GROOT: Learning to Follow Instructions by Watching Gameplay Videos

Shaofei Cai[1,2], **Bowei Zhang**[3], **Zihao Wang**[1,2], **Xiaojian Ma**[5], **Anji Liu**[4], **Yitao Liang**[1*]
Team CraftJarvis
[1]Institute for Artificial Intelligence, Peking University
[2]School of Intelligence Science and Technology, Peking University
[3]School of Electronics Engineering and Computer Science, Peking University
[4]Computer Science Department, University of California, Los Angeles
[5]Beijing Institute for General Artificial Intelligence (BIGAI)
{caishaofei,zhangbowei,zhwang}@stu.pku.edu.cn
xiaojian.ma@ucla.edu,liuanji@cs.ucla.edu,yitaol@pku.edu.cn
https://craftjarvis-groot.github.io

Abstract

We study the problem of building a controller that can follow open-ended instructions in open-world environments. We propose to follow reference videos as instructions, which offer expressive goal specifications while eliminating the need for expensive text-gameplay annotations. A new learning framework is derived to allow learning such instruction-following controllers from gameplay videos while producing a video instruction encoder that induces a structured goal space. We implement our agent GROOT in a simple yet effective encoder-decoder architecture based on causal transformers. We evaluate GROOT against open-world counterparts and human players on a proposed **Minecraft SkillForge** benchmark. The Elo ratings clearly show that GROOT is closing the human-machine gap as well as exhibiting a 70% winning rate over the best generalist agent baseline. Qualitative analysis of the induced goal space further demonstrates some interesting emergent properties, including the goal composition and complex gameplay behavior synthesis.

1 Introduction

Developing human-level embodied agents that can solve endless tasks in open-world environments, such as Minecraft [15, 13], has always been a long-term goal pursued in AI. Existing goal-conditioned policies use task indicator [33], future outcome [9, 19], and language [4] to represent the goal. While it is easy to learn a controller with some of these goal specifications, they may not be expressive enough for diverse tasks. Taking future outcome goals as an example, an image of a desired house clearly lacks procedural information on how the house was built. One exception is language, but learning a controller that can receive language goals is prohibitively expensive as it requires a huge number of trajectory-text pairs with text that precisely depicts the full details of the gameplay, therefore preventing them from scaling up to more open-ended tasks. Having observed the limitations of goal specification in the prior works, this paper seeks to find a balance between the capacity of goal specification and the cost of policy learning. Concretely, we propose to specify the goal as a reference gameplay video clip. While such video instruction is indeed expressive, there are two challenges: 1) How can the controller understand the actual goal being specified as the video

[*]Corresponding Author.

NeurIPS 2023 workshop on GCRL.

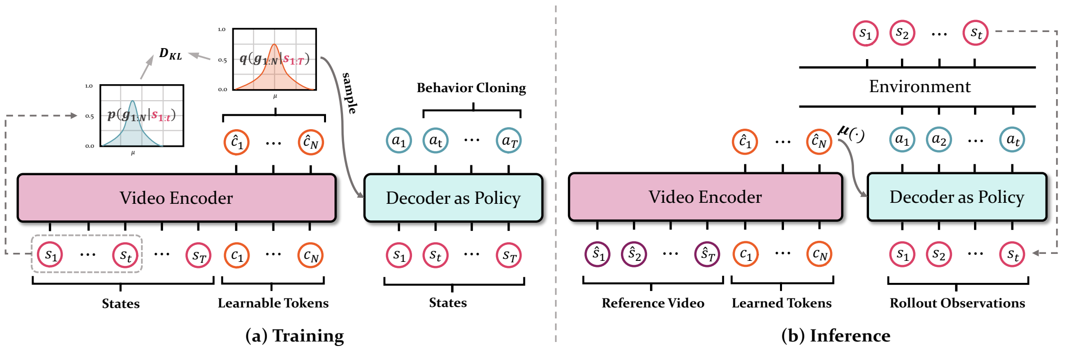


Figure 1: **Our GROOT agent architecture. Left:** In the training stage, a video encoder (non-causal transformer) learns to extract the semantic meaning and transfer the video (state sequence) into the goal embedding space. A goal-conditioned policy (causal transformer) is learned to predict actions following the given instructions. We learn the agent using behavior cloning under a KL constraint. **Right:** During the inference, any reference video can be used to generate the goal embeddings that drives the policy to interact with the environment.

itself can be ambiguous, i.e. a goal space or video instruction encoder has to be learned; 2) How to ultimately map such goal to actual motor commands? To this end, we introduce a learning framework that simultaneously produces a goal space and a video instruction-following controller from gameplay videos. The fundamental idea is casting the problem as future state prediction based on past observations: (1) The predicting model needs to identify which goal is being pursued from the past observations, which requires a good goal space (induced by a video instruction encoder); (2) Since the transition dynamics model is fixed, a policy that maps both the state and the recognized goal to action is also needed by the predicting model when rolling the future state predictions. More details are in Appendix C. Effectively, this results in the goal space and control policy we need. We introduce a variational learning objective for this problem, which leads to a combination of a cloning loss and a KL regularization loss. Based on this framework, we implement GROOT, an agent with an encoder-decoder architecture to solve open-ended Minecraft tasks by following video instructions. As shown in Figure 1, the video encoder is a non-causal transformer that extracts the semantic information expressed in the video and maps it to the latent goal space. The controller policy is a decoder module implemented by a causal transformer, which decodes the goal information in the latent space and translates it into a sequence of actions in the given environment states.

To comprehensively evaluate an agent's mastery of skills, we designed a benchmark called **Minecraft SkillForge**. The benchmark covers six common Minecraft task groups: `collect`, `build`, `survive`, `explore`, `tool`, and `craft`, testing the agent's abilities in resource collection, structure building, environmental understanding, and tool usage, in a total of 30 tasks. We calculate Elo ratings among GROOT, several counterparts, and human players based on human evaluations. Our experiments showed that GROOT is closing the human-machine gap and outperforms the best baseline by 150 points (or 71% winning rate) in an Elo tournament system. Our qualitative analysis of the induced goal space further demonstrates some interesting emergent properties, including the goal composition and complex gameplay behavior synthesis.

2 Architecture Design and Training Strategy

We implement GROOT with transformer-based encoder-decoder architecture, as shown in Figure 1. The video encoder is a non-causal transformer that extracts semantic information and generates goal embeddings. The policy is a causal transformer decoder that receives the goal embeddings as the instruction and autoregressively translates the state sequence into a sequence of actions. Next, we describe how each module is constructed together with the training strategy.

Video Encoder. The video encoder includes a CNN to extract spatial information from image states $s_{1:T}$ and a non-causal transformer to capture temporal information from videos. Specifically, we use a CNN backbone to extract visual embeddings $\{x_{1:T}\}$ for all frames. Additionally, motivated by [11], we construct a set of learnable embeddings (or summary tokens), represented as $\{c_{1:N}\}$, to capture the semantic information present in the video. The visual embeddings and summary tokens are passed to a non-causal transformer, resulting in the output corresponding to the summary tokens

$$\begin{aligned} x_{1:T} &\leftarrow \texttt{Backbone}(s_{1:T}), \\ \hat{c}_{1:N} &\leftarrow \texttt{Transformer}([x_{1:T}, c_{1:N}]). \end{aligned} \tag{1}$$

Similar to VAE [17], we assume that the latent goal space follows a Gaussian distribution, hence we use two fully connected layers, $\mu(\cdot)$ and $\sigma(\cdot)$, to generate the mean and standard deviation of the distribution, respectively. During training, we use the reparameterization trick to sample a set of embeddings $\{g_{1:N}\}$ from the distribution, where $g_t \sim \mathcal{N}(\mu(\hat{c}_t), \sigma(\hat{c}_t))$. During inference, we use the mean of the distribution as the goal embeddings, i.e. $g_t \leftarrow \mu(\hat{c}_t)$.

Decoder as Policy. To introduce our policy module, we start with VPT [3], a Minecraft foundation model trained with standard behavioral cloning. It is built on Transformer-XL [10] that can leverage long-horizon historical states and predict the next action seeing the current observation. However, the vanilla VPT architecture does not support instruction input. To condition the policy on goal embeddings, we draw the inspiration from Flamingo [1], that is, to insert *gated cross-attention dense* layers into every Transformer-XL block. The keys and values in these layers are obtained from goal embeddings, while the queries are derived from the environment states

$$\begin{aligned} \hat{x}^{(l)}_{1:t} &\leftarrow \texttt{GatedXATTN}(kv = g_{1:N}, q = x^{(l-1)}_{1:t}; \theta_l), \\ x^{(l)}_{1:t} &\leftarrow \texttt{TransformerXL}(qkv = \hat{x}^{(l)}_{1:t}; \theta_l), \\ \hat{a}_t &\leftarrow \texttt{FeedForward}(x^{(M)}_t), \end{aligned} \tag{2}$$

where the policy reuses the visual embeddings extracted by the video encoder, i.e. $x^{(0)}_{1:t} = x_{1:t}$, the policy consists of M transformer blocks, θ_l is the parameter of l-th block, $\hat{a}_t$ is the predicted action. Since our goal space contains information about how to complete a task that is richer than previous language-conditioned policy [7, 19], the cross-attention mechanism is necessary. It allows the GROOT to query the task progress from instruction information based on past states, and then perform corresponding behaviors to complete the remaining progress.

Training. The training dataset can be a mixture of Minecraft gameplay videos and offline trajectories. For those videos without actions, an inverse dynamic model [3] can be used to generate approximate actions. Limited by the computation resources, we truncated all the trajectories into segments with a fixed length of T without using any prior. We denote the final dataset as $\mathcal{D} = \{(x_{1:T}, a_{1:T})\}_M$, where M is the number of trajectories. We train GROOT in a fully self-supervised manner while the training process can be viewed as self-imitating, that is, training GROOT jointly using behavioral cloning and KL divergence loss

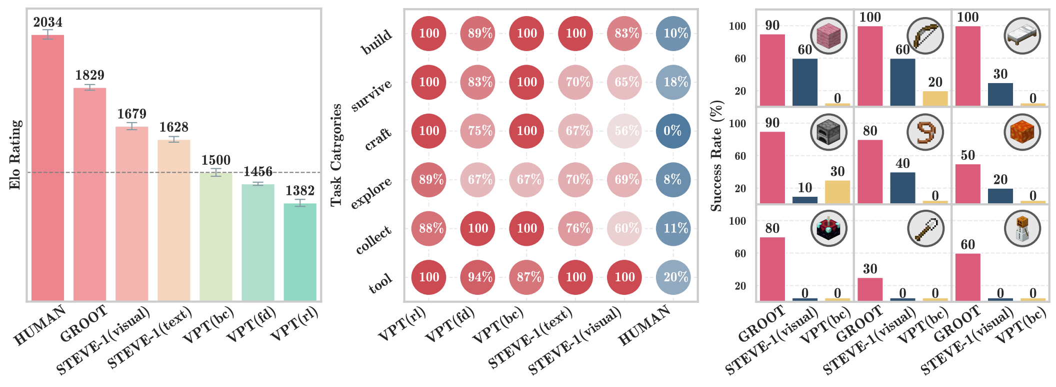
$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\frac{\lambda_{bc}}{T} \sum_t -\log \pi_\theta(a_t|s_{1:t}) + \lambda_{kl} \sum_\tau D_{KL}\left(q_\phi(g|s_{0:T}) \parallel p_\psi(g|s_{0:\tau})\right) \right], \tag{3}$$

where $\lambda_{bc}, \lambda_{kl}$ are tradeoff coefficients, q_ϕ is the visual encoder, p_ψ is a prior video encoder.

3 Experimental Results

Minecraft SkillForge Benchmark. In order to comprehensively evaluate the mastery of tasks by agents in Minecraft, we created a diverse benchmark called **Minecraft SkillForge**. It covers 30 tasks from 6 major categories of representative skills in Minecraft, including `collect`, `explore`, `craft`, `tool`, `survive`, and `build`. For example, the task "dig three down and fill one up" in the `build` category asks the agent to first dig three blocks of dirt, then use the dirt to fill the space above; The task of "building a snow golem" () requires the agent to sequentially stack 2 snow blocks () and 1 carved pumpkin (). We put the details of this benchmark in the Appendix M. Apart from some relatively simple or common tasks such as "collect wood" and "hunt animals", other tasks require the agent to have the ability to perform multiple steps in succession.

We compare GROOT with the following baselines: (a) VPT [3], a foundation model pre-trained on large-scale YouTube data, with three variants: VPT (fd), VPT(bc), and VPT(rl), indicating vanilla foundation model, behavior cloning finetuned model, and RL finetuned model; (b) STEVE-1 [19], an instruction-following agent finetuned from VPT, with two variants: STEVE-1 (visual) and STEVE-1 (text) that receives visual and test instructions. It is worth noting GROOT was trained from scratch.



(a) Elo Rating Comparison **(b) Winning Rate of GROOT vs. Baselines** **(c) Success Rate Comparison**

Figure 2: **Results on Minecraft SkillForge benchmark. Left:** Tournament evaluation of GROOT assessed by human players. GROOT performs better than state-of-the-art Minecraft agent STEVE-1. A 150-score gap corresponds to a 70% probability of winning. **Middle:** Winning rate of GROOT v.s. other agents on specific task categories. Colors from red to blue denote a decrease in the winning rate. Apart from the human player, GROOT surpasses all other baselines. **Right:** Success rate on 9 representative tasks. GROOT champions process-oriented tasks, such as `dig three down and fill one up` () and `build snow golems` ().

Human Evaluation with Elo Rating. We evaluated the relative strength of agents by running an internal tournament and reporting their Elo ratings, as in [21]. Before the tournament, each agent is required to generate 10 videos of length 600 on each task. Note that, all the reference videos used by GROOT are generated from another biome to ensure generalization. Additionally, we also invited 3 experienced players to do these tasks following the same settings. After the video collection, we asked 10 players to judge the quality of each pair of sampled videos from different agents. Considering the diversity of tasks, we designed specific evaluation criteria for every task to measure the quality of rollout trajectories. For example, in the task of "build snow golem", we rank the completion degree of the task in ascending order: no blocks placed, one type of block placed, two types of blocks placed, and snow golem built successfully. After 1500 comparisons, the Elo rating converged as in Figure 2 (left). Although there is a large performance gap compared with human players, GROOT has significantly surpassed the current state-of-the-art STEVE-1 series and condition-free VPT series on the overall tasks. Additional details are in Appendix L.

In Figure 2 (middle), we compare GROOT with other baselines in winning rate on six task groups. We found that except for the performance on `craft` tasks, where STEVE-1 (visual) outperforms our model, GROOT achieves state-of-the-art results. In particular, GROOT greatly outperforms other baselines by a large margin on `build` and `tool`. For `build`, the goal space needs to contain more detailed procedural information, which is the disadvantage of methods that use future outcomes as the goal. Moreover, such tasks are distributed sparsely in the dataset, or even absent in the dataset, which requires the agent to have strong generalization ability. As for `craft` group, GROOT is not superior enough, especially on the "crafting table" task. We attribute this to the wide task distribution in the dataset. Thus the future outcomes can prompt STEVE-1 to achieve a high success rate.

Programmatic Evaluation. To quantitatively compare the performance of the agents, we selected 9 representative tasks out of 30 and reported the success rate of GROOT, STEVE-1 (visual), and VPT (bc) on these tasks in Figure 2 (right). We found that, based on the success rate on tasks such as `dye and shear sheep`(), `enchat sword` (), `smelt food` (), `use bow` (), `sleep` (), and `lead animals` (), GROOT has already reached a level comparable to that of human players (100%). However, the success rates for `build snow golems` () and `build obsidian` () tasks are only 60% and 50%. By observing the generated videos, we found that GROOT cannot precisely identify the items in Hotbar (such as buckets, lava buckets, snow blocks, and pumpkin heads), resulting in a low probability of switching to the correct item. STEVE-1 also has the same problem. This may be due to the current training paradigm lacking strong supervisory signals at the image level. Future work may introduce auxiliary tasks such as vision-question answering (VQA) to help alleviate this phenomenon. For more details, please refer to Appendix J.

References

[1] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan. Flamingo: a visual language model for few-shot learning. *ArXiv*, abs/2204.14198, 2022. 3, 14

[2] Y. Aytar, T. Pfaff, D. Budden, T. L. Paine, Z. Wang, and N. de Freitas. Playing hard exploration games by watching youtube. In *Neural Information Processing Systems*, 2018. 8

[3] B. Baker, I. Akkaya, P. Zhokhov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *ArXiv*, abs/2206.11795, 2022. 3, 8, 9, 12, 13, 14, 15, 16

[4] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. J. Joshi, R. C. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K.-H. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath, I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. S. Ryoo, G. Salazar, P. R. Sanketi, K. Sayed, J. Singh, S. A. Sontakke, A. Stone, C. Tan, H. Tran, V. Vanhoucke, S. Vega, Q. H. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich. Rt-1: Robotics transformer for real-world control at scale. *ArXiv*, abs/2212.06817, 2022. 1, 8, 13

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. J. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020. 8

[6] J. Bruce, A. Anand, B. Mazoure, and R. Fergus. Learning about progress from experts. In *International Conference on Learning Representations*, 2023. 8

[7] S. Cai, Z. Wang, X. Ma, A. Liu, and Y. Liang. Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13734–13744, 2023. 3, 8, 12

[8] M. Chang, A. Gupta, and S. Gupta. Semantic visual navigation by watching youtube videos. *ArXiv*, abs/2006.10034, 2020. 8

[9] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Neural Information Processing Systems*, 2021. 1, 8

[10] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Jan 2019. 3, 14

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019. 2, 8

[12] H. Du, X. Yu, and L. Zheng. Vtnet: Visual transformer network for object goal navigation. *ArXiv*, abs/2105.09447, 2021. 8

[13] L. J. 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. *ArXiv*, abs/2206.08853, 2022. 1, 8, 12

[14] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. M. Veloso, and R. Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. In *International Joint Conference on Artificial Intelligence*, 2019. 12

[15] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell. The malmo platform for artificial intelligence experimentation. In *International Joint Conference on Artificial Intelligence*, 2016. 1, 9, 12

[16] A. Khandelwal, L. Weihs, R. Mottaghi, and A. Kembhavi. Simple but effective: Clip embeddings for embodied ai. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14809–14818, 2021. 8

[17] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013. 3

[18] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. B. Girshick. Segment anything. *ArXiv*, abs/2304.02643, 2023. 8

[19] S. Lifshitz, K. Paster, H. Chan, J. Ba, and S. A. McIlraith. Steve-1: A generative model for text-to-behavior in minecraft. *ArXiv*, abs/2306.00937, 2023. 1, 3, 8, 12, 14

[20] A. Majumdar, G. Aggarwal, B. Devnani, J. Hoffman, and D. Batra. Zson: Zero-shot object-goal navigation using multimodal goal embeddings. *ArXiv*, abs/2206.12403, 2022. 8

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. 4

[22] 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 *International Conference on Machine Learning*, 2021. 8

[23] V. Sanh, A. Webson, C. Raffel, S. H. Bach, L. Sutawika, Z. Alyafeai, A. Chaffin, A. Stiegler, T. L. Scao, A. Raja, M. Dey, M. S. Bari, C. Xu, U. Thakker, S. S. Sharma, E. Szczechla, T. Kim, G. Chhablani, N. Nayak, D. Datta, J. Chang, M. T.-J. Jiang, H. Wang, M. Manica, S. Shen, Z. X. Yong, H. Pandey, R. Bawden, T. Wang, T. Neeraj, J. Rozen, A. Sharma, A. Santilli, T. Fevry, J. A. Fries, R. Teehan, S. Biderman, L. Gao, T. Bers, T. Wolf, and A. M. Rush. Multitask prompted training enables zero-shot task generalization, 2021. 8

[24] J. Schmidhuber. Reinforcement learning upside down: Don't predict rewards - just map them to actions. *ArXiv*, abs/1912.02875, 2019. 8

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. 17

[26] R. K. Srivastava, P. Shyam, F. W. Mutz, W. Jaśkowski, and J. Schmidhuber. Training agents using upside-down reinforcement learning. *ArXiv*, abs/1912.02877, 2019. 8

[27] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946, 2019. 9, 13

[28] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov. Label Studio: Data labeling software, 2020-2022. Open source software available from https://github.com/heartexlabs/label-studio. 17

[29] L. van der Maaten and G. E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008. 9

[30] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. J. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *ArXiv*, abs/2305.16291, 2023. 8, 12

[31] Z. Wang, S. Cai, A. Liu, X. Ma, and Y. Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *ArXiv*, abs/2302.01560, 2023. 8, 10, 12

[32] Z. Xie, Z. Lin, D. Ye, Q. Fu, W. Yang, and S. Li. Future-conditioned unsupervised pretraining for decision transformer. In *International Conference on Machine Learning*, 2023. 8

[33] T. Yu, D. Quillen, Z. He, R. C. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. *ArXiv*, abs/1910.10897, 2019. 1

[34] K. Zakka, A. Zeng, P. R. Florence, J. Tompson, J. Bohg, and D. Dwibedi. Xirl: Cross-embodiment inverse reinforcement learning. In *Conference on Robot Learning*, 2021. 8

[35] Q. Zhang, Z. Peng, and B. Zhou. Learning to drive by watching youtube videos: Action-conditioned contrastive policy pretraining. In *European Conference on Computer Vision*, 2022. 8

[36] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. *ArXiv*, abs/1909.08593, 2019. 8

Appendix

A Limitation and Conclusion

Although GROOT has demonstrated powerful capabilities in expressing open-ended tasks in the form of video instructions, training such a goal space remains highly challenging. We found that GROOT is quite sensitive to the selection of reference videos, which we attribute to the fact that the goal space trained from an unsupervised perspective may not be fully aligned with the human intention for understanding the semantics of the reference video. Therefore, it would be a promising research direction in the future to use RLHF [36] and SFT (supervised fine-tuning, [23]) to align the pre-trained goal space with human preference.

In conclusion, we propose a paradigm for learning to follow instructions by watching gameplay videos. We argue that video instruction is a good form of goal space that not only expresses open-ended tasks but can also be trained through self-supervision. Based on this, we built an encoder-decoder transformer architecture agent named GROOT in Minecraft. Without relying on any annotated data, GROOT demonstrated extraordinary instruction-following ability and crowned the Minecraft SkillForge benchmark. Additionally, we also showed its potential as a planner downstream controller in the challenging `obtain diamond` task. We believe that this training paradigm has strong generalization and hope to see its application to more complex open-world environments.

B Related Works

Pre-train Policy on Offline Data. Pre-training neural networks on internet-scale data has been demonstrated as a very effective training paradigm in Computer Vision [22, 18] and Nature Language Processing [11, 5]. Recently, researchers tried to transfer this success to the field of decision-making from pre-training visual representations and directly distilling the policy from offline data. As the former, [2, 34, 6] leveraged temporal information present in videos as the supervision signal to learn visual representations. The representations are used to generate intrinsic rewards for boosting downstream policy training, which still requires expensive online interactions with the environment. [24, 26, 9] tried to leverage scalable offline trajectories to train optimal policy by conditioning it on cumulated rewards. However, these methods require clear task definitions and reward labels. This makes it hard to be applied to open worlds, where the tasks are infinite and the reward is open-ended. More generally, [8, 35, 3] used a pre-trained inverse dynamic model to label actions for gameplay videos and directly distilled the policy with imitation learning.

Condition Policy on Goal Space. Researchers have explored many goal modalities, such as language [16], image [12], and future video [32], to build a controllable policy. [4] collected a large-scale dataset of trajectory-text pairs and trained a transformer policy to follow language instructions. Despite the language being a natural instruction interface, the cost of collecting paired training data is expensive. As a solution, [20] sorted to use hindsight relabeling to first train a policy conditioned on the target image, then aligned text to latent image space, which greatly improves training efficiency. [19] moved a big step on this paradigm by replacing the target image with a 16-frame future video and reformulating the modality alignment problem into training a prior of latent goal given text.

Build Agents in Minecraft. As a challenging open-world environment, Minecraft is attracting increasing researchers to develop AI agents on it, which can be divided into plan-oriented [31, 30] and control-oriented methods [3, 7, 19] based on their emphasis. Plan-oriented agents aim to reason with Minecraft knowledge and decompose the long-horizon task into sub-tasks followed by calling a low-level controller. Control-oriented works follow the given instructions and directly interact with the environments using low-level actions (mouse and keyboard). [3] pre-trained the first foundation model VPT in Minecraft using internet-scale videos. Although it achieves the first obtaining diamond milestone by fine-tuning with RL, it does not support instruction input. [19] created the first agent that can solve open-ended tasks by bridging VPT and MineCLIP [13]. However, its goal space is not expressive enough and prevents it from solving multi-step tasks.

C Goal Space Discovery via Future State Prediction

This section explains our learning framework: discovering a "good" goal space as well as a video instruction following controller through the task of predicting future states given previous ones. We start with an illustrative example in Minecraft [15]. Imagine that an agent is standing inside a grassland holding an axe that can be used to chop the tree in front of them. Suppose in the gameplay video, players either go straight to chop the tree or bypass it to explore the territory. In order to predict future frames, it is sufficient to know (i) which goal (chop tree or bypass tree) is being pursued by the agent, and (ii) what will happen if the agent chooses a particular option (i.e., transition dynamics). Apart from the latter information that is irrelevant to the past observations, we only need to capture the goal information, i.e., whether the agent decides to chop the tree or bypass the tree. Therefore, the task of establishing a comprehensive while succinct goal space can be interpreted as predicting future states while conditioning on the transition dynamics of the environment.

Formally, our learning objective is to maximize the log-likelihood of future states given past ones: $\log p_\theta(s_{t+1:T}|s_{0:t})$. Define g as a latent variable conditioned on past states (think of it as the potential goals the agent is pursuing given past states), the evidence lower-bound of the objective given variational posterior $q_\phi(g|s_{0:T})$ is the following (see Appendix E for the derivation of this and the following equations):

$$\begin{aligned}\log p_\theta(s_{t+1:T}|s_{0:t}) &= \log \sum_g p_\theta(s_{t+1:T}, g|s_{0:t}) \\ &\geq \mathbb{E}_{g\sim q_\phi(\cdot|s_{0:T})}\left[\log p_\theta(s_{t+1:T}|s_{0:t}, g)\right] - D_{\mathrm{KL}}\left(q_\phi(g|s_{0:T}) \parallel p_\theta(g|s_{0:t})\right),\end{aligned}$$

where $D_{\mathrm{KL}}(\cdot\|\cdot)$ denotes the KL-divergence. Next, we break down the first term (i.e., $p(s_{t+1:T}|s_{0:t}, g)$) into components contributed by the (unknown) goal-conditioned policy $\pi(a|s, g)$ and the transition dynamics $p(s_{t+1}|s_{0:t}, a_t)$:

$$\begin{aligned}\log p_\theta(s_{t+1:T}|s_{0:t}, g) &= \sum_{\tau=t}^{T} \log \sum_{a_\tau} \pi_\theta(a_\tau|s_{0:\tau}, g)\cdot p_\theta(s_{\tau+1}|s_{0:\tau}, a_\tau) \\ &\geq \sum_{\tau=t}^{T} \mathbb{E}_{a_\tau\sim p_\theta(a_\tau|s_{0:\tau+1})}\big[\log \pi_\theta(a_\tau|s_{0:\tau}, g) + C\big],\end{aligned}$$

where the constant C contains terms that depend solely on the environment dynamics and are irrelevant to what we want to learn (i.e., the goal space and the goal-conditioned policy). Bring it back to the original objective, we have

$$\log p(s_{t+1:T}|s_{0:t}) \geq \underbrace{\sum_{\tau=t}^{T-1} \mathbb{E}_{g\sim q_\phi(\cdot|s_{0:T}), a_\tau\sim p_\theta(\cdot|s_{0:\tau+1})}\left[\log \pi_\theta(a_\tau|s_{0:\tau}, g)\right]}_{\text{behaviour cloning}} - \underbrace{D_{\mathrm{KL}}\left(q_\phi(g|s_{0:T}) \parallel p_\psi(g|s_{0:t})\right)}_{\text{goal space constraint (KL regularization)}},$$

where $q_\phi(\cdot|s_{0:T})$ is implemented as a video encoder that maps the whole state sequence into the latent goal space. $p(\cdot|s_{0:\tau+1})$ is the inverse dynamic model (IDM) that predicts actions required to achieve a desired change in the states, which is usually a pre-trained model. Thus, the objective can be explained as jointly learning a video encoder and a goal-controller policy through behavior cloning under succinct goal space constraints.

D Additional Results

D.1 Properties of Learned Goal Space

This section studies the properties of learned goal space. We used the t-SNE algorithm [29] to visualize the clustering effect of reference videos encoded in goal space, as in Figure 3. We select 7 kinds of videos, including `craft items`, `combat enemies`, `harvest crops`, `hunt animals`, `chop trees`, `trade with villagers`, and `mine ores`. These videos are sampled from the contractor data [3] according to the meta information (details are in Appendix I). Each category contains 1k video segments. As a control group, in Figure 3 (left), we showed the initial goal space of the video encoder (with a pre-trained EfficientNet-B0 [27] as the backbone) before training. We

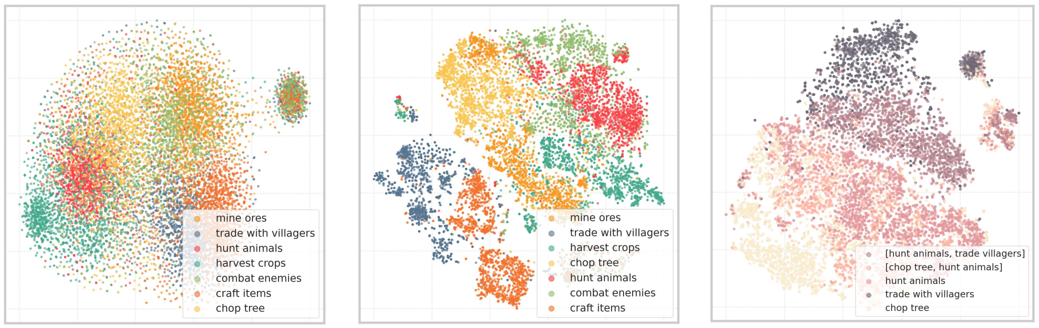


Figure 3: **t-SNE visualization of the goal space.** Each color corresponds to a specific video category. **Left:** Space of randomly initialized video encoder. All the videos are entangled together. **Middle:** Space of GROOT trained with self-supervised learning. The videos are clustered based on their semantics. **Right:** Synthesized videos using concatenation manner. The concatenated videos lay on the position between the source videos.

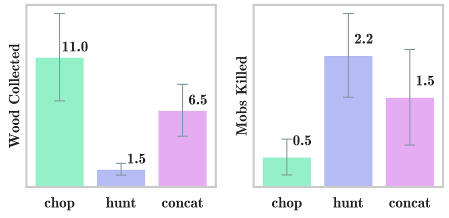


Figure 4: **Comparision on using raw and concatenated reference videos as conditions. Left:** Collected wood in the forest biome. **Right:** Killed mobs in the plains biome. "concat" denotes the reference video is `[chop trees, hunt animals]`.

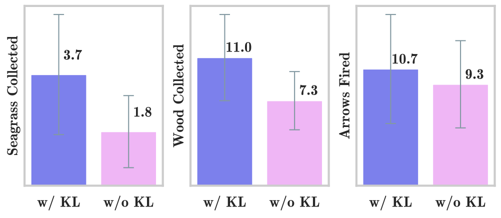


Figure 5: **Ablation study on KL loss.** After being jointly trained with KL loss, GROOT can collect 2× more seagrass () underwater and 1.5× wood () in the forest while the difference is not as impressive on the `use bow` () task.

found that the points are entangled together. After being trained on offline trajectories, as in Figure 3 (middle), it well understands reference videos and clusters them according to their semantics. This proves that it is efficient to learn behavior-relevant task representations using our self-supervised training strategy. Inevitably, there are still some videos from different categories entangled together. We attribute this to the possibility of overlap in the performed behaviors of these videos. For example, `chop trees` and `harvest crops` both rely on a sequential of "attack" actions.

Condition on Concatenated Videos. We also study the possibility of conditioning the policy on concatenated videos. First, we collect 3 kinds of source videos, including `chop trees`, `hunt animals`, and `trade with villagers`. We randomly sampled two videos from sources of `chop trees` and `hunt animals`, downsampled and concatenated them into a synthetic video, denoted as `[chop trees, hunt animals]`. By the same token, we can obtain `[hunt animals, trade with villagers]`. We visualize these videos together with the source videos in Figure 3 (right). We found that the source videos lie far away from each other while the concatenated videos are distributed between their source videos. Based on this intriguing phenomenon, we infer that concatenated videos may prompt GROOT to solve both tasks simultaneously. To verify this, we evaluate GROOT on three kinds of reference videos, i.e., `chop trees`, `hunt animals`, and `[chop trees, hunt animals]`. We launched GROOT in the forest and in the animal plains, respectively. The collected wood and killed mobs are reported in Figure 4. We found that although the concatenated video may not be as effective as raw video in driving an agent to complete a single task (60% of the performance of raw video), it does possess the ability to drive the agent to perform multiple tasks. This is an important ability. As discussed in [31], sometimes the high-level planner will propose multiple candidate goals, it will be efficient if the low-level controller can automatically determine which to accomplish based on the current observation.

Ablation on KL Divergence Loss. To investigate the role of KL loss in training, we evaluated GROOT (w/ KL) and its variant (w/o KL) on three tasks: `collect seagrass` (), `collect wood` (), and `use bow` (). As shown in Figure 5, we found that introducing the constraint of KL loss improved agent performance by 2× and 1.5× in the first two tasks, whereas there was no significant effect in the `use bow` task. This may be because the first two tasks require the agent to generalize the corresponding skills to different terrains (e.g. locating trees in the environment for

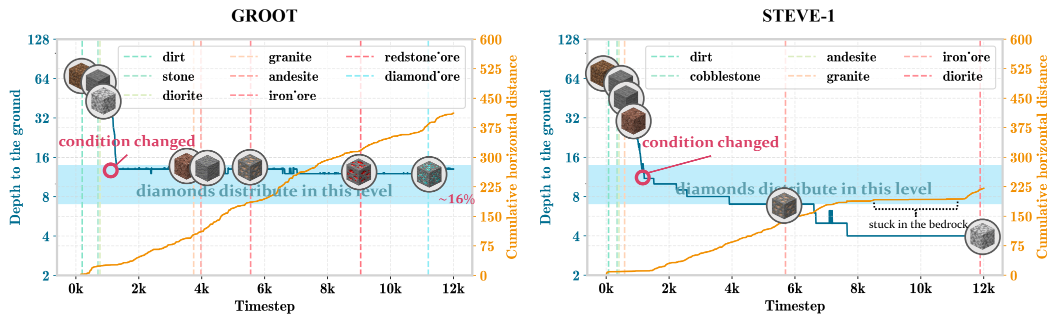


Figure 6: **Results on solving challenging `obtain diamond` task.** The vertical dashed lines represent the time when a certain item is first obtained. **Left:** GROOT first dug down to the depth of 12 and then mined horizontally to obtain diamonds with an average success rate of 16%. **Right:** STEVE-1 quickly dug down to the specific depth but struggled to maintain its height.

collecting wood and sinking to specific locations for collecting seagrass). Therefore, it puts higher demands on the agent's ability to generalize in the goal space, and this is exactly the role played by the KL loss. The `use bow` task is relatively simple in comparison because it only requires charging and shooting the arrow, without considering environment factors.

D.2 Combining Skills for Long-horizon Tasks

In this section, we explore whether GROOT can combine skills to solve long-horizon tasks, which is key to its integration with a high-level planner. Taking the task of mining diamonds as an example, prior knowledge is that diamond ores are generally distributed between the 7th and 14th floors underground, and the probability of appearing in other depths is almost zero. Therefore, the agent needs to first dig down to the specified depth (12) and then maintain horizontal mining. To achieve this, we designed two reference videos, each 128 frames long. One describes the policy of starting from the surface and digging down, and the other demonstrates the behaviors of horizontal mining. We show an example in Figure 6 (left). In the beginning, GROOT quickly digs down to the specified depth and then switches to horizontal mining mode. It maintains the same height for a long time and found diamonds at 11k steps. In addition, we compared STEVE-1 (visual) under the same setting in Figure 6 (right). After switching to the horizontal mining prompt, STEVE-1 maintains its height for a short time before it stuck in the bedrock layer (unbreakable in survival mode), greatly reducing the probability of finding diamonds. This indicates that our goal space is expressive enough to instruct the way of mining, and the policy can follow the instructions persistently and reliably. In contrast, STEVE-1, which relies on future outcomes as a condition, was unable to maintain its depth, despite attempts at various visual prompts. We conducted 25 experiments each on GROOT and STEVE-1, with success rates of 16% and 0% for finding diamonds. Additional details are in the Appendix K.

E Derivation

In this section, we detail how we derive the final objective. Recall that the goal is to maximize the log-likelihood of future states given past ones: $\log p(s_{t+1:T}|s_{0:t})$. Using Bayes' theorem and the Jensen's inequality, we have:

$$\begin{aligned}
\log\ p(s_{t+1:T}|s_{0:t}) &= \log \sum_z p(s_{t+1:T}, z|s_{0:t}), && (4)\\
&= \log \sum_z \frac{p(s_{t+1:T}, z|s_{0:t})\ q(z|s_{0:T})}{q(z|s_{0:T})}, && (5)\\
&\geq \mathbb{E}_{z\sim q(z|s_{0:T})}\big[\log\ p(s_{t+1:T}, z|s_{0:t}) - \log\ q(z|s_{0:T})\big], && (6)\\
&= \mathbb{E}_{z\sim q(z|s_{0:T})}\big[\log\ p(s_{t+1:T}|s_{0:t}, z) + \log\ p(z|s_{0:t}) - \log\ q(z|s_{0:T})\big], && (7)\\
&= \mathbb{E}_{z\sim q(z|s_{0:T})}\big[\log\ p(s_{t+1:T}|s_{0:t}, z)\big] + \mathbb{E}_{z\sim q(z|s_{0:T})}\Big[\log \frac{p(z|s_{0:t})}{q(z|s_{0:T})}\Big], && (8)\\
&= \mathbb{E}_{z\sim q(z|s_{0:T})}\big[\log\ p(s_{t+1:T}|s_{0:t}, z)\big] - D_{\text{KL}}\big(q(z|s_{0:T})\ \|\ p(z|s_{0:t})\big). && (9)
\end{aligned}$$

We break down $p(s_{t+1:T}|s_{0:t}, z)$ into components: goal-conditioned policy $\pi(a_\tau|s_{0:\tau+1})$ and the transition dynamics $p(s_{t+1}|s_{0:t}, a_t)$, we have

$$p(s_{t+1:T}|s_{0:t}, z) = \prod_{\tau=t}^{T-1} \big(\sum_{a_\tau} \pi(a_\tau|s_{0:\tau}, z) \cdot p(s_{\tau+1}|s_{0:\tau}, a_\tau) \big). \tag{10}$$

Furthermore, using Jensen's inequality, $\log p(s_{t+1:T}|s_{0:t}, z)$ can be written as

$$\begin{aligned} \log\ p(s_{t+1:T}|s_{0:t}, z) &= \sum_{\tau=t}^{T-1} \log \sum_{a_\tau} \pi(a_\tau|s_{0:\tau}, z) \cdot p(s_{\tau+1}|s_{0:\tau}, a_\tau), & (11)\\ &= \sum_{\tau=t}^{T-1} \log \sum_{a_\tau} \pi(a_\tau|s_{0:\tau}, z) \cdot \frac{p(a_\tau|s_{0:\tau}, s_{\tau+1}) \cdot p(s_{\tau+1}|s_{0:\tau})}{p(a_\tau|s_{0:\tau})}, & (12)\\ &\geq \sum_{\tau=t}^{T-1} \mathbb{E}_{a_\tau \sim p(a_\tau|s_{0:\tau}, s_{\tau+1})} \big[\log \pi(a_\tau|s_{0:\tau}, z) + C\big], & (13) \end{aligned}$$

where the constant $C = \log p(s_{\tau+1}|s_{0:\tau}) - \log p(a_\tau|s_{0:\tau})$ depends solely on the environment dynamics and are irrelevant to what we want to learn (i.e., the goal space and the goal-conditioned policy), we have:

$$\begin{aligned} \mathbb{E}_{z\sim q(z|s_{0:T})} \big[\log p(s_{t+1:T}|s_{0:t}, z)\big] &\geq \mathbb{E}_{z\sim q(z|s_{0:T})} \Big[\sum_{\tau=t}^{T-1} \mathbb{E}_{a_\tau \sim p(a_\tau|s_{0:\tau}, s_{\tau+1})} \big[\log \pi(a_\tau|s_{0:\tau}, z)\big]\Big], & (14)\\ &= \sum_{\tau=t}^{T-1} \mathbb{E}_{z\sim q(z|s_{0:T}), a_\tau \sim p(a_\tau|s_{0:\tau}, s_{\tau+1})} \big[\log \pi(a_\tau|s_{0:\tau}, z)\big]. & (15) \end{aligned}$$

Thus, we derived the evidence lower-bound of $\log p(s_{t+1:T}|s_{0:t})$ as follows

$$\log p(s_{t+1:T}|s_{0:t}) \geq \sum_{\tau=t}^{T-1} \mathbb{E}_{z\sim q(z|s_{0:T}), a_\tau \sim p(a_\tau|s_{0:\tau+1})} \big[\log \pi(a_\tau|s_{0:\tau}, z)\big] - D_{\text{KL}}\big(q(z|s_{0:T}) \parallel p(z|s_{0:t})\big). \tag{16}$$

F Minecraft Environment

Minecraft is an extremely popular sandbox game that allows players to freely create and explore their world. This game has infinite freedom, allowing players to change the world and ecosystems through building, mining, planting, combating, and other methods (shown in Figure 7). It is precisely because of this freedom that Minecraft becomes an excellent AI testing benchmark [15, 3, 13, 7, 19, 31, 30]. In this game, AI agents need to face situations that are highly similar to the real world, making judgments and decisions to deal with various environments and problems. Therefore, Minecraft is a very suitable environment to be used as AI testing benchmark. By using Minecraft, AI researchers can more conveniently simulate various complex and diverse environments and tasks, thereby improving the practical value and application of AI technology.

We use the combination of 1.16.5 version MineRL [14] and MCP-Reborn[2] as our testing platform, which is consistent with the environment used by VPT [3] and STEVE-1 [19]. Mainly because this platform preserves observation and action space that is consistent with human players to the fullest extent. On the one hand, this design brings about high challenges, as agents can only interact with the environment using low-level mouse and keyboard actions, and can only observe visual information like human players without any in-game privileged information. Therefore, the AI algorithms developed on this platform can have higher generalization ability. On the other hand, this also presents opportunities for us to conduct large-scale pre-training on internet-scale gameplay videos.

[2]https://github.com/Hexeption/MCP-Reborn

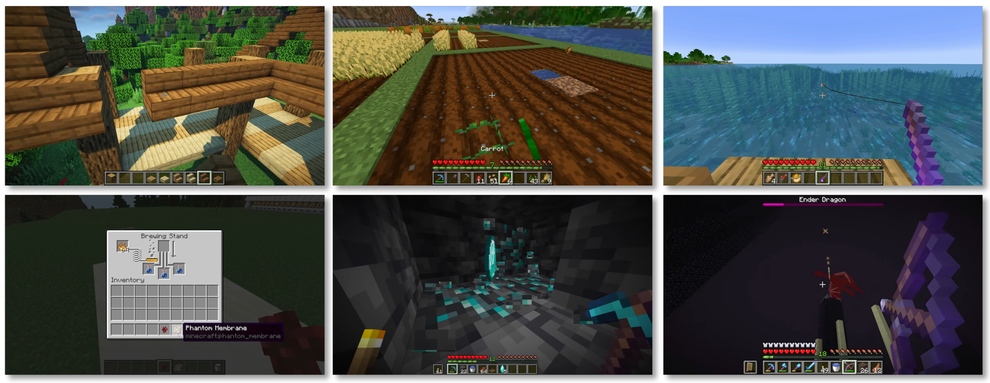


Figure 7: Examples of Minecraft environment. Tasks from top to bottom, from left to right are building houses, planting wheat, fishing, brewing a potion, mining diamond ores, and combating the ender dragon, respectively.

F.1 Observation Space

The visual elements included in our observation space are completely consistent with those seen by human players, including the Hotbar, health indicators, player hands, and equipped items. The player's perspective is in the first person with a field of view of 70 degrees. The simulator first generates an RGB image with dimensions of 640×360 during the rendering process. Before inputting to the agent, we resize the image to 224×224 to enable the agent to clearly see item icons in the inventory and important details in the environment. When the agent opens the GUI, the simulator also renders the mouse cursor normally. The RGB image is the only observation that the agent can obtain from the environment during inference. It is worth noting that to help the agent see more clearly in extremely dark environments, we have added a night vision effect for the agent, which increases the brightness of the environment during nighttime.

F.2 Action Space

Our action space is almost identical to that of humans, except for actions that involve inputting strings. It consists of two parts: the mouse and the keyboard. The mouse movement is responsible for changing the player's camera perspective and moving the cursor when the GUI is opened. The left and right buttons are responsible for attacking and using items. The keyboard is mainly responsible for controlling the agent's movement. We list the meaning of each action in the Table 1. To avoid predicting null action, we used the same joint hierarchical action space as [3], which consists of button space and camera space. Button space encodes all combinations of keyboard operations and a flag indicating whether the mouse is used, resulting in a total of 8461 candidate actions. The camera space discretizes the range of one mouse movement into 121 actions. Therefore, the action head of the agent is a multi-classification network with 8461 dimensions and a multi-classification network with 121 dimensions.

G Implementation Details

G.1 Model Architecture

The video encoder consists of a convolutional neural network backbone and a non-causal transformer. Inspired by [4], we adopted the EfficientNet [27] as the backbone. Specifically, we use its variant EfficientNet-B0 for efficiency, which takes in images of size 224×224 and extracts a feature vector of shape $7 \times 7 \times 1280$, where 7×7 denotes the spatial dimensions. In order to adaptively enhance the important visual information, we use a shallow transformer to pool the feature map along spatial channels. To fuse global visual features, we construct another learnable embedding `[sp]`, concatenate it with the 49 features in space, and obtain a token sequence of length 50. After being processed by the transformer, the output for the `[sp]` token corresponds to the pooled visual feature, whose dimension is $d_{hid} = 1024$. To capture the temporal features of the video, we remove

Table 1: Action space descriptions from 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	Jump	key Space	Jump. When swimming, keeps the player afloat.
6	Sneak	key left Shift	Slowly move in the current direction of movement. When used in conjunction with the attack function in the GUI, it can swap items between inventory and Hotbar. When used with the craft function, it crafts the maximum possible number of items instead of just one.
7	Sprint	key left Ctrl	Move quickly in the direction of current motion.
8	Attack	left Button	Destroy blocks (hold down); Attack entity (click once).
9	Use	right Button	Put down the item being held or interact with the block that the player is currently looking at. Within the GUI, pick up a stack of items or place a single item from the stack that is being held by the mouse.
10	hotbar.[1-9]	keys 1 - 9	Selects the appropriate hotbar item. When in the inventory GUI, swap the contents of the inventory slot under the mouse pointer and the corresponding hotbar slot.
11	Yaw	move Mouse X	Turning; aiming; camera movement.Ranging from -180 to +180.
12	Pitch	move Mouse Y	Turning; aiming; camera movement.Ranging from -180 to +180.

the code related to the casual mask in the minGPT[3] and obtain a non-causal transformer. The policy decoder consists of 4 identical blocks, where each block contains a Flamingo *gated-attention dense layer* [1] and a Transformer-XL block[10]. The Transformer-XL block maintains a recurrence memory of past 128 key-value pairs to memory long-horizon history states. We directly use the Transformer-XL implementation in [3] with a simple modification, i.e., before passing states into the policy decoder, we add the previous action to the state embedding at each timestep. Notably, We find this modification **very useful** especially when we need to train the policy from scratch. As it not only accelerates the training process but makes the predicted action more **consistent** and **smooth**. Additional hyperparameters can be found in Table 2.

G.2 Inference

To generate reference videos, we invited three human players to play each task according to the task description. Each person was asked to produce two videos, so we could prepare six videos for each task in total. Then, we selected the most relevant video to the task description from the six videos and cropped the first 128 frames into a new video, which was used to instruct GROOT to complete this task. In addition, we selected a 16-frame segment that best expressed the task information as the visual prompt for STEVE-1 (visual) from these six videos. This ensures fairness in comparison.

During inference, we found that in some tasks, such as `build obsidian` (⬢), GROOT's behavior mixed with the intention of traveling around. We believe this is a bias introduced during training. We draw the inspiration from STEVE-1 [19] and subtract this bias in the action logits space before sampling the action. Specifically, we infer two models at the same time, where one model's condition is a specific task video and the other model's condition is a 128-frame video of traveling freely in the environment. The input observations for the two models are exactly the same. At each time step, we use the action logits of the previous model to subtract a certain proportion of the action logits predicted by the latter model before using the Gumbel-Softmax trick to sample

[3]https://github.com/karpathy/minGPT

Table 2: Hyperparameters for training GROOT.

Hyperparameter	Value
Optimizer	AdamW
Weight Decay	0.001
Learning Rate	0.0000181
Warmup Steps	2000
Number of Workers	4
Parallel Strategy	ddp
Type of GPUs	NVIDIA RTX 4090Ti, A40
Parallel GPUs	8
Accumulate Gradient Batches	8
Batch Size/GPU (Total)	2 (128)
Training Precision	bf16
Input Image Size	224×224
CNN Backbone	EfficientNet-B0
Encoder Transformer	minGPT (w/o causal mask)
Decoder Transformer	TransformerXL
Number of Encoder Blocks	8
Number of Decoder Blocks	4
Hidden Dimension	1024
Number of Condition Slots	1
Trajectory Chunk size	128
Attention Memory Size	256
Weight of BC Loss	1
Weight of KL Loss	0.01

the action. We found that this technique can significantly improve the success rate of tasks such as `build obsidian` () and `enchant sword` ().

H Dataset Details

H.1 Contractor Data

The contractor data is a Minecraft offline trajectory dataset provided by [3] [4], which is annotated by professional human players and used for training the inverse dynamic model. In this dataset, human players play the game while the system records the image sequence $\{s_{1:T}\}_M$, action sequence $\{a_{1:T}\}_M$, and metadata $\{e_{1:T}\}_M$ generated by the players. Excluding frames containing empty actions, the dataset contains 1600M frames with a duration of approximately 2000 hours. The metadata records the events triggered by the agent in the game at each time step, including three types: `craft item`, `pickup`, and `mine block`, which represent the agent's activities of crafting items using the GUI, picking up dropped items and destroying blocks at the current time step, respectively. In the process of training the model, we use all trajectories provided by the contractor data, but without including any metadata. We only use the metadata to retrieve relevant trajectory segments during the visualization of the goal space.

I t-SNE Visualization Details

This section details how the videos are sampled to do visualization. The selected videos are categorized into seven groups: `craft items`, `combat enemies`, `harvest crops`, `hunt animals`, `chop trees`, `trade with villagers`, and `mine ores`. Generally, each group contains two types of videos, each with 1000 data points sampled. The sampling method retrieves the time when a certain event occurs in the metadata and goes back 128 frames from that time to obtain a video segment that is 128 frames long. We illustrate video configurations in Table 3. For example, in the `combat enemies` task, taking "combat zombies" as an example, we retrieve all

[4]https://github.com/openai/Video-Pre-Training

the moments when the event "pickup:rotten_flesh" occurs, because after killing zombies, they will drop rotten flesh, which can then be picked up by players. Through sampling observations, we found that this method can sample videos that are consistent with the descriptions.

Table 3: Sample videos from the contractor data [3] for the goal space visualization.

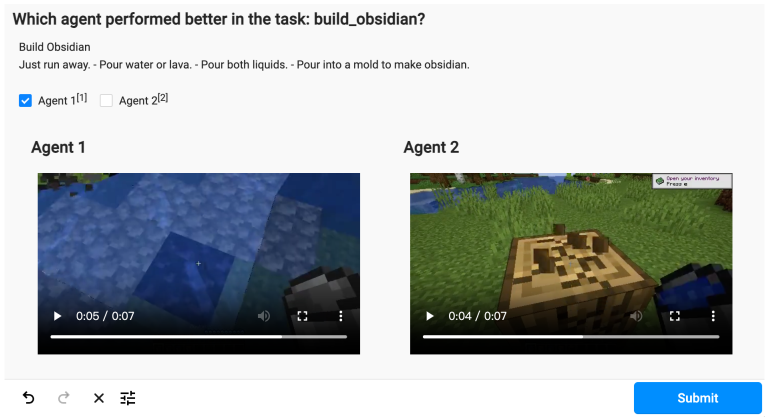
Group	**Video Description**	**Event in Metadata**
craft items	craft wodden_pickaxe with crafting_table	craft_item:wooden_pickaxe
craft items	craft iron_pickaxe with crafting_table	craft_item:iron_pickaxe
combat enemies	combat zombies	pickup:rotten_flesh
combat enemies	combat spiders	pickup:spider_eye
harvest crops	harvest wheat	mine_block:wheat
harvest crops	harvest melon	mine_block:melon
hunt animals	hunt sheep	pickup:mutton
hunt animals	hunt cow	pickup:beef
chop trees	chop oak trees	mine_block:oak_log
chop trees	chop birch trees	mine_block:birch_log
trade with villagers	trade with villagers for emerald	craft_item:emerald
trade with villagers	trade with villagers for enchanted_book	craft_item:enchanted_book
mine ores	mine coal ores with pickaxe	mine_block:coal_ore
mine ores	mine iron ores with pickaxe	mine_block:iron_ore

J Programmatic Evaluation Details

In this section, we elaborated on how each episode is regarded as successful. For the `dye and shear sheep` () task, dyeing the sheep and shearing its wool must be successfully performed to be considered a success. For the `use bow` () task, firing the arrow after charging it to the maximum degree is required to be successful. For the `sleep` () task, placing the bed and spending the night on it are required to be successful. For the `smelt` () task, placing the furnace and dragging coal and mutton into the designated slots are required to be successful. For the `lead` () task, successfully tethering at least one animal is considered a success. For the `build obsidian` () task, pouring a water bucket and a lava bucket to fuse them is required to be successful. For the `enchant` () task, placing the enchantment table, putting a diamond sword and lapis lazuli into the slots, and clicking the enchanting option are required to be successful. For the `dig down three fill one up` () task, the agent must first vertically break three dirt blocks below and then use one dirt block to seal the area above. For the `build snow golems` () task, placing 2 snow blocks and 1 carved pumpkin head in order and triggering the creation of a snow golem are required to be successful.

K Combining Skills Experimental Details

First, we introduce the experimental environment selected for our study. The agent is summoned on the plains biome, holding a diamond pickaxe, and granted the night vision status to enable the agent to see the various ores underground. At the beginning of each episode, we set the agent's condition to `dig down`. When the agent descends to a depth below 12 layers, the condition automatically switches to `horizontal mining`. Each round of episodes lasts for 12,000 frames, which is equivalent to 10 minutes in the real world. For GROOT, both the reference videos of `dig down` and `horizontal mining` were recorded by a human player. For STEVE-1, we invited the same player to carefully record the prompt videos. It is worth noting that while we could easily prompt it to dig down, it was difficult to keep it in the horizontal mining condition. This made STEVE-1 prone to falling into the bedrock layer and getting stuck. Finally, we did not observe STEVE-1 finding any diamonds in the 25 experiments, which can be attributed to the inability of its goal space to encode details such as horizontal mining.


Which agent performed better in the task: build_obsidian?

Build Obsidian

Just run away. - Pour water or lava. - Pour both liquids. - Pour into a mold to make obsidian.

[✓ Agent 1[1]] Agent 2[2]

Figure 8: Example of the annotating system for human evaluation.

L Elo Rating System Details

The Elo rating system is widely adopted for evaluating the skill levels of multiple players in two-player games, such as Chess and Go [25]. In this section, we elaborate on how we introduce human evaluation and use the Elo Rating system to measure the relative performance of agents on the Minecraft SkillForge benchmark.

In the Elo rating system, each agent's skill level is represented by a numerical rating. We repeatedly let agents play against each other in pairs. Specifically, in each game, we sample a task and two agents, denoted as Agent A and Agent B. Then, we randomly sample a trajectory for each agent corresponding to the designated task. The two trajectories are assigned to a human annotator, who selects the most task-relevant one. We implement the annotating system with Label Studio [28], as shown in Figure 8. We consider the agent that produced this trajectory to be the winner, let's assume it is Agent A. After each round, we update the scores of Agent A and Agent B as follows

$$\begin{aligned} R_A &\leftarrow R_A + K \cdot \frac{1}{1 + 10^{(R_A - R_B)/400}}, \\ R_B &\leftarrow R_B - K \cdot \frac{1}{1 + 10^{(R_A - R_B)/400}}, \end{aligned} \tag{17}$$

where K is the update factor and we set it to 8. After calculating the score of the agent, we use VPT (bc) as 1500 points and shift the scores of other agents accordingly. Based on the Elo ratings, we can easily measure the relative winning rate for each paired agent. The win rate of Agent A over Agent B can be represented as $\frac{1}{1+10^{(R_B-R_A)/400}}$. For example, the win rate ratio between two agents with a score difference of 100 scores is $64\% : 36\%$. A score difference of 200 scores implicit $76\% : 24\%$.

M Minecraft SkillForge Benchmark Details

In this section, we detail the benchmark titled "Minecraft SkillForge" which meticulously incorporates a wide spectrum of tasks prevalent within Minecraft. Our aim is to ensure that every task provides a meaningful evaluation of a specific skill that an AI agent might possess. We categorize these tasks into six groups: `collect`, `explore`, `craft`, `tool`, `survive`, and `build`. In the following subsections, we will provide a detailed introduction to each of them. The "Description" field provides a brief description of the task, the "Precondition" field outlines the initial settings of the testing environment for the task, the "SkillAssessed" field indicates which aspect(s) of the agent's ability are being assessed by the task, and the "Evaluation" field describes the quality evaluation metrics for task completion (based on which human players judge the quality of two rollout videos).

M.1 Collect

The tasks categorized under the `collect` section of our benchmark are specifically designed to evaluate an AI agent's capability in resource acquisition proficiency and spatial awareness. This means the agent should not only be adept at identifying and gathering specific resources but also possess the acumen to navigate through varied environments while being aware of its surroundings and the available tools at its disposal.

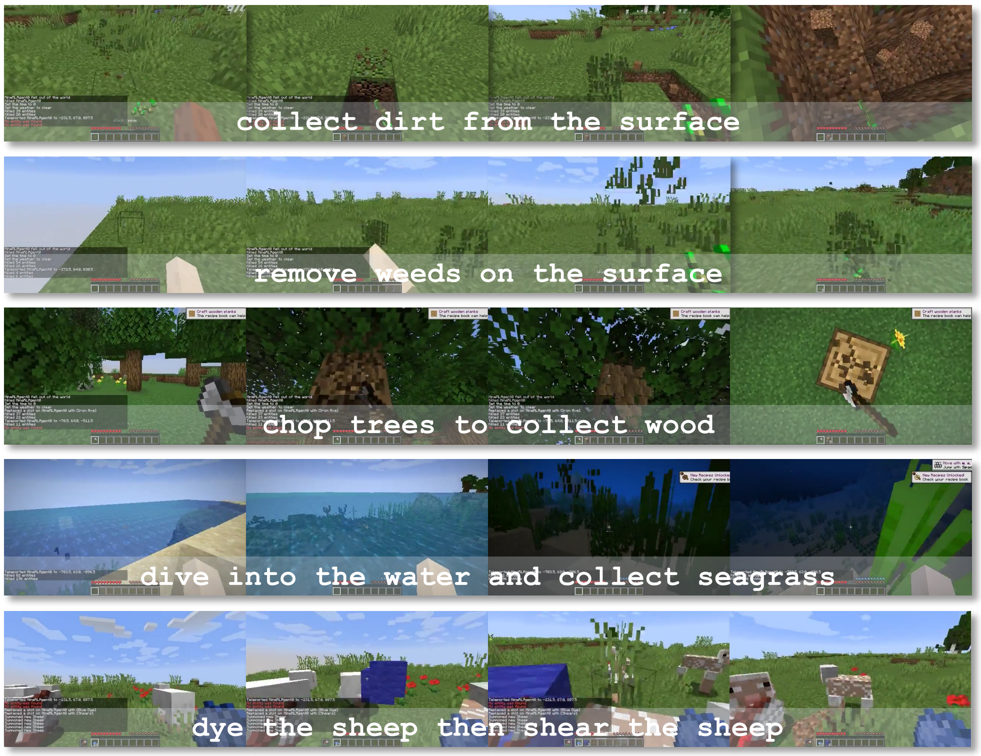

collect dirt from the surface
remove weeds on the surface
chop trees to collect wood
dive into the water and collect seagrass
dye the sheep then shear the sheep

Figure 9: Examples of tasks in `collect` category.

```
**Task**: collect dirt
**Description**: Collect dirt from the surface.
**Precondition**: Spawn the player in the plains biome.
**SkillAssessed**: Basic terrain understanding and the ability to differentiate
    between surface-level blocks.
**Evaluation**: Run away. < Look down. < Dig down. < Break the dirt on the surface.

**Task**: collect grass
**Description**: Remove weeds on the surface.
**Precondition**: Spawn the player in the plains biome.
**SkillAssessed**: Surface navigation and comprehension of vegetation blocks.
**Evaluation**: Run away. < Break the grass block. < Break a large field of grass
    blocks.

**Task**: collect wood
**Description**: Cut down trees to collect wood.
**Precondition**: Spawn the player in the forest biome with an iron_axe in its hand.
**SkillAssessed**: Recognition of tree structures, efficient utilization of tools, and
     block harvesting capability.
**Evaluation**: Run away. < Approach trees. < Chop the tree and collect logs.

**Task**: collect seagrass
**Description**: Dive into the water and collect seagrass.
**Precondition**: Spawn the player near the sea.
**SkillAssessed**: Water navigation, diving mechanics understanding, and underwater
    block interaction.
**Evaluation**: Walk on the land. < Swim on the water < Dive into the water. < Break
    seagrass blocks.

**Task**: collect wool
**Description**: Dye and shear the sheep for wool.
```

```
Precondition: Spawn the player in the plains biome with a shear (mainhand) and a
    stack of blue_dye (offhand), 5 sheep near the player.
SkillAssessed: Interaction with entities, tool and item application, and
    sequential action execution.
Evaluation: Ignore the sheep. < Dye the sheep. < Shear the sheep. < First dye then
     shear the sheep.
```

Listing 1: The environment configuration and evaluation metric for `collect` series tasks.

M.2 Explore

The tasks encompassed within the `explore` category of our benchmark are intricately devised to evaluate an AI agent's navigation proficiency, understanding of diverse environments, and intrinsic motivation for exploration. Through these tasks, we gauge an agent's ability to actively traverse, understand, and interact with varied elements of the Minecraft world, and its propensity to unravel mysteries and challenges posed by the environment.

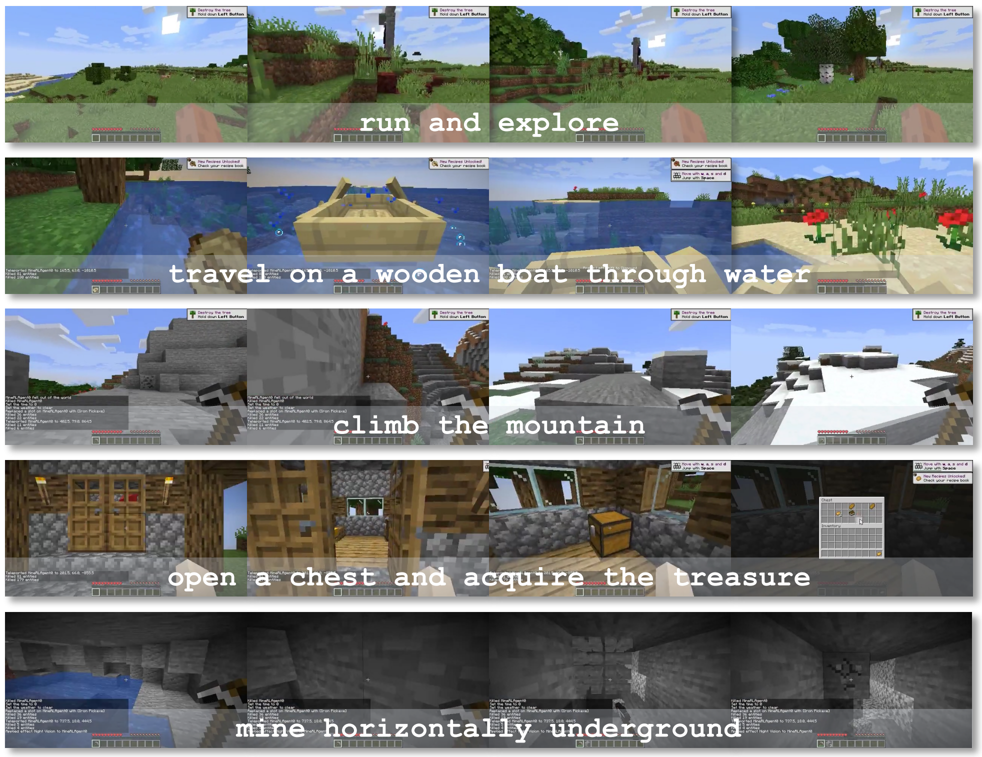

run and explore
travel on a wooden boat through water
climb the mountain
open a chest and acquire the treasure
mine horizontally underground

Figure 10: Examples of tasks in `explore` category.

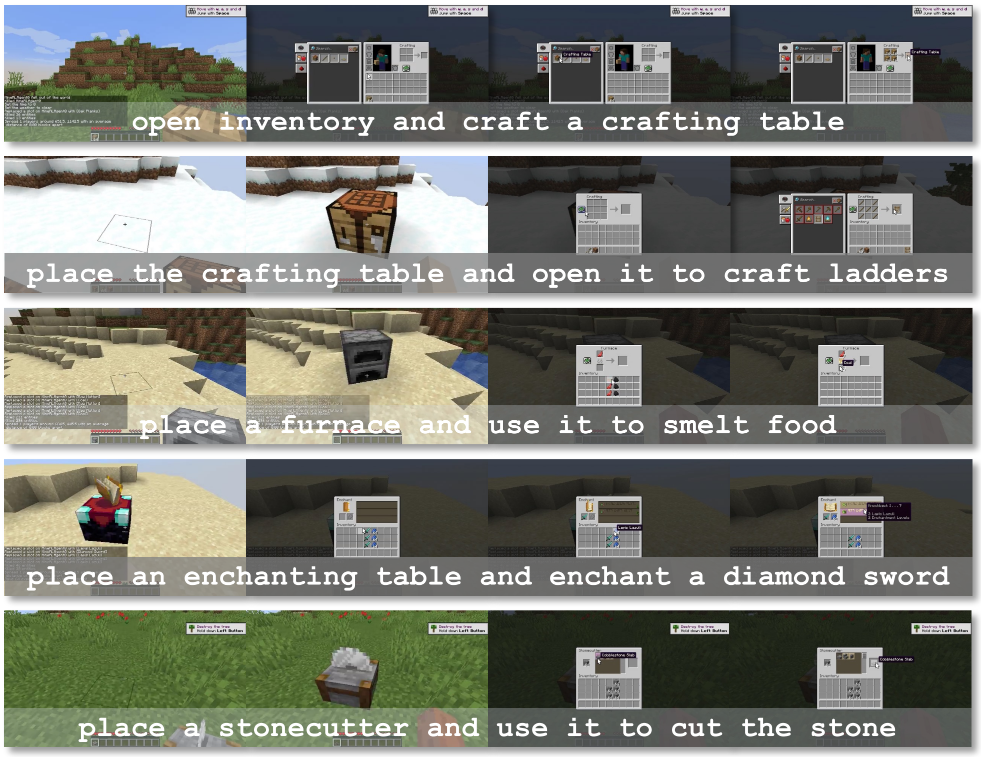
```
Task: run and explore
Description: Run and explore.
Precondition: Spawn the player in the plains biome.
SkillAssessed: Stamina utilization and distance-based exploration.
Evaluation: Exploring as far as possible.

Task: climb the mountain
Description: Climb the mountain.
Precondition: Spawn the player in the stone shore biome and near the mountain.
SkillAssessed: Vertical navigation, terrain adaptation, and goal-oriented movement.

Evaluation: Run away and ignore the mountain. < Approach the mountain. < Climbing
    the mountain. < Climb to the top of the mountain.

Task: mine horizontally
Description: Mine horizontally underground.
Precondition: Spawn the player in a deep cave with an iron_pickaxe in the hand.
```

```
SkillAssessed: Underground navigation, tool utilization, and spatial reasoning in
    confined spaces.
Evaluation: Run away. < Break the stone. < Dig down. < Mine horizontally.

Task: travel by boat
Description: Travel on a wooden boat through water.
Precondition: Spawn the player near the sea with a wooden boat in the hand.
SkillAssessed: Aquatic travel, tool placement, and boat maneuverability.
Evaluation: Did not place the boat. < Place the boat on the water. < Board the
    boat. < Row in the water.

Task: explore the treasure
Description: Rush into a villager's home and open a chest and acquire the treasure.

Precondition: Spawn the player in front of a villager's house.
SkillAssessed: Interaction with structures, curiosity-driven exploration, and
    object acquisition.
Evaluation: Ignore the house and run away. < Open the door. < Enter the house. <
    Open the chest. < Acquire the treasure.
```

Listing 2: The environment configuration and evaluation metric for `explore` series tasks.

M.3 Craft

The tasks under the `craft` category in our benchmark have been designed to shed light on an AI agent's prowess in item utilization, the intricacies of Minecraft crafting mechanics, and the nuances of various game mechanic interactions. These tasks provide a detailed examination of an agent's capability to convert materials into functional items and harness the game's various crafting and enhancement mechanics.



open inventory and craft a crafting table
place the crafting table and open it to craft ladders
place a furnace and use it to smelt food
place an enchanting table and enchant a diamond sword
place a stonecutter and use it to cut the stone

Figure 11: Examples of tasks in `craft` category.

```
Task: craft the crafting_table
Description: Open inventory and craft a crafting table.
Precondition: Spawn the player in the plains biome with a stack of oak_planks in
    the inventory.
```

```
SkillAssessed: Inventory management and basic crafting.
Evaluation: Open the inventory. < Click on the recipe button. < Click on the
    crafting_table. < Drag the crafting_table into the inventory.

Task: craft ladders
Description: Place the crafting table and open it to craft ladders.
Precondition: Spawn the player in the plains biome with a crafting_table in its
    main hand and a stack of oak_planks in the inventory.
SkillAssessed: Advanced crafting using crafting stations and recipe navigation.
Evaluation: Place the crafting_table on the surface. < Open the crafting_tabe. <
    Click on the recipe book. < Click on the ladder. < Drag the ladder into the
    inventory.

Task: enchant sword
Description: Place an enchanting table and use it to enchant a diamond sword.
Precondition: Spawn the player in the plains biome with an enchanting table in its
     main hand, 3 diamond swords, and 3 stacks of lapis_lazuli in the inventory.
SkillAssessed: Tool enhancement using enchantment stations and decision-making in
    choosing enchantments.
Evaluation: Place the enchanting_table on the surface. < Open the enchanting_table.
     < Place the lapis_lazuli or diamond sword. < Place the lapis_lazuli and
    diamond sword. < Choose any enchantment.

Task: smelt food
Description: Place a furnace and use it to smelt food.
Precondition: Spawn the player in the plains biome with a furnace table in its
    main hand, 3 stacks of mutton, and 3 stacks of coal in the inventory.
SkillAssessed: Food processing using a smelting furnace, raw material to product
    conversion, and patience in awaiting outcomes.
Evaluation: Place the furnace on the surface. < Open the furnace. < Place raw meat
     or coal. < Place both raw meat and coal. < Wait for the raw meat to be cooked.
     < Take out cooked meat.

Task: cut stone
Description: Place a stonecutter and use it to cut stones.
Precondition: Spawn the player in the plains biome with a stonecutter in its main
    hand, 6 stacks of stones in the inventory.
SkillAssessed: Tool enhancement using enchantment stations and decision-making in
    choosing enchantments.
Evaluation: Place the stonecutter on the surface. < Open the stonecutter. < Place
    the stones. < Select a target type of stone. < Drag stones to the inventory.
```

Listing 3: The environment configuration and evaluation metric for `craft` series tasks.

M.4 Tool

The tasks within the `Tool` category of our benchmark are designed to deeply investigate an AI agent's capabilities in tool utilization, precision in tool handling, and contextual application of various tools to carry out specific tasks. This category provides insights into the agent's skill in wielding, using, and exploiting tools optimally within different Minecraft scenarios.

```
Task: use bow
Description: Draw a bow and shoot.
Precondition: Spawn the player in the plains biome with a bow in the mainhand and
    a stack of arrows in the inventory.
SkillAssessed: Precision, tool handling, and projectile mastery.
Evaluation: Just run. < Draw the bow and shoot the arrow. < Hold the bow steady
    and charge up the shot before releasing the arrow.

Task: set fires
Description: Set fires on the trees.
Precondition: Spawn the player in the forest biome with a flint_and_steel in its
    main hand.
SkillAssessed: Environment manipulation and controlled chaos creation.
Evaluation: Attack the tree. < Start a fire with the flint_and_steel. < Go wild
    with the fire.

Task: lead animals
Description: Use rein to tie up the animals.
Precondition: Spawn the player in the plains biome with a stack of leads in its
    main hand. Spawn 5 sheep and 5 cows near the player's position.
```

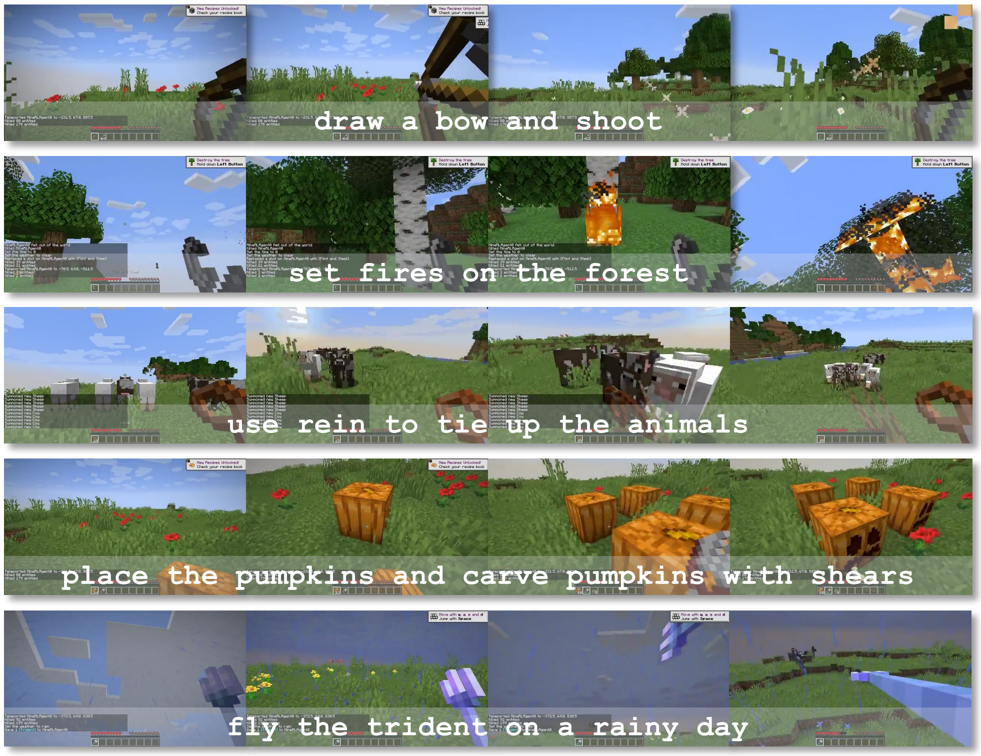

draw a bow and shoot
set fires on the forest
use rein to tie up the animals
place the pumpkins and carve pumpkins with shears
fly the trident on a rainy day

Figure 12: Examples of tasks in `tool` category.

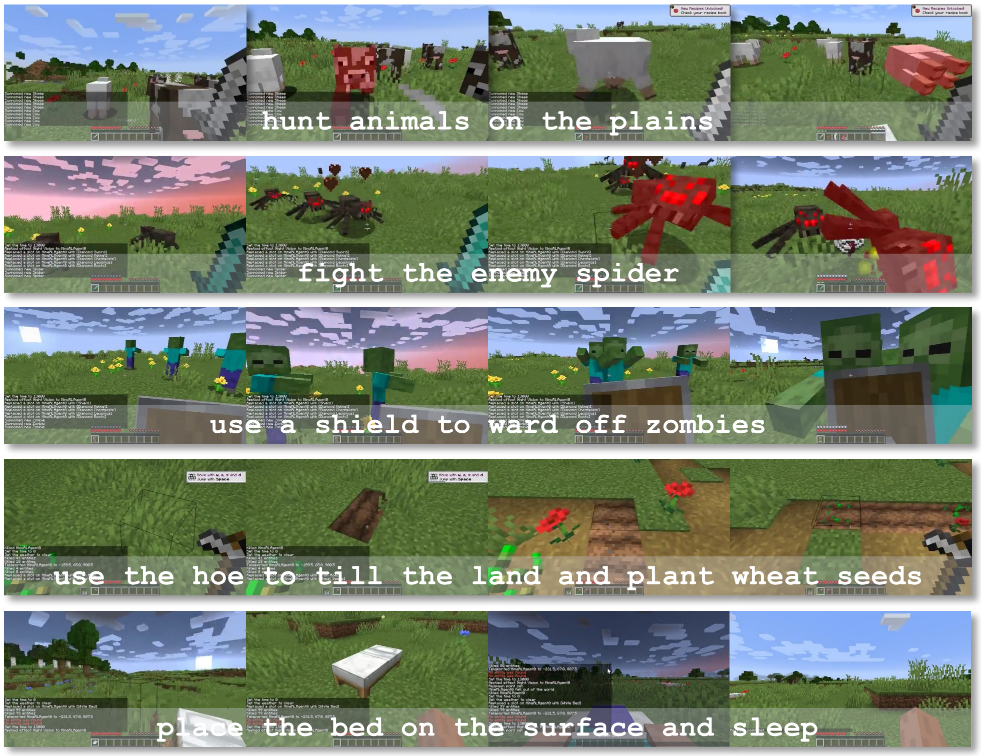
```
SkillAssessed: Entity interaction, tool application on moving entities, and
    livestock
Evaluation: Ignore the animals and run away. < Use the rein to tie up animals.

Task: carve pumpkins
Description: Place the pumpkins and carve pumpkins with shears.
Precondition: Spawn the player in the plains biome with a shear in its main hand
    and a stack of pumpkins in the inventory.
SkillAssessed: Block placement, block modification, and crafting interaction.
Evaluation: Just run. < Place the pumpkin on the surface. < Use the shear to carve
     it. < Get a carved pumpkin.

Task: use trident
Description: Fly the trident on a rainy day.
Precondition: Spawn the player in the plains biome with a trident in the main hand,
     which is enchanted with riptide. The weather is rain.
SkillAssessed: Weather-adaptive tool utilization, motion dynamics, and advanced
    weapon handling.
Evaluation: Just run. < Use the trident to break the block. < Use the trident for
    quick movement. < Charge to throw the trident farther.
```

Listing 4: The environment configuration and evaluation metric for `tool` series tasks.

M.5 Survive

The tasks embedded within the `survive` category of our benchmark aim to analyze an AI agent's ability to ensure its own survival, adeptness in combat scenarios, and its capability to interact with the environment in order to meet basic needs. Survival, being a core aspect of Minecraft gameplay, necessitates an intricate balance of offensive, defensive, and sustenance-related actions. This category is structured to ensure a thorough evaluation of these skills.

```
Task: hunt animals
Description: Hunt animals on the plains.
```



hunt animals on the plains
fight the enemy spider
use a shield to ward off zombies
use the hoe to till the land and plant wheat seeds
place the bed on the surface and sleep

Figure 13: Examples of tasks in `survive` category.

```
Precondition: Spawn the player in the plains biome with an iron sword in the main
    hand. Spawn 5 sheep and 5 cows near the player's position.
SkillAssessed: Predator instincts, combat efficiency, and sustenance acquisition.
Evaluation: Ignore animals and run away. < Hurt animals. < Kill animals.

Task: combat enemies
Description: Fight the enemy spider.
Precondition: Spawn the player in the plains biome with a diamond sword in its
    main hand and a suite of diamond equipment. Spawn 3 spiders in front of the
    player.
SkillAssessed: Self-defense, offensive combat strategy, and equipment utilization.
Evaluation: Ignore spiders and run away. < Hurt spiders. < Kill spiders.

Task: use shield
Description: Use a shield to ward off zombies.
Precondition: Spawn the player in the plains biome with a shield in its main hand
    and a suite of diamond equipment. Spawn 3 zombies in front of the player.
SkillAssessed: Defensive tactics, tool application in combat, and strategic
    protection.
Evaluation: Ignore zombies and run away. < Use the shield to protect itself.

Task: plant wheats
Description: Use an iron_hoe to till the land and then plant wheat seeds.
Precondition: Spawn the player in the plains biome with an iron hoe in its main
    hand, and a stack of wheat seeds in the off hand.
SkillAssessed: Land cultivation, planting proficiency, and sustainable resource
    creation.
Evaluation: Just run away. < Till the land. < Plant the wheats.

Task: sleep on the bed
Description: Place the bed on the surface and sleep.
Precondition: Spawn the player in the plains biome with a white bed in its main
    hand.
SkillAssessed: Self-preservation, understanding of day-night cycle implications,
    and use of utilities for rest.
```

```
Evaluation: Just run away. < Place the bed on the surface. < Sleep on the bed.
```

Listing 5: The environment configuration and evaluation metric for `survive` series tasks.

M.6 Build

The tasks within the `build` category of our benchmark are devised to evaluate an AI agent's aptitude in structural reasoning, spatial organization, and its capability to interact with and manipulate the environment to create specific structures or outcomes. Building is an integral component of Minecraft gameplay, requiring an intricate interplay of planning, creativity, and understanding of block properties.

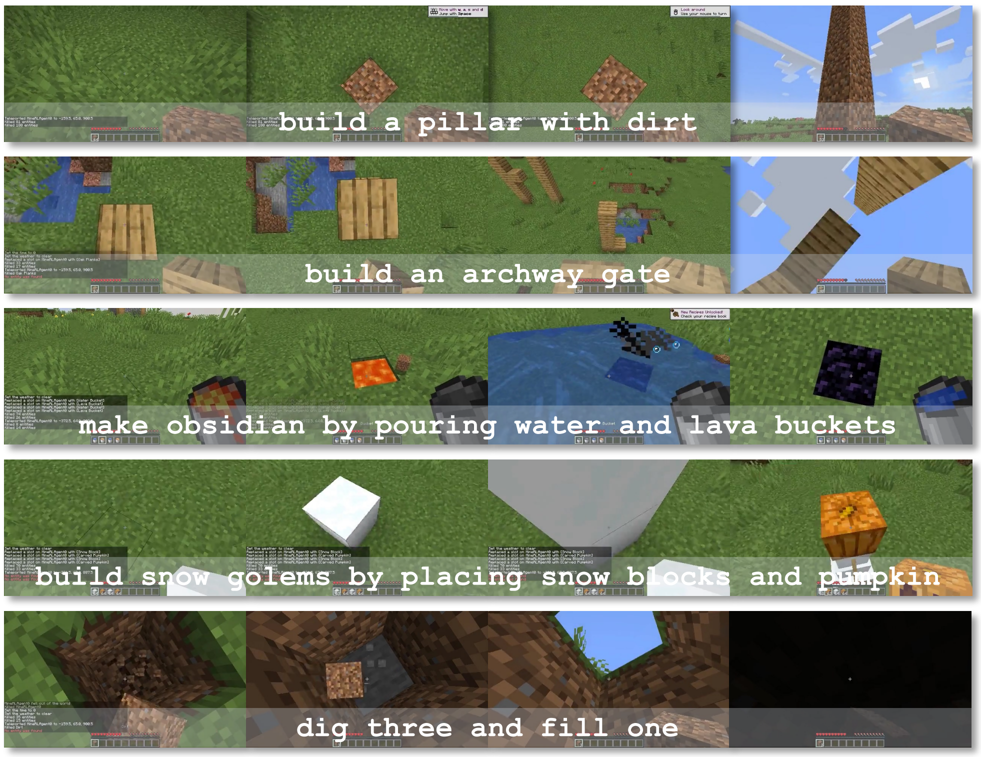

build a pillar with dirt
build an archway gate
make obsidian by pouring water and lava buckets
build snow golems by placing snow blocks and pumpkin
dig three and fill one

Figure 14: Examples of tasks in `build` category.

```
Task: build pillar
Description: Build a pillar with dirt.
Precondition: Spawn the player in the plains biome with a stack of dirt in the
    main hand.
SkillAssessed: Vertical construction and basic structure formation.
Evaluation: Just run away. < Look down. < Jump and place the dirt. < Pile the dirt
     into a few pillars. < Make a really high pillar.

Task: dig three down and fill one up
Description: Dig three dirt blocks and fill the hole above.
Precondition: Spawn the player in the plains biome.
SkillAssessed: Ground manipulation and depth perception.
Evaluation: Just run away. < Look down. < Dig down three dirt blocks. < Raise the
    head. < Raise the head and use dirt to fill the hole.

Task: build gate
Description: Build an archway gate.
Precondition: Spawn the player in the plains biome with a stack of oak_planks in
    the main hand.
SkillAssessed: Symmetry, planning, and aesthetic construction.
Evaluation: Place no plank. < Build 1 pillar. < Build 2 pillars. < Build an
    archway gate.
```

```
Task: build obsidian
Description: Make obsidian by pouring a water bucket and a lava bucket.
Precondition: Spawn the player in the plains biome with two water buckets and two
    lava buckets in the Hotbar.
SkillAssessed: Material transformation, understanding of in-game chemistry, and
    precise pouring.
Evaluation: Just run away. < Pour water or lava. < Pour both liquids. < Pour into
    a mold to make obsidian.

Task: build snow golems
Description: Build snow golems by placing two snow blocks and one carved pumpkin.
Precondition: Spawn the player in the plains biome with two stacks of snow blocks
    and two stacks of carved pumpkins in the Hotbar.
SkillAssessed: Entity creation, sequential block placement, and combination of
    multiple materials.
Evaluation: Place no block. < Place at least one kind of block. < Place both kinds
     of blocks. < Build a snow golem.
```

Listing 6: The environment configuration and evaluation metric for `build` series tasks.