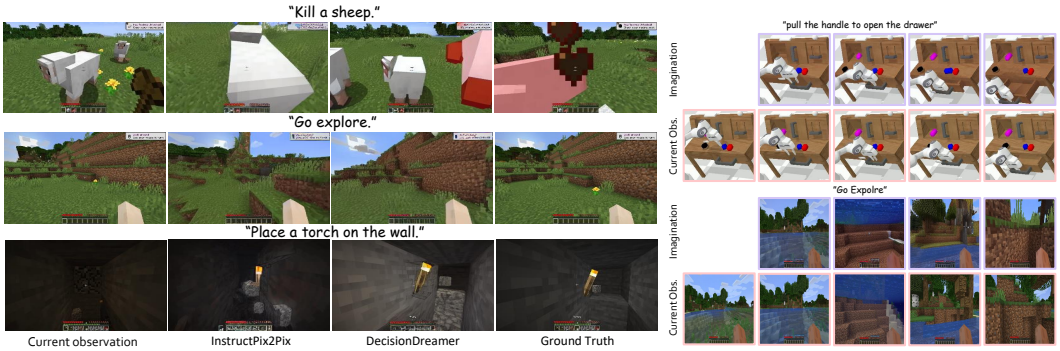
As shown in Tab. 2, we present the **zero-shot** performance of the successful rate for completing each instruction in the chain and *DecisionDreamer* also outperforms the previous state-of-the-art. We draw the following conclusions: (1) Our method can easily adapt to PolicyNet of different domains and different architectures, including Minecraft's transformer-based and robot manipulation's diffusion policy-based PolicyNet. (2) Compared to the video generative model (*i.e.*, UniPi), CoI can provide better and more efficient visual sub-goals, as the former has high computational costs and often produces videos with hallucinations and physical inconsistencies.

4.3 Qualitative Results of Imaginator

We first compare Imaginator with the common image editing model, InstructPix2Pix [5] in the open-world Minecraft. We use the same training set and test both models in the evaluation set. In Fig. 6a, we find that our model can generate sub-goal imaginations that contain more about the physical consistencies and adhere more closely to environmental knowledge from an egocentric per-

Table 3: **Ablation Study of *Goal Drift Collection* and Imaginator design.**

Instruction	Backward Once with Fixed Timestep	Only Backward Drift	Only Forward Drift	Random Noise	Instruct-Pix2Pix	**Ours**
"`Chop a tree`"	17.60 ± 3.76	10.10 ± 1.28	4.20 ± 1.38	2.70 ± 1.85	22.90 ± 2.73	$\mathbf{24.30 \pm 2.59}$
"`Collect dirt`"	38.60 ± 16.63	30.30 ± 9.59	18.10 ± 11.36	10.90 ± 6.95	59.50 ± 5.50	$\mathbf{65.20 \pm 9.39}$
"`collect seeds`"	12.20 ± 7.36	10.10 ± 5.24	3.20 ± 2.16	2.70 ± 1.00	17.90 ± 7.73	$\mathbf{19.30 \pm 6.78}$



(a) Qualitative Result of Imagination Generation.

(b) Example rollouts.

Figure 6: **Qualitative Results of Imaginator.**

spective. For example, "`Kill a sheep`" shows the color change of a sheep after being beaten by the agent, "`Go explore`" captures the agent's perspective shift, and "`Place a torch`" shows the associated increase in illumination. In Fig. 6b, we also visualize sub-goal Imagination sequences and trajectory rollouts in CALVIN and Minecraft. We observe that employing the *Goal Drift Collection* will make Imaginator break down the textual instructions into a series of intermediate visual sub-goals one-by-one and guide the PolicyNet in action generation. More demos can be found at https://sites.google.com/view/decisiondreamer.

4.4 What Contributes to Performance

Dataset Collection Method. We study different collection methods, including Backward once with fixed timestep, only Backward Drift, only Forward Drift, and *Goal Drift Collection*. In Tab. 3, we observe that the Imaginator is affected by Goal Illusion (see Fig. 3 above, the model edits the final goal, *i.e.*, a broken wood in the sky) and Imagination Stagnation(see Fig. 3 below, the model cannot imagine what to do next after chopping the highest wood) using the first three methods. This prevents the Imaginator from understanding how to complete the instruction step-by-step or repeatedly.

Imaginator Design. In Tab. 3, we observe that: (1) "Random noise" as imagination results in vague visual embeddings, which drastically reduce performance to merely 10% of the original level, which means that the precise sub-goal imagination guidance is crucial. (2) Our Imaginator outperforms the InstructPix2Pix because by leveraging MLLM, our generated images adhere more closely to physical rules and environmental knowledge, as shown in Fig. 6a.

Generate images v.s. latent goals directly. Tab. 1 shows the results of applying the latent goal tokens before the Q-former in Imaginator to condition the same PolicyNet. These goal tokens are aligned with the PolicyNet input visual goal space after re-training. Following the Programmatic Evaluation, its performance is lower. We believe visual knowledge about the environment stored in the pre-trained diffusion model would give complementary clues when making decisions and can achieve good performance using various domain video data.

5 Conclusion

In this paper, we propose a simple but effective mechanism called Chain-of-Imagination for enhancing the agent's instruction-following ability. In detail, it is designed by translating the textual instructions into a series of intermediate situation-aware visual sub-goals and allowing the pre-trained

decision-making foundation models to achieve them one-by-one. We further propose *Goal Drift Collection* method to gather data and train an MLLM-enhanced diffusion model to understand how the instructions are achieved step-by-step. The experimental results demonstrate the effectiveness of the proposed method as our agent *DecisionDreamer* equipped with this mechanism significantly outperforms the best baselines in Minecraft and CALVIN.

References

[1] B. Baker, I. Akkaya, P. Zhokov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022.

[2] K. Black, M. Nakamoto, P. Atreya, H. Walke, C. Finn, A. Kumar, and S. Levine. Zero-shot robotic manipulation with pretrained image-editing diffusion models. *arXiv preprint arXiv:2310.10639*, 2023.

[3] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[4] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[5] T. Brooks, A. Holynski, and A. A. Efros. Instructpix2pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18392–18402, 2023.

[6] S. Cai, B. Zhang, Z. Wang, X. Ma, A. Liu, and Y. Liang. Groot: Learning to follow instructions by watching gameplay videos. *arXiv preprint arXiv:2310.08235*, 2023.

[7] M. Cao, X. Wang, Z. Qi, Y. Shan, X. Qie, and Y. Zheng. Masactrl: Tuning-free mutual self-attention control for consistent image synthesis and editing. *arXiv preprint arXiv:2304.08465*, 2023.

[8] S. Changpinyo, P. Sharma, N. Ding, and R. Soricut. Conceptual 12m: Pushing web-scale image-text pre-training to recognize long-tail visual concepts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3558–3568, 2021.

[9] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[10] W. Dai, J. Li, D. Li, A. M. H. Tiong, J. Zhao, W. Wang, B. Li, P. Fung, and S. Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning, 2023.

[11] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[12] Y. Du, S. Yang, B. Dai, H. Dai, O. Nachum, J. Tenenbaum, D. Schuurmans, and P. Abbeel. Learning universal policies via text-guided video generation. *Advances in Neural Information Processing Systems*, 36, 2024.

[13] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362, 2022.

[14] T.-J. Fu, W. Hu, X. Du, W. Y. Wang, Y. Yang, and Z. Gan. Guiding instruction-based image editing via multimodal large language models. *arXiv preprint arXiv:2309.17102*, 2023.

[15] Y. Ge, A. Macaluso, L. E. Li, P. Luo, and X. Wang. Policy adaptation from foundation model feedback. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19059–19069, 2023.

[16] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. Minerl: a large-scale dataset of minecraft demonstrations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 2442–2448, 2019.

[17] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[18] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[19] Y. Huang, L. Xie, X. Wang, Z. Yuan, X. Cun, Y. Ge, J. Zhou, C. Dong, R. Huang, R. Zhang, et al. Smartedit: Exploring complex instruction-based image editing with multimodal large language models. *arXiv preprint arXiv:2312.06739*, 2023.

[20] A. Kamath, M. Singh, Y. LeCun, G. Synnaeve, I. Misra, and N. Carion. Mdetr-modulated detection for end-to-end multi-modal understanding. In *Proceedings of the IEEE/CVF international*

conference on computer vision, pages 1780–1790, 2021.
[21] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
[22] J. Y. Koh, D. Fried, and R. R. Salakhutdinov. Generating images with multimodal language models. *Advances in Neural Information Processing Systems*, 36, 2024.
[23] C. Li, R. Zhang, J. Wong, C. Gokmen, S. Srivastava, R. Martín-Martín, C. Wang, G. Levine, M. Lingelbach, J. Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023.
[24] J. Li, D. Li, S. Savarese, and S. Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*, 2023.
[25] S. Lifshitz, K. Paster, H. Chan, J. Ba, and S. McIlraith. Steve-1: A generative model for text-to-behavior in minecraft. *arXiv preprint arXiv:2306.00937*, 2023.
[26] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*, 2023.
[27] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
[28] C. Lynch and P. Sermanet. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*, 2020.
[29] O. Mees, L. Hermann, and W. Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and Automation Letters*, 7(4):11205–11212, 2022.
[30] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.
[31] OpenAI. Gpt-4v(ision) system card. 2023.
[32] A. Padalkar, A. Pooley, A. Jain, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Singh, A. Brohan, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.
[33] A. Pashevich, R. Strudel, I. Kalevatykh, I. Laptev, and C. Schmid. Learning to augment synthetic images for sim2real policy transfer. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2651–2657. IEEE, 2019.
[34] Y. Qin, E. Zhou, Q. Liu, Z. Yin, L. Sheng, R. Zhang, Y. Qiao, and J. Shao. Mp5: A multi-modal open-ended embodied system in minecraft via active perception. *arXiv preprint arXiv:2312.07472*, 2023.
[35] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.
[36] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.
[37] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.
[38] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.
[39] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.
[40] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.
[41] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.
[42] H. R. Walke, K. Black, T. Z. Zhao, Q. Vuong, C. Zheng, P. Hansen-Estruch, A. W. He, V. Myers, M. J. Kim, M. Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.
[43] M. Wen, R. Lin, H. Wang, Y. Yang, Y. Wen, L. Mai, J. Wang, H. Zhang, and W. Zhang. Large sequence models for sequential decision-making: a survey. *Frontiers of Computer Science*, 17(6):176349, 2023.

[44] H. Yuan, Z. Mu, F. Xie, and Z. Lu. Pre-training goal-based models for sample-efficient reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024.

[45] A. Zeng, M. Attarian, B. Ichter, K. Choromanski, A. Wong, S. Welker, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

[46] K. Zhang, L. Mo, W. Chen, H. Sun, and Y. Su. Magicbrush: A manually annotated dataset for instruction-guided image editing. *arXiv preprint arXiv:2306.10012*, 2023.

Chain-of-Imagination for Reliable Instruction Following in Decision Making

Supplementary Material

The supplementary document is organized as follows:

- More details about the Chain-of-Imagination, Imaginator, and Hyperparameter.
- Environment Setting, like observation and action space.
- Dataset composition and collection.
- Implementation Details, like training details.
- Experiment Details, like baseline and evaluation details.
- More Experiments about *DecisionDreamer*.
- More Visualizations in Minecraft and CALVIN.
- Demo videos in Minecraft and CALVIN.
- Reproducibility Checklist.

A Chain-of-Imagination Details

We show the Chain-of-Imagination mechanism implementation details in Algorithm 2. We also show the hyperparameters used for *Goal Drift Collection* and inference in Tab. 4.

Algorithm 2 Chain-of-Imagination mechanism

Model and Input: Imaginator $\mathcal{F}_\theta(\mathcal{I}_t \mid \mathcal{O}_t, y)$, Goal Generator $\mathcal{G}_\xi(g_t \mid \mathcal{I}_t, \mathcal{O}_t, y)$, PolicyNet $\pi_\phi(a \mid g, \mathcal{O}_t)$, current observation $\mathcal{O}_t$, textual instruction y, time limit T, prediction horizon hyperparameter k.

```
t ← 0
while t ≤ T do
    Sample Î_t ~ F_θ(I_t | O_t, y)
    if PolicyNet's input g needs an image then
        ĝ ← Î_t
    else
        ĝ ~ G_ξ(g | Î_t, O_t, y)
    end if
    for i ← 1 to k do
        Sample a_t ~ π_φ(a | ĝ, O_t)
        Execute a_t
        t ← t
        Update O_t
    end for
end while
```

B Imaginator Details

Given a current observation $\mathcal{O}_t$ and a textual instruction y, the Imaginator generates a fine-grained future goal imagination $\mathcal{I}_t$ for the Goal Generator. In Fig. 3 in the main paper, current observation $\mathcal{O}_t$ is encoded by a frozen image encoder $\mathbf{E}_v$ into $\mathbf{E}_v(O_t)$, textual instruction y is tokenized into $(x_1, ..., x_T)$, they are sent to the LLM together. Imaginator can now acquire a fine-grained goal imagination of the instruction intention but are limited to the language modality. Inspired by GILL [22], we bridge the language-vision modalities gap by extending the LLM's vocabulary with N Learnable Goal Tokens $[\mathrm{GOAL}_1], \ldots, [\mathrm{GOAL}_n]$, appending them to instruction y. Specifically, a trainable

matrix $\mathbf{E}_g$, representing these [GOAL] embeddings, is added to the LLM's embedding matrix. We aim to minimize the negative log-likelihood of predicting the next [GOAL] token given previously generated [GOAL] tokens:

$$\mathcal{L}_{\text{LLM}} = -\sum_{i=1}^{k} \log p_{\{\theta_L \cup \theta_l \cup \mathbf{E}_g\}}([\text{GOAL}_i] \mid \mathbf{E}_v(O_t), x_1, ..., x_T, [\text{GOAL}_1], \ldots, [\text{GOAL}_{i-1}]) \tag{1}$$

We add LoRA [17] parameters θ_l into the LLM's self-attention projection layers for efficient fine-tuning while keeping all LLM parameters θ_L frozen. During training, only the LoRA [17] parameters θ_l and the Learnable Goal Tokens $\mathbf{E}_g$ are updated. The hidden states $h_{[\text{GOAL}]}$ corresponding to $\mathbf{E}_g$ tokens are used to generate imaginations in the following module.

B.0.1 Fine-grained Goal Imagination Generation via Latent Imagination.

To address the disparity between the LLM's hidden states and the CLIP [35] text encoder's feature spaces, we must transform the LLM's sequential goal tokens into semantically relevant representations for guiding goal imagination generation. Inspired by BLIP2 [24] and InstructBLIP [10], we employ a Goal Q-Former $\mathcal{Q}$ with several Learnable Dream Query, to derive the fine-grained goal imagination representation f^*:

$$f^* = \mathcal{Q}\left(h_{[\text{GOAL}]}\right) \tag{2}$$

Drawing from InstructPix2Pix's [5] latent diffusion approach, a cornerstone in instruction-based image editing, our model introduces noise to the latent encoding $z = \mathcal{E}(\mathcal{I}_t)$ of the goal imagination $\mathcal{I}_t$ through encoder $\mathcal{E}$, yielding a noisy latent z_s across timesteps $s \in S$. A U-Net [40] ϵ_δ is trained to estimate this noise, conditional on the current observation $c_o = \mathcal{E}(O_t)$ and text instruction c_T, by merging c_o with z_s. The specific process can be formulated as follows:

$$\mathcal{L}_{\text{dream}} = \mathbb{E}_{\mathcal{E}(\mathcal{I}_t), \mathcal{E}(O_t), c_T, \epsilon \sim \mathcal{N}(0,1), s}[\|\epsilon - \epsilon_\delta(s, \text{concat}[z_s, \mathcal{E}(O_t)] + f^*)\|_2^2] \tag{3}$$

where ϵ is unscaled noise, s is the sampling step, z_s is latent noise at step s, $\mathcal{E}(O_{t_n})$ is the current observation condition, and c_T is the text instruction condition. The concat corresponds to the concatenation operation.

Table 4: **The Hyperparameters for *Goal Drift Collection* and inference.**

Hyperparameter Name	Dataset	
	Minecraft	CALVIN
k_{min}	20	20
k_{max}	25	22
k	25	20
backward_times	3	3
forward_times	1	1

C Minecraft Environment

Minecraft is a widely popular sandbox game that offers players the freedom to build and explore their worlds without limits, which also extends to AI agents as well. Within the game, AI agents encounter situations that closely mirror real-world challenges, requiring them to make decisions and solve endless tasks in an open-world setting. Consequently, Minecraft is an ideal platform for AI evaluation and stands as an exemplary benchmark for AI testing, due to its vast freedom and open nature. With the help of Minecraft, AI researchers can more easily simulate a wide variety of complex and dynamic environments and tasks, allowing them to conduct experiments that enhance the practical and applicable value of AI technologies.

We use MineRL [16] v1.0, which corresponds to Minecraft 1.16.5, as our simulation platform, ensuring an environment that is consistent with those used by VPT [1] and STEVE-1 [25]. In this version of MineRL [16], a significant advancement over its predecessor (*i.e.*, MineRL v0.4.4), lies

https://github.com/minerllabs/minerl/releases/tag/v1.0

Table 5: **Action Space utilized in the MineRL [16] simulator.** The action space primarily consists of 14 keyboard and mouse operations, with detailed descriptions sourced from the Minecraft wiki (https://minecraft.fandom.com/wiki/Controls).

Index	Action	Human Action	Description
1	Forward	key W	Move forward.
2	Back	key S	Move backward.
3	Left	key A	Strafe left.
4	Right	key D	Strafe right.
5	Inventory	key E	Open or close GUI inventory.
6	Drop	key Q	Drop a single item from the stack of items the player is currently holding.
7	Jump	key Space	Jump. When in the water, it keeps the player afloat.
8	Sneak	key left Shift	Move slowly in the current direction of movement.
9	Sprint	key left Ctrl	Move fast in the current direction of movement.
10	Attack	left Mouse Button	Destroy blocks (hold down); Attack entity (click once); Pick up the stack of items or place the stack of items in the GUI (click once)
11	Use	right mouse Button	Place the item being held or interact with the block that the player is currently looking at.
12	Hotbar.[1-9]	keys 1 - 9	Switch the appropriate hotbar cell.
13	Yaw	move Mouse X	Turning; aiming; camera movement.Ranging from -180 to +180.
14	Pitch	move Mouse Y	Turning; aiming; camera movement.Ranging from -180 to +180.

in the simulation environment. The environment now enables AI agents to interact in a manner entirely consistent with human players, eschewing primitive actions or script-based APIs. This approach presents a more complex and challenging scenario for AI research. More specifically, AI agents experience the environment as humans do, solely through egocentric RGB images, devoid of any privileged in-game information. Additionally, their interactions with the environment are restricted to low-level keyboard and mouse actions. Consequently, AI agents trained in this version of MineRL [16] (*i.e.*, MineRL v1.0) resemble embodied agents capable of performing various tasks in an open-world environment, demonstrating a higher degree of generalization. Furthermore, the abundance of gaming videos available on the internet (*e.g.*, YouTube), provides AI researchers with the opportunity to harness these vast datasets for extensive pre-training, enabling the development of a foundation model in the sequential decision-making domain.

C.1 Observation Space

Our observation space aligns with that of human players, comprising simply the raw pixels from Minecraft. This includes the hotbar, health indicators, player hands, equipped items, and the game environment itself. Specifically, the simulator produces RGB images with a resolution of 640x360. When the agent takes action within the environment, the simulator renders the player's first-person perspective with a field of view of 70 degrees. If the agent opens the inventory, the simulator will render the GUI interface along with the mouse cursor.

Notably, we do not employ privileged information such as voxels and lidar information available in MineDojo [13], which could be provided to the agent. During actual inference, the PolicyNet of *DecisionDreamer* **only accepts the raw RGB pixels observations as input** that the agent can obtain from the environment and generates text-conditioned low-level action controls based on these observations, which are consistent with those used in VPT [1] and STEVE-1 [25].

C.2 Action Space

As shown in Tab. 5, our action space encompasses a vast array of actions that are consistent with those of human players (*i.e.*, keyboard and mouse), including keypresses, mouse movements, and clicks. Keyboard presses and mouse clicks are binary functional actions (*e.g.*, "`Forward`", "`Back`", "`Left`", "`Right`" and *etc.*). Beyond these binary input options, our action space also has mouse cursor movements. While the GUI is closed (*i.e.*, activated by pressing "E" for the GUI inventory)

and remains inactive, the mouse's horizontal and vertical movements direct the agent's yaw and pitch. Conversely, with GUI open, the same movements are re-purposed to navigate the cursor across the display.

It is noteworthy that we have not employed structured APIs such as "`craft`" and "`smelt`" as seen in MineDojo [13], which replace the need for precise mouse movements that are necessary for interacting with the inventory for certain tasks, effectively turning these operations into GUI functional binary actions. During actual inference, our *DecisionDreamer*'s PolicyNet **only outputs keyboard and mouse actions** to dictate the agent's movements, aligning these actions with those utilized in VPT [1] and STEVE-1 [25].

C.3 Environment Settings and Rules

In our experiments, the agent's *initial position* at the start of the game, as well as the *seed* used to generate the environment, are completely random. This introduces an element of unpredictability and variety into the experimental setup, ensuring that the agent will encounter a wide range of scenarios and challenges.

To better evaluate the agent's ability to follow textual instructions for action prediction and its ability to rapidly adapt its behavior based on instructions, we have modified MineRL [16] to enable "`chat`" action operations. This allows for the swift initialization of the agent with predefined conditions through instructions. Specifically, for Programmatic Evaluation, we ensure that each experiment for all agents is conducted with the same seed and within the biome most conducive to completing the current instruction; across multiple experiments, different seeds are used. For Prompt-Chaining Evaluation for Long-Horizon Tasks, all agents are placed in the same seed and biome optimal for the current instruction. In addition, the following rules are applied as aids:

- `/difficulty peaceful`: Set the difficulty of the environment to peaceful mode.
- `/gamerule doDaylightCycle false`: Set the environment to daytime forever.
- `/gamerule keep inventory true`: Set agent to not drop items upon death.

Specifically, for the task of "Obtain diamonds" , we add two additional rules on top of the ones above as assistance:

- `/effect give @a night_vision 99999 250 true`: Help the agent see more clearly in extremely dark environments (*e.g.*, at night or underground).
- `/give @p minecraft:diamond_pickaxe`: Provided the agent with a diamond pickaxe , enabling it to break almost all blocks and mine all ores within Minecraft.

D CALVIN Environment

CALVIN [30] is a benchmark designed for evaluating long-horizon, language-conditioned manipulation tasks. It comprises four distinct simulated environments, labeled A through D. Each environment features a dataset of human-collected play trajectories to aid in benchmarking. Each simulated setting is equipped with a Franka Emika Panda robot arm situated adjacent to a desk that holds various manipulatable objects. These include a drawer, a sliding cabinet, a light switch, and an assortment of colored blocks. The environments are differentiated by unique table textures, varying placements of the furniture, and diverse configurations of the colored blocks. In our experiments, we **exclusively use RGB images for the observation space**, while **the action space is limited to the 7 degrees of freedom (DOF) of the robot arm**.

E Dataset Details

E.1 CALVIN Robot Manipulation Dataset

CALVIN features four unique simulated environments, labeled A through D. Each environment is accompanied by a dataset of human-collected play trajectories, designed to facilitate effective benchmarking. Approximately 35% of this play data is annotated with descriptive language, which is utilized to train our Imaginator module. The benchmark is particularly focused on the challenging

Table 6: **The detailed event name and description in MineRL [16] simulator.** The simulator records the names of events that occur as well as related information, including quantities. We can use these events to collect a large amount of data for completing event-related instruction tasks with clarity.

Event Name	Description
`mine_block`	The moment the agent breaks a block, the type of block is recorded.
`craft_item`	The moment the agent crafts items, the type and number of items are recorded.
`use_item`	The moment the agent uses or places items, the type of item is recorded.
`kill_entity`	The moment the agent kills an entity, the type of entity is recorded.
`break_item`	The moment the tool of the agent is broken, the type of tool is recorded.
`pick_up`	The moment the agent picks up items, the type and number of items are recorded.

scenario of zero-shot multi-environment learning; we train the models in environments A, B, and C, and subsequently test their performance in environment D. It is important to note that in our simulation experiments, we do not employ any video-only datasets from environment D.

E.2 OpenAI Contractor Gameplay Dataset

All our raw data are based on the contractor dataset, which consists of offline trajectory data in Minecraft used for training VPT [1]. This dataset is created by hiring human contractors to play Minecraft and complete predetermined tasks, and it includes video (*i.e.*, image sequences), corresponding action sequences and metadata. OpenAI releases six subsets of contractor data: 6.x, 7.x, 8.x, 9.x, 10.x, and the MineRL BASALT 2022 dataset. Our Goal Drift Dataset ultimately selects three of these subsets as our raw data, including 8.x (house building from scratch), 10.x (↗), and the `FindCave` dataset from the MineRL BASALT 2022 dataset. For each video, there is an associated metadata file that not only records the contractor's actions for every frame but also documents events triggered by the contractor within the simulator; the specific events are detailed in Tab. 6.

E.3 Event Selection

In constructing our dataset, we opt to **select events directly from the MineRL simulator and supplement them with manually annotated events**. Specifically, to train the Imaginator within the constraints of limited resources, we focus on the following types of events: "`mine_block`", "`craft_item`", "`use_item`", "`kill_entity`" and a manually defined event named "`easy_action`". Details of the specific items selected for each event can be found in Tab. 7. The simulator's built-in events have a clearly defined completion time t^*, while manually annotated events are marked with a manually labeled completion time.

Table 7: **Details of the specific items selected for each event.** We select four built-in events from the simulator, along with a manually defined event called "`easy_action`". The built-in events have a clearly defined completion moment, while the collection of the "`easy_action`" event is manually annotated.

Event Name	`mine_block`	`craft_item`	`use_item`	`kill_entity`	`easy_action`
Detail items	Wooden Log Grass Dirt Grass Block Sand Snow Stone Coal Ore Iron Ore Redstone Ore Diamond Ore	wooden planks	torch	sheep	Go explore Dig down Look at the sky Go Swimming Stay underwater Build a tower Mine horizontally

https://github.com/openai/Video-Pre-Training

E.4 Dataset Collection

After obtaining the completion times t^* for all events, we employ `gpt-4-turbo` [31] to generate corresponding event-related instructions. Specifically, we provide `gpt-4-turbo` [31] with the event's name, description, and detailed items, and prompt it to generate multiple distinct simple instructions. These instructions include specific actions, while others mention the items to be obtained upon completing the action. For instance, for "`Grass`" in the "`mine_block`" event, `gpt-4-turbo` [31] would generate instructions like "break grass", "break tall grass", "gather seeds", and "collect seeds". After gathering instructions for all events, we apply the *Goal Drift Collection* method to conduct backward and forward drift on the completion times t^* of event-related instructions. For each pair (current observation, goal imagination), there are many instructions created by `gpt-4-turbo` [31] to describe that event. This process results in a substantial collection of triplets (current observation, goal imagination, instruction), which serve as training data for the Imaginator, forming what we call the Goal Drift Dataset. The final Goal Drift Dataset contains approximately 500,000 triplets (current observation, goal imagination, instruction), with about 400,000 of these triplets derived from events built into the simulator.

We follow the method used in STEVE-1 [25] for training the CVAE [41; 21] and collect a subset of approximately 10,000 quadruplets from the Goal Drift Dataset for the events we need to test subsequently. This subset consists of quadruplets where the current observation, goal imagination, and instruction are consistent as conditions with the Goal Drift Dataset. Additionally, there is a visual prompt embedding that serves as ground truth. This embedding is derived from a video composed of the goal imagination and the preceding 16 frames, processed through the MineCLIP [13] video encoder.

F Implementation Details

F.1 Imaginator

The training process of Imaginator is divided into three main stages. In the first stage, the MLLM is aligned with the CLIP [35] text encoder using the QFormer [24]. In the second stage, we apply InstructPix2Pix [5] to warm up the weights for the diffusion model in Minecraft. In the third stage, we optimize Imaginator in an end-to-end manner. To be specific, the weights of LLaVA [26] are frozen and LoRA [17] is added for efficient fine-tuning. For the diffusion model, we directly use the weights pre-trained in the second stage as the initial weights in Imaginator.

For the Large Language Model with visual input (e.g., LLaVA [26]), we choose LLaVA-1.1-7b [26] as the base model. During training, the weights of LLaVA are frozen and we add LoRA for efficient fine-tuning. We expand the original LLM vocabulary with 32 new tokens. The QFormer is composed of 6 transformer layers and 77 learnable query tokens. We use the AdamW optimizer [27] in all three stages. In the initial stage of training, we configure the learning rate and weight decay parameters at 2e-4 and 0, respectively. The training targets for this stage encompass a dual-objective framework, comprising the Mean Squared Error (MSE) loss between the outputs of LLaVA [26] and the CLIP [35] text encoder, alongside the language model loss. Both losses are assigned equal weights of 1. The training setting in the second is the same as InstructPix2Pix [5]. In the final stage, the settings for the learning rate, weight decay, and warm-up ratio are adjusted to 1e-5, 0, and 0.001, respectively. During this phase, the loss function is diffusion loss. The hyperparameters used during the training are listed in the following Tab.8.

F.2 Goal Generator

In CALVIN, PolicyNet is an image diffusion policy that requires pixel-level images as visual goals. Therefore, we don't need to specifically design a Goal Generator to bridge and align images to visual goals.

In Minecraft, we require a Goal Generator that can transform imagination into the video embeddings needed for PolicyNet (STEVE-1(visual) [25]) goal input. Our Goal Generator in Minecraft is mainly a conditional variational autoencoder (CVAE) [41; 21] with a Gaussian prior and a Gaussian posterior. Both the encoder and decoder of CVAE [41; 21] are parameterized as three-layer MLPs with 512 hidden units and layer normalization. It encodes the current observations, goal imaginations,

Table 8: **The Hyperparameters of Imaginator.**

Hyperparameter Name	Value
base_model	LLaVA [26]
input_image_size	256×256
expand_vocabulary_num	32
transformer_layers_num	6
QFormer_learnable_query_num	77
optimizer	AdamW
learning_rate_initial_stage	2e-4
weight_decay_initial_stage	0
learning_rate_final_stage	1e-5
weight_decay_final_stage	0
warm-up_ratio_final_stage	0.001
n_iterations_initial_stage	5000
n_iterations_final_stage	10000

and instructions then reconstructs a latent video goal embedding aligning the input goal space of our PolicyNet. The video goal embedding's space is the visual space of MineCLIP [13], where MineCLIP [13] is a pre-trained CLIP model that employs a contrastive objective on pairs of Minecraft videos and associated transcripts from the web. Specifically, the process of generating goals by the Goal Generator mainly involves two steps. First, we stack the current observation and the goal imagination 16 times each to create two static 16-frame videos. These are then processed through MineCLIP [13]'s video encoder to obtain two visual embeddings. Concurrently, the instruction is encoded into a text embedding using MineCLIP [13]'s text encoder. This ensures that all embeddings are encoded within the MineCLIP [13] space. We then train a CVAE [41; 21] using the ELBO loss, which generates a latent visual embedding from the previous three embeddings. This representation is a video embedding that describes the process within the MineCLIP [13] visual space. This representation is a video embedding that captures the process within the MineCLIP [13] visual space. The ground truth is derived from the goal imagination and the preceding 16 frames, which have been processed through the MineCLIP [13] video encoder. For each event to be evaluated subsequently, we train a CVAE [41; 21] on the dataset, specifically for 150 epochs with early stopping on a small validation set. Notably, the parameters of the MineCLIP [13] within Prompt Generator remain unchanged. The hyperparameters used during the training are listed in the following Tab.9.

F.3 PolicyNet

In CALVIN, we use the pre-trained diffusion policy that uses pixel-level images as visual goals to be the PolicyNet. The model's architecture and weight are the same as SuSIE [2]. PolicyNet's implementation, which aligns with the methodology described in [42], involves channel-wise stacking of observation and goal images prior to processing through a ResNet-50 image encoder. This encoded image data then conditions a diffusion process tailored to model the action distribution. The diffusion model is structured around a multi-layer perceptron (MLP) that includes three layers, each containing 256 units, and incorporates residual connections for enhanced learning dynamics. Adopting the approach from [9], PolicyNet extends beyond predicting a single action. Instead, it forecasts a sequence of four actions, a strategy aimed at promoting temporal consistency within the action sequences.

In Minecraft, we use the pre-trained transformer-based policy that uses MineCLIP [13] video embeddings as visual goals to be the PolicyNet. The model's architecture and weight are the same as STEVE-1(visual) [25]. More details can be found in the baseline details below.

G Experiment Details

In this section, we first detail the three baselines we select. We then separately present the Programmatic Evaluation details and the Prompt-Chaining Evaluation for Long-Horizon Tasks details.

Table 9: **The Hyperparameters of CVAE within Goal Generator.**

Hyperparameter Name	Value
architecture	MLP
visual_prompt_dim	512
text_dim	512
current_img_dim	512
goal_img_dim	512
hidden_layers	3
batch_size	256
learning_rate	1e-4
β	0.001
n_epochs	150

Table 10: **The detailed settings for the Programmatic Evaluation.**

Id	Text Instruction	Biome	Time Limit	Prediction Horizon k	Metric
1	`go explore`	Plains	3000 Frames	25 Frames	Travel Distance (Blocks)
2	`collect seeds`	Plains			Seeds Collected
3	`chop a tree`	Forest			Wooden Logs Collected
4	`collect dirt`	Plains			Dirt Collected
5	`dig down`	Plains			Dig Depth (Blocks)

G.1 Baseline Datails

Video Pretraining (VPT) [1] is the first foundation model in the Minecraft domain, pre-trained on 70k hours of gameplay by Baker et al. [1]. Its architecture primarily consists of two parts: ImpalaCNN and TransformerXL [11]. VPT has three variants: VPT(fd), VPT(bc), and VPT(rl), representing the vanilla foundation model, the behavior cloning fine-tuned model, and the RL fine-tuned model, respectively. Specifically, they initially pre-trained on a large corpus of YouTube videos using a behavior cloning algorithm to obtain VPT(fd), which is capable of free exploration within the environment. This model gains a fundamental understanding of the environment and acquires some environmental knowledge. To enhance the agent's capability in completing early-game tasks (*e.g.*, "Collect wood" and "Craft wooden planks", they collect an "Early-Game" video dataset and fine-tune the VPT(fd) to obtain VPT(bc). This model performs well in early-game tasks but struggles with long-horizon tasks, such as obtaining diamonds. Building on VPT(bc), they employ online reinforcement learning with carefully designed rewards to fine-tune the model, enabling it to complete the task of obtaining diamonds from scratch, ultimately resulting in the creation of VPT(rl). Hence, it is noteworthy that **all three variants of VPT are unable to follow instructions**; they must first be fine-tuned on downstream tasks before they can be completed. Despite their extensive environmental knowledge, this knowledge cannot be unlocked by instruction-following capabilities. In our experiments, we use VPT(rl) because it initially seeks out trees and gathers wood, a critical step in the pathway to obtaining diamonds. When set in the appropriate biome, VPT(rl) explores further and collects more wood compared to VPT(fd) and VPT(bc).

STEVE-1 [25] is a Minecraft agent that can follow both textual and visual instructions, built upon MineCLIP [13] and VPT. Drawing from the paradigms of instruction tuning in large language models and multimodal large language models, it successfully unlocks the instruction-following abilities

Table 11: **The detailed settings for the Prompt-Chaining Evaluation.**

Id	Text Instruction	Biome	Switch Condition	Time Limit	Prediction Horizon k
1	`chop a tree` `craft wooden planks`	Forest	Reach 1500 Frames	3000 Frames	25 Frames
2	`collect dirt` `build a tower`	Plains	Reach 2000 Frames	3000 Frames	25 Frames
3	`dig down` `mine horizontally`	Plains	Reach 13th floors	12000 Frames	25 Frames

of the foundation model (*i.e.*, VPT) in the domain of decision-making. STEVE-1 comes in two variants, STEVE-1(visual) and STEVE-1(text). The training process is divided into two steps. The first step involves training a goal-based policy conditioned on future video as visual instructions using the packed hindsight relabeling method. Specifically, they utilize the OpenAI Contractor Gameplay Dataset to fine-tune VPT(rl) to follow visual instructions, resulting in STEVE-1(visual). **The STEVE-1(visual) is the final pre-trained goal-based decision-making model in Minecraft.** The second step is to train a model that can map text instructions to visual instructions. Inspired by UnCLIP, they trained a Conditional Variational Autoencoder (CVAE) on a dataset of video-text pairs they collected, thus obtaining STEVE-1(text) which can follow text instructions. It is important to note that **STEVE-1(text) does not consider the current observation when generating the visual goal given the textual instruction. The visual goal remains unchanged throughout the task, serving as an initial guide without adapting to environmental changes**.

Multi-Modal Memory serves as a substitute for the Imaginator and Grompt Generator in the *DecisionDreamer*framework, essentially functioning by supplying PolicyNet with visual goals that best align with the current observations and textual instructions. We construct a multi-modal memory comprised of numerous video-text pairs. This memory is specifically built upon the triplets (current observation, goal imagination, instruction) from the Goal Drift Dataset. By tracing back 16 frames from the timestamp of the goal imagination, we create a 16-frame video segment, resulting in a revised triplet format: (current observation, goal imagination video, instruction). Each event, whether from the MineRL [16] environment or manually defined, contains 1,000 pairs. The retrieval process is as follows: First, we encode the current instruction and all instructions in the multi-modal memory using the OpenCLIP [36] text encoder to obtain embeddings. We then compare these embeddings using cosine similarity. Next, within the memory corresponding to the text instruction with the highest similarity, we find the match where the current observation and the memory's observation, once encoded through the OpenCLIP [36] Image encoder, have the highest cosine similarity in their embeddings. Finally, the video from the final retrieval result is then encoded using the MineCLIP [13] video encoder, and the resulting visual embedding is used as the final visual goal. Therefore, **Multi-Modal Memory leverages the current observation when getting the visual goal**.

G.2 Programmatic Evaluation Datails

In this part, we will elaborate on the selection of experimental tasks for Programmatic Evaluation, the methodology for calculating evaluation metrics, and the specific details of the experimental setup.

For the Programmatic Evaluation, we evaluate the agents on five **single-step** instruction tasks derived from the *early-game evaluation suite* proposed in Table 3 of the STEVE-1 [25] appendix. The purpose of this evaluation is to quantitatively measure an agent's ability to follow instructions with minimal human intervention. Specifically, we calculate the programmatic evaluation metrics by monitoring the state of the MineRL [16] environment during each evaluation episode. Consistent with VPT [1] and STEVE-1 [25], we compute multiple programmatic metrics, including travel distance, dig depth, and early-game item collection. The calculation is as follows:

1. `Travel Distance (Blocks)`: The agent's maximum horizontal displacement, in the X-Z plane, is measured from the initial spawn point.

2. `Dig Depth (Blocks)`: The agent's maximum vertical (Y-axis) displacement is measured from its initial spawn point.

3. `Early-Game Inventory Counts`: The maximum number of log, seed, and dirt items seen in the agent's inventory during the episode.

We test all agents on these five **single-step** instruction tasks, with each task running 10 episodes of 3000 timesteps (*i.e.*, 2.5 minutes of gameplay). Each episode used a unique environmental seed, yet all agents were tested under the same seed for consistency. It is important to note a key difference in our experimental setup compared to STEVE-1 [25]: **for each task, we initialize the agents in the biome most conducive to task completion** to enhance the reliability of our evaluation metrics. For instance, in the "`Chop a tree`" task, all agents are spawned in a forest biome, rather than a plain, to avoid the added randomness of searching for trees before chopping them. The detailed settings for the Programmatic Evaluation can be found in Tab. 10.

Figure 7: **The Generalizability of *DecisionDreamer*. (Left)** Despite excluding data involving 'Dirt' or 'Dig' from Goal Drift Dataset and retraining, Imaginator can still generate relatively high-quality imaginations aligned with the instruction's concept. **(Right)** The retrained Imaginator remains operational with the CoI mechanism and can handle unseen instructions while largely preserving its previous performance.

G.3 Prompt-Chaining Evaluation Datails

In this part, we will also detail the selection of experimental tasks for Prompt-Chaining Evaluation for Long-Horizon Tasks, the calculation methods for evaluation metrics, and the specific details of the experimental setup.

The Prompt-Chaining Evaluation for Long-Horizon Tasks comprises three **multi-step** instructions tasks sourced from the *early-game evaluation suite* of STEVE-1 [25], except the "`Obtain diamonds`" task which originates from GROOT [6], designed to steadily follow video instructions. These tasks aim to evaluate an agent's ability to swiftly adapt to new instructions following an instruction switch, a critical capability for a downstream controller operating under an LLM-based high-level planner. We employ success rate as the performance metric, also by monitoring the MineRL [16] environment state throughout each evaluation episode. The criteria for determining success across the three different tasks are as follows:

1. `collect wood` `and then craft planks`: Success is defined as successfully crafting at least one wooden log into four wooden planks within the given time frame.

2. `gather dirt` `and then build a tower`: Success is defined as successfully building a tower with a height of at least 7 blocks within the given time frame.

3. `dig down` `and then mine horizontally`: Success is obtaining at least one diamond within the given time frame.

For these three **multi-step** instructions tasks, we run 50 episodes of testing per task. The time limit for the first two tasks is set at 3000 frames (*i.e.*, 2.5 minutes of gameplay), consistent with STEVE-1 [25], while the final task has an episode time limit of 12,000 frames (*i.e.*, 10 minutes of gameplay), aligning with what is mentioned in the main paper of GROOT [6]. Each episode utilizes a unique environmental seed to ensure variability; however, all agents are tested with the same seed for consistency across episodes. It is important to note that our experimental setup differs from that of STEVE-1 [25] in that **we initialize the agents in the biome most conducive to task completion** for each task. Specifically, we utilize the "`chat`" action to initialize the agent. For the "`Obtain diamonds`" task, we equip the agent with night vision and a diamond pickaxe, which is consistent with the description provided in the main paper of GROOT [6]. **Considering that STEVE-1 [25] may not be explicitly trained on the "`mine horizontally`" instruction, we augment STEVE-1 [25]'s prior original training data with the corresponding text-video pairs from the Goal Drift Dataset and retrain the prior**. This ensures that the updated prior can map the textual instruction "`mine horizontally`" to the associated visual instructions. The detailed settings for the Prompt-Chaining Evaluation for Long-Horizon Tasks experiment can be found in Tab. 11.

G.4 Manipulation Evaluation Datails

In our study, we adhere to the evaluation protocol outlined in [30] for the CALVIN benchmark. Our method is assessed in a zero-shot setting, where the model is trained on three environments (A, B, and C) and tested on a fourth environment (D). Each trial allocates 360 timesteps to the policy to execute a sequence of five instructions, progressing to the next only after successfully completing the preceding one. Following the completion of each instruction, subsequent instructions are dynamically selected by the simulator based on the specific affordances of the environment.

The results are averaged over 100 trajectories, facilitating direct comparisons with the findings from [15], which similarly averages results over 100 trajectories. In contrast, results for UniPi (HiP) and UniPi [12] are averaged over 25 trajectories. This adjustment is due to the high computational demands associated with querying video diffusion models and the necessity for frequent regeneration given the short video generation horizon of 10 timesteps. The prediction horizon hyperparameter k is 20.

H More Experiments about *DecisionDreamer*

In this part, we will discuss whether the Imaginator of *DecisionDreamer* has good generalizability to solve OOM (Out of Distribution), as the agent's ability to generalize is key to its behavior in the open world where environments are complex and instructions vary widely. Since STEVE-1 [25] has shown its prior ability to map text to visual prompts effectively, and our Grompt Generator is built upon it, we will now concentrate on the generalizability of our Imaginator and the entire agent. At first, we exclude data related to the words '`Dirt`' or '`Dig`' from the Goal Drift Dataset and retrain the model. Then, we observe the images generated in response to the instruction "`Collect dirt`" based on the current state and the quantity of dirt collected by the agent.

As shown in fig. 7, we find that even after completely removing the concepts of '`Dirt`' or '`Dig`', Imaginator is still able to generate goal imaginations of relatively good quality aligned with the instruction's concept (*i.e.*, **agent points towards the dirt and attempt to break it**), which can still guide the PolicyNet to follow instructions. The resulting collection of dirt is about 70% of the original amount, which shows that the Imaginator can respond to unseen novel instructions while largely maintaining its previous performance. We attribute this to three key factors: **(1)** The MLLM within Imaginator has the relevant environmental knowledge to map the text '`Dirt`' to its corresponding element in Minecraft images, recognizing its visual counterpart; **(2)** training data for related tasks, such as "`Collect seeds`", enables the MLLM to comprehend the meaning of action '`Collect`' in Minecraft task; **(3)** The pre-trained Diffusion model can generalize to the Minecraft domain and generate goal imaginations by leveraging the MLLM's latent representations for understanding textual semantics mentioned above from the instructions.

I More Visual Results

I.1 Imaginations and trajectory rollouts in CALVIN

In Fig. 9, we visualize an example of Imagination and trajectory rollouts on a successful 5-instruction language chain in CALVIN, where the robot arm must complete the previous instruction before sequentially executing the next one.

I.2 Imaginations without Goal Drift in Minecraft

To evaluate the efficacy of the *Goal Drift Collection* method, we carry out experiments comparing various data collection approaches. Fig. 10 illustrates the imagination generated by the Imaginator trained on data collected without any goal drift. Due to the absence of backward drift, all imaginations generated by the Imaginator correspond to the moment when the event-related instructions are completed. Consequently, this leads to the phenomenon of "Goal Illusion", where the Imaginator edits the current observation to depict the completed instruction. For the instruction "`Chop a tree`", when the agent faces the sky, the Imaginator may unrealistically insert a broken wooden log into the sky. For the instruction "`Collect dirt`", even though the agent is pointing at a stone, the Imaginator still imagines dirt and shatters it, resulting in the agent eventually attempting

to break the stone. Fig. 11 shows the imaginations generated by the Imaginator trained on data collected without forward drift. Because there is no forward drift, all imaginations generated by the Imaginator represent moments before the completion of event-related instructions. This results in the phenomenon of "Imagination Stagnation", where the Imaginator fails to conceive repeated task completion. For the instruction "`Chop a tree`", after cutting the uppermost wood by looking up, the agent will not look down for more trees, which impedes continuous task performance. In contrast, an Imaginator trained with data collected including forward drift is able to understand that the agent should now look down to find other trees to continue the task.

I.3 Imaginations on Evaluation Set in Minecraft

We compare Imaginator with the existing common image editing model, namely InstructPix2Pix [5]. Given this model has been trained on specific datasets, its performance would inevitably be sub-optimal if directly applied to the Minecraft domain. To facilitate a fair comparison, we fine-tune InstructPix2Pix [5] using the same training set employed by the Imaginator and evaluate the performance of the fine-tuned models in addressing tasks in Minecraft. Fig 12 shows qualitative results in the evaluation set, our methodology exhibits enhanced abilities in Goal Imagination Generation within intricate scenarios.

I.4 Imaginations During Solving Tasks in Minecraft

We visualize the agent's imagination during task execution alongside the next observation in Fig. 13 and Fig. 14 to evaluate the Imaginator's generalization capability in open scenarios. It is observed that the Imaginator is capable of generating high-quality visualizations that closely align with the current scene in an open environment, thereby guiding the subsequent PolicyNet to autoregressively predict the next action accurately.

I.5 User Studies

To further evaluate *DecisionDreamer*'s efficacy, we conduct a user study. Specifically, we randomly select 15 images from the evaluation set, representing a wide range of tasks and scenarios within Minecraft and CALVIN. For each image, we generate results using both InstructPix2Pix [5] and *DecisionDreamer*, then randomly shuffle the order of these results. As noted in the main paper, InstructPix2Pix [5] is fine-tuned on the same dataset as *DecisionDreamer*. This process yields 15 sets of images in a shuffled sequence. Participants are asked to independently identify the two superior images for each set: the first being the one that best matches the given instructions (named **Instruct-Alignment**), and the second being the image that most closely mirrors real-world appearances, including perspective and physical laws (named **Image Quality**). A total of 25 individuals participate in the study. The findings, illustrated in Fig. 8, reveal that over 69.40% of participants find *DecisionDreamer*'s outputs to be more aligned with the instructions, and more than 70.31% favor the results produced by *DecisionDreamer* for their realism. These outcomes further underscore *DecisionDreamer*'s instruction following ability and generalization ability.

J Demo Videos

Demo videos are all hosted on the project webpage: https://sites.google.com/view/decisiondreamer.

J.1 Programmatic Evaluation

We demonstrate videos of the four tasks from the Programmatic Evaluation on the aforementioned project webpage. Of course, you can also view the demo videos for the respective tasks by directly accessing the video URLs.

- "`Go explore`": https://youtu.be/UdG0ckoGRCY
- "`Collect seeds`": https://youtu.be/TFchu_YBiuI
- "`Chop a tree`": https://youtu.be/Sx_NKjq5DTA
- "`Collect dirt`": https://youtu.be/7TOR0SOFaB8

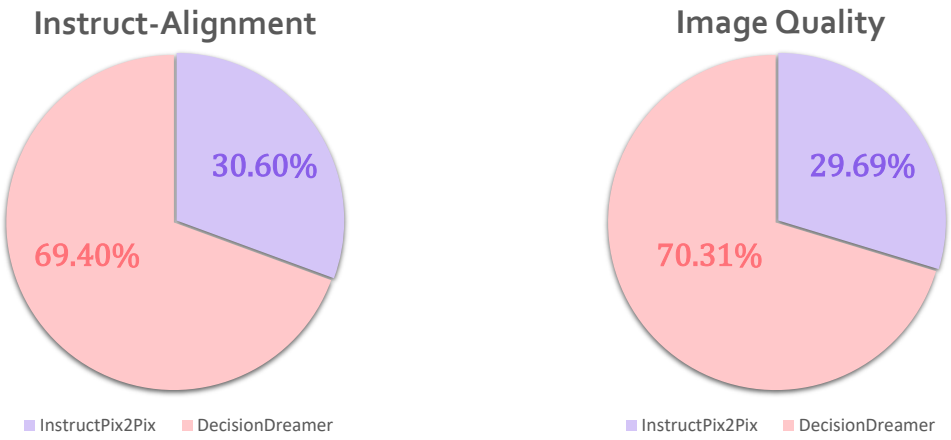


Figure 8: **The results of user studies**, comparing the results generated by InstructPix2Pix and *DecisionDreamer*. Based on the results from both the Instruction Alignment and Image Quality perspectives, *DecisionDreamer* demonstrates superior effectiveness.

J.2 Prompt-Chaining Evaluation

We demonstrate videos of the three tasks from the Prompt-Chaining Evaluation on the above project webpage. Of course, you can also view the demo videos for the respective tasks by accessing the video URLs.

- "`Chop a tree` `to Craft planks`": https://youtu.be/YtY2M_Hi7OE
- "`Gather dirt` `to Build a tower`": https://youtu.be/Zy2t2RpeNtQ
- "`dig down` `to Obtain Diamond`": https://youtu.be/hThbWh0q5EE

J.3 Manipulation Evaluation

We demonstrate the videos of imagination and trajectory rollout from the Manipulation Evaluation on the above project webpage.

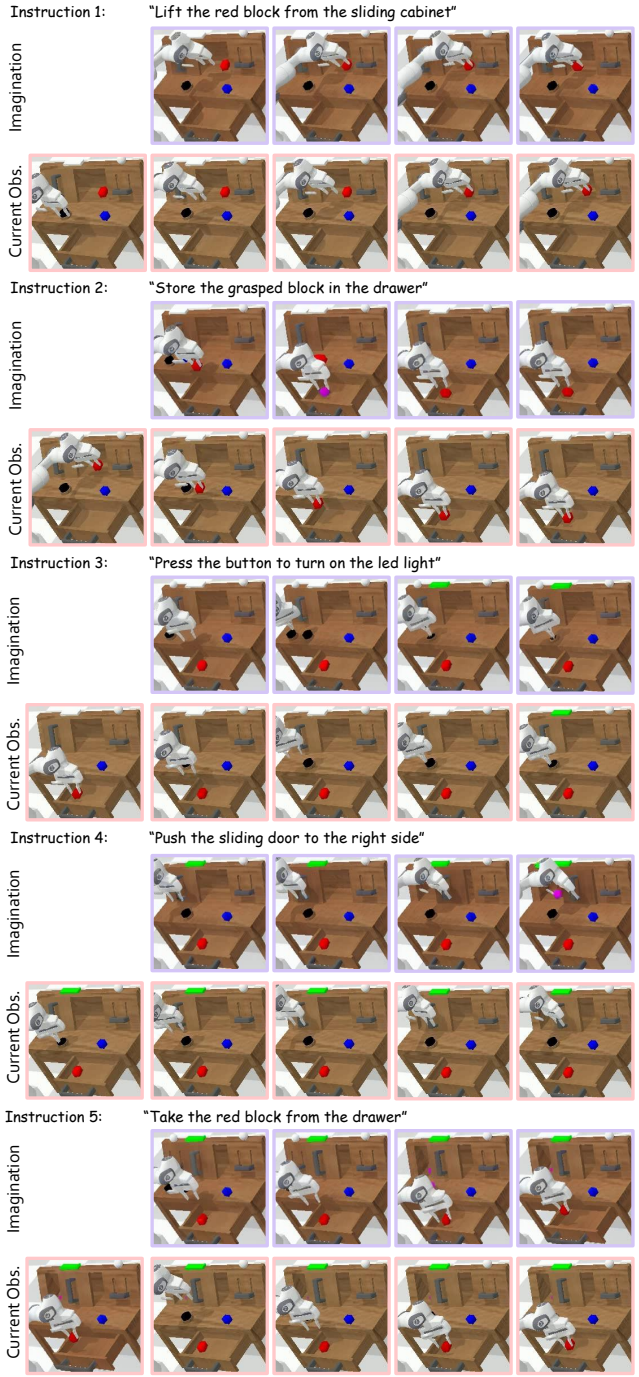


Figure 9: **Imaginations and trajectory rollouts on a successful 5-instruction language chain in CALVIN.**

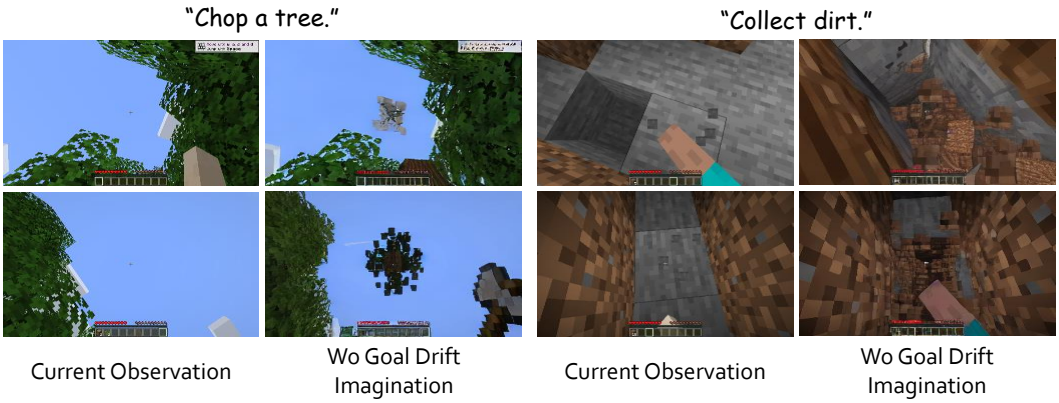


Figure 10: **Imagination Visual Results without Goal Drift.** Due to the absence of goal drift, the imaginations generated by the Imaginator are all related to the moment of event-related instruction completion, leading to the phenomenon known as "Goal Illusion", where the Imaginator edits the current observation to represent the executed instruction. In the figure depicted, the agent inserts broken wooden blocks into the sky and, facing a stone, imagines itself breaking dirt.

Figure 11: **Imagination Visual Results without Forward Drift.** Due to the lack of forward drift, the imaginations produced by the Imaginator are all from moments prior to the completion of event-related instructions, resulting in a phenomenon called "Imagination Stagnation". This means the Imaginator fails to anticipate the outcomes of repeated tasks. For example, in the figure provided, after the agent cuts the uppermost wood by looking up, it will not look down for more trees to continue the task.

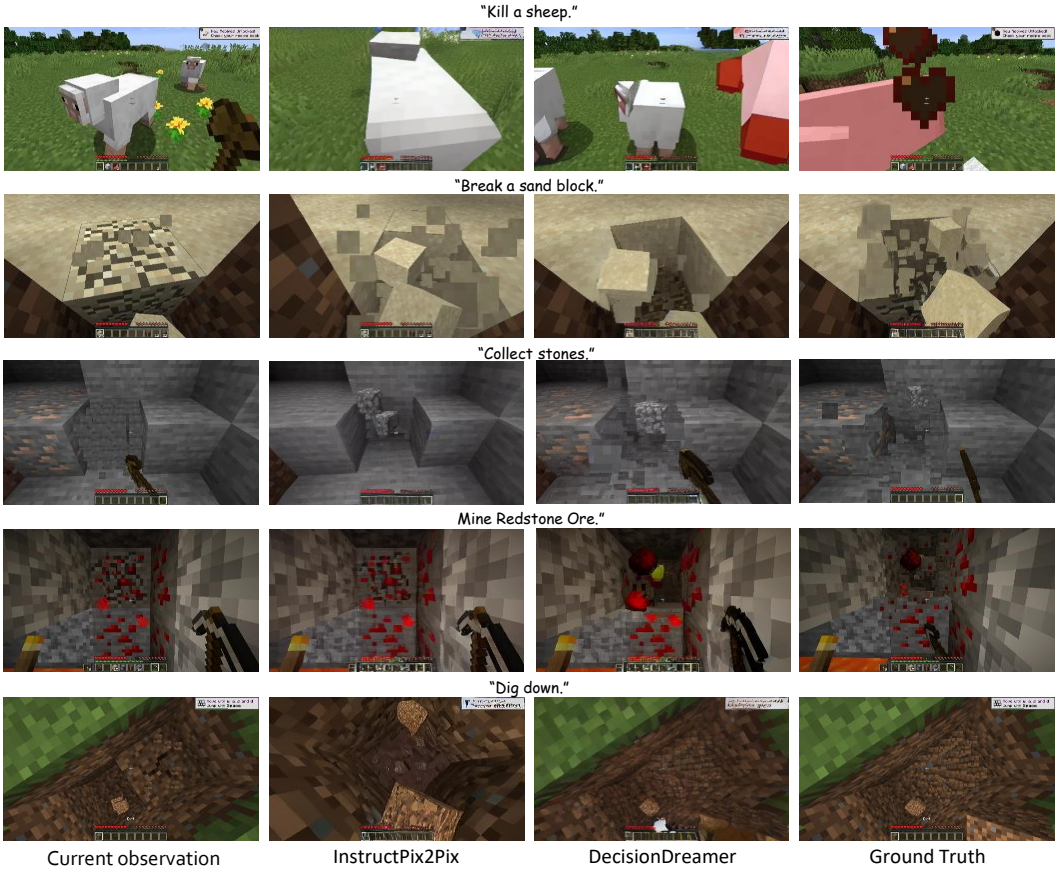


Figure 12: **Imagination visual results on Goal Drift Evaluation Set.**

Figure 13: **Imagination visual results during agent solving tasks.**

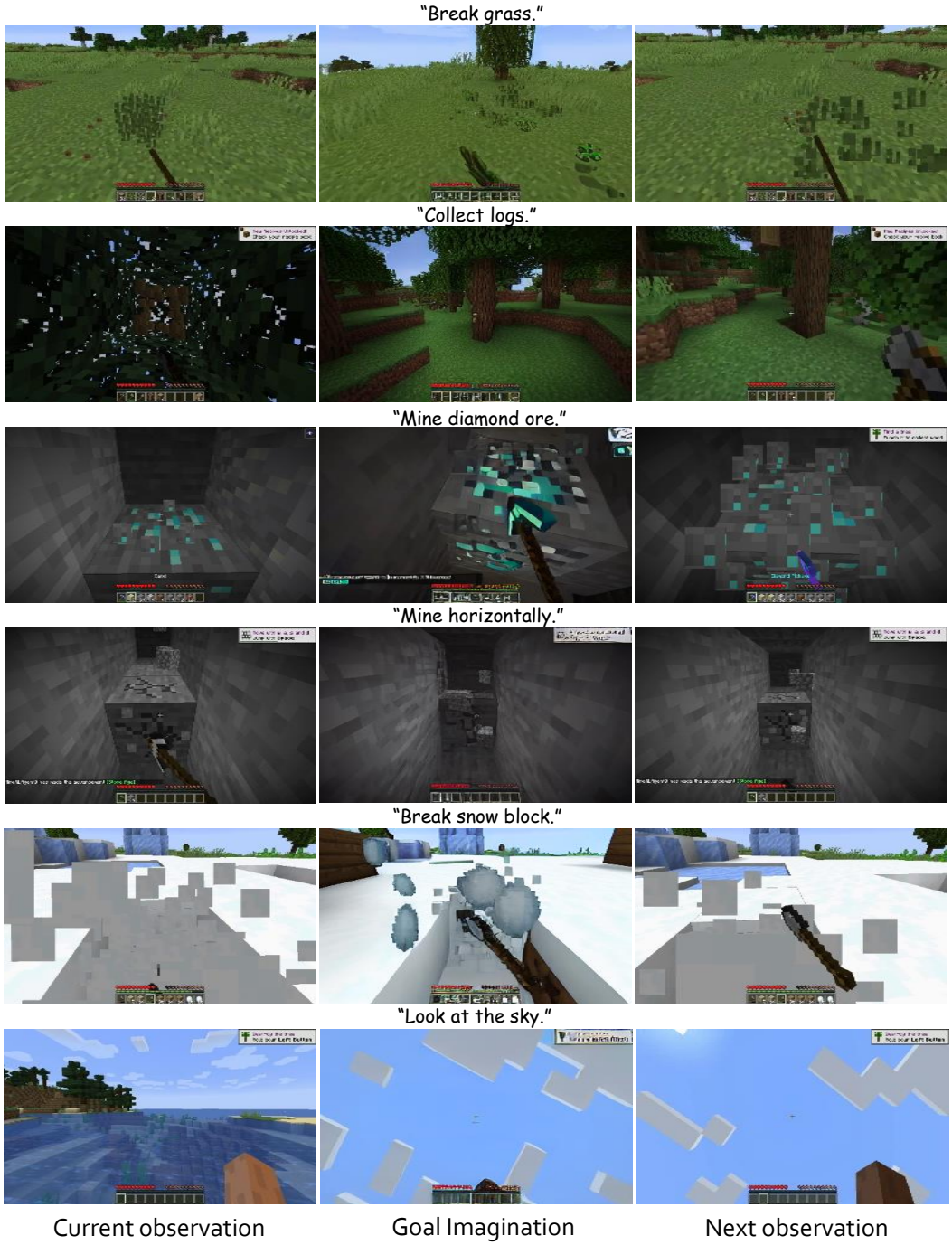


Figure 14: **Imagination visual results during agent solving tasks.**