A Segmentation Details

A.1 Extracting Targets via LLM

In order to extract the target name from a natural language instruction, we resort to GPT-4 [1]. We input the following prompt:

```
You are a helper agent in Minecraft. I will describe a task, and
your job is to identify the specific entity (name of blocks, anim-
als, or items) that I need to directly interact with to complete
this task. This means you should tell me what entity I need to aim
at with the crosshair in the center of my screen, assuming I have
all the necessary items and tools in my inventory. Please provide
the name of the target entity.

Requirements:

1. For tasks with multiple possible solutions or those requiring
additional context, provide the most straightforward or common
solution.

2. Only output the target entity after "Target: ", DO NOT add other
examples, descriptions, or explanations.

For example:

Task: "Obtain 8 obsidian blocks with a diamond pickaxe and a water
bucket."

Thinking: "To mine obsidian, the player must aim their crosshair at
an obsidian block and use a diamond or netherite pickaxe on it. Ob-
sidian is created when flowing water touches a lava source block.
Therefore, the entity to aim at for this task is obsidian."

Target: "Obsidian"

Task: "Obtain milk with an empty bucket."

Thinking: "To fill a bucket with milk, the player must aim their
crosshair at a cow and use the empty bucket on it. Therefore, the
entity to aim at for this task is a cow."

Target: "Cow"
```

We try some instructions without telling the name of the true target entity:

```
We:

Task: "Harvest wool"

Thinking:

GPT-4:
```

To harvest wool in Minecraft, you need to aim your crosshair at a sheep and use shears on it. This will allow you to collect wool without harming the sheep.

Target: "Sheep"

We:

Task: "Collect beef"

Thinking:

GPT-4:

To collect beef in Minecraft, you need to aim your crosshair at a cow and use a weapon or your hand to kill it. Upon death, the cow will drop beef.

Target: "Cow"

We:

Task: "Get mutton as food"

Thinking:

GPT-4:

To get mutton as food in Minecraft, you need to aim your crosshair at a sheep and use a weapon or your hand to kill it. Upon death, the sheep will drop mutton.

Target: "Sheep"

These results show that GPT-4 understands the underlying relationship between the target item (`wool`, `beef`, `mutton`) and the target animal (`sheep`, `cow`, `sheep`) based on its prior knowledge about Minecraft. Certainly, GPT-4 also works on instructions containing the target entity directly:

We:

Task: "Harvest a flower"

Thinking:

GPT-4:

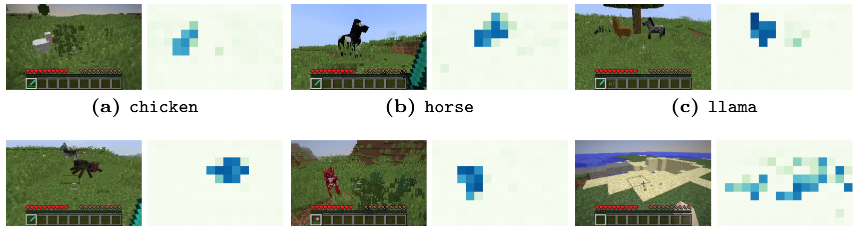
To harvest a flower in Minecraft, you need to aim your crosshair at the specific flower you want to collect and either use your hand or any tool to break and collect it.

Target: "Flower"

We:

Task: "Hunt a cow"

Thinking:



(a) chicken (b) horse (c) llama

(d) spider (e) mushroom cow (f) sand

Fig. 6: Confidence map instances of targets: (a) `chicken`, (b) `horse`, (c) `llama`, (d) `spider`, (e) `mushroom cow`, and (f) `sand`.

GPT-4:

```
To hunt a cow in Minecraft, you need to aim your crosshair at a cow
and use a weapon or your hand to kill it.

Target: "Cow"
```

A.2 Negative Words

For better segmentation, the negative word list should contain names of objects that frequently appear in Minecraft. To this end, we utilize the TF-IDF algorithm to select top-100 words from the subtitles of YouTube videos [17], excluding stop words like "we" and "is", as well as modal particles such as "yeah" and "uh". Additionally, we filter out verbs and some irrelevant nouns from the top-100 words to reduce noise. The final negative word list is shown below:

```
diamond, block, village, house, iron, farm, chest, dragon,
redstone, water, tree, zombie, sword, stone, door, armor, lava,
fish, portal, chicken, wood, wall, glass, cave, stair, bed,
torch, fire, creeper, island, food, slab, book, head, button,
apple, skeleton, potion, spider, egg, pickaxe, arrow, boat,
horse, hopper, box, wool, table, seed, cow, brick, trap, dog,
bow, dirt, roof, leaves, sand, window, bucket, coal, hole, pig,
ice, bone, stick, flower, tower, sheep, grass, sky.
```

Furthermore, in constructing text embeddings, we employ prompt engineering to improve zero-shot ability on classification [44]. Same as MaskCLIP [67], we utilize 85 prompt templates such as "`a photo of many {}.`". The mean of these embeddings is set to be the text embedding of the target. During segmentation, if the target object already exists in the list, it will be removed from the list in advance.

A.3 Segmentation Results

We provide more examples of confidence maps, as illustrated in Figure 6. Our modified MineCLIP effectively locates these target objects.

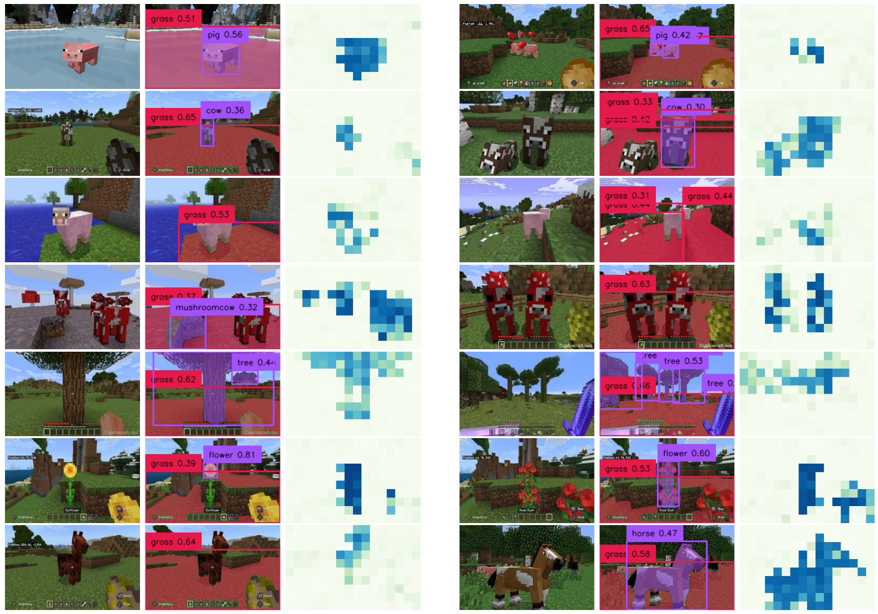


Fig. 7: Comparison between Grounded SAM and our method on seven objects.

A.4 Off-the-shelf Object Detection Models

We choose one off-the-shelf object detection model, Grounded SAM [25, 33], to evaluate its effectiveness in Minecraft. In order to conduct a fair side-by-side comparison between it and our method, we Google searched "minecraft [object name] screenshot" in the image tab, and selected the first two images that include objects and have them fully in the field of view. The evaluation objects includes `pig`, `cow`, `sheep`, `mushroom cow`, `tree`, `flower`, and `horse`. We follow the setting in the official demo[5] to evaluate the effectiveness of Grounded SAM on detecting these objects in Minecraft. For both Grounded SAM and our modified MineCLIP in this evaluation, we use the same word list which consists of the seven evaluation objects and `grass`.

The detection results of the two methods are illustrated in Figure 7. For a more detailed evaluation, we quantify the number of objects present in each image, the number detected by Grounded SAM, and the number detected by our method. These quantitative results are summarized in Table 5. Across all images, there are 24 target objects. Grounded SAM can successfully identify 14 objects, which translates to a detection rate of 58.3%. In contrast, our method demonstrates a significantly higher efficacy, successfully detecting 22 of the 24 objects, achieving a detection rate of 91.7%. There are two failures in our method. One is the sunflower in the bottom-right corner of the first flower image, and the

[5] `https://github.com/IDEA-Research/Grounded-Segment-Anything/blob/main/grounded_sam_colab_demo.ipynb`

Table 5: Result statistics of Grounded SAM and our method on seven objects.

Object	Ground-truth	Grounded SAM	Ours
`pig`	3	3	3
`cow`	3	2	3
`sheep`	2	0	2
`mushroom cow`	4	0	4
`tree`	6	6	5
`flower`†	4	2	3
`horse`	2	1	2
total	**24**	**14**	**22**

† We also count the two flowers held in players' hands.

other is the tree in the left of the second tree image. In both cases, our method generates some activation in the target patches, but it does not cover the entire object (flower) or is relatively weak (tree). We regard them as failures for a more strict result.

B Policy Learning Details

B.1 Observation Space and Action Space

The observation space adopted in our experiments consists of RGB, compass, GPS, voxels, and biome index. The shape and description of each modality are listed in Table 6. We simplify the original action space of MineDojo [17] into a 2-dimensional multi-discrete action space. The first dimension contains 12 discrete actions about movement: *no_ op*, *move forward*, *move backward*, *move left*, *move right*, *jump*, *sneak*, *sprint*, *camera pitch* -30, *camera pitch* +30, *camera yaw* -30, and *camera yaw* +30. The second dimension includes 3 discrete actions about interacting with items: *no_ op*, *attack*, and *use*.

Table 6: Observation space adopted in our experiments.

Modality	Shape	Description
RGB	(3, 160, 256)	Ego-Centric RGB frames.
Compass	(4,)	Sine and cosine of yaw and pitch.
GPS	(3,)	GPS location of the agent.
Voxels	(27,)	Indices of $3 \times 3 \times 3$ surrounding blocks.
Biome_ID	(1,)	Index of the environmental biome.

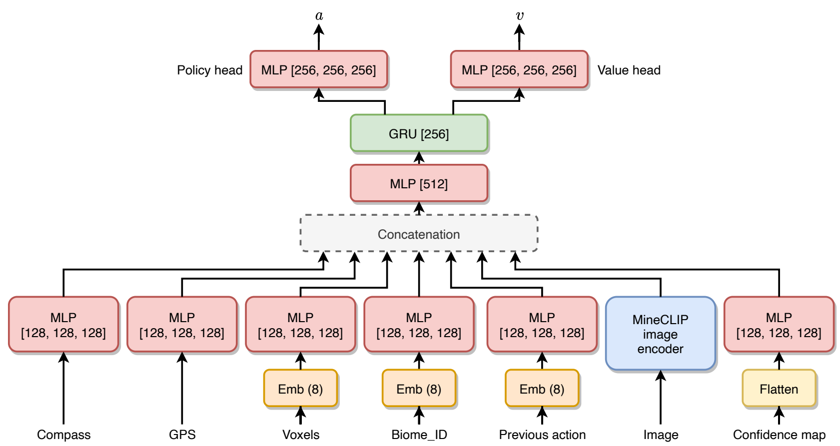


Fig. 8: Network architecture of COPL agent.

B.2 Network Architecture

The input of COPL agent includes observations from the environment listed in Table 6, the agent's action taken at the last time step a_{t-1}, and the confidence map. As illustrated in Figure 8, all inputs except the RGB image are processed by MLPs with three hidden layers and ReLU activation. In this step, voxels, biome index, and previous action are first embedded into dense vectors. The RGB image is processed using the MineCLIP image encoder to generate an embedding. All these processed features are concatenated and processed by an MLP with one hidden layer and ReLU activation. Then a GRU layer is implemented to integrate the historical information. The policy head and the value head take as input the output of GRU and both process it using an MLP with three hidden layers and ReLU activation. The policy head generates the distribution of actions, and the value head outputs the estimated value of the current state. Some variants are as follows: (1) **Single-task** model: In single-task experiments, the agent *does not* take as input the confidence map; (2) **LCRL**: The branch of confidence map is replaced by the MineCLIP text encoder processing the target name or the instruction; (3) **One-Hot**: The branch of confidence map is replaced by an MLP processing the one-hot vector which indicates the index of the current task. The MLP has one hidden layer with size 32 and ReLU activation.

C Single-Task Experiments

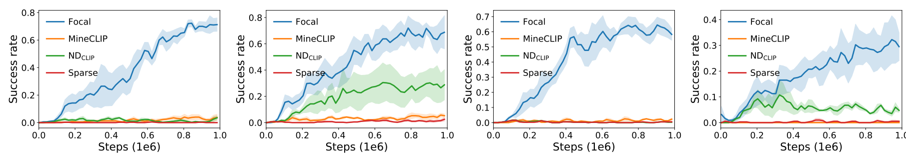
C.1 Settings

Our single-task experiments include four tasks: `hunt a cow`, `hunt a sheep`, `hunt a pig`, and `hunt a chicken`. The parameters we used to make environments in MineDojo are listed in Table 7. In all tasks, the agent spawns in `plains`

Table 7: Single-task settings in our experiments.

Task	Target	Initial Animals	Range[1]	Inventory	Biome	Length[2]
`hunt a cow`	cow	cow, sheep, pig	7	diamond_sword	plains	200
`hunt a sheep`	sheep	cow, sheep, pig	7	diamond_sword	plains	200
`hunt a pig`	pig	cow, sheep, pig	7	diamond_sword	plains	200
`hunt a chicken`	chicken	cow, sheep, chicken	7	diamond_sword	plains	200

[1] Range indicates the spawn range of initial animals.
[2] Length indicates the maximum length of one episode.



(a) `hunt a cow` (b) `hunt a sheep` (c) `hunt a pig` (d) `hunt a chicken`

Fig. 9: Learning curves of PPO with focal reward, MineCLIP reward, $\mathrm{ND}_{\mathrm{CLIP}}$ reward, and environmental sparse reward only, on four Minecraft tasks.

biome holding a diamond sword. Several animals including the target spawn near the agent. The agent will receive a +100 reward after successfully killing the target animal. Each episode is limited to a maximum of 200 steps. The challenge lies in the fact that animals will flee after being attacked, thus requiring the agent to keep chasing the target and attacking. Killing a cow, sheep, or pig requires at least two attacks, while killing a chicken only requires at least one attack. Although it takes fewer attacks to kill a chicken, aiming at the small size of the chicken poses an additional challenge. For ablation experiments on Gaussian kernel, we double the initial animals and increase the animal spawn range to 10.

C.2 Learning Curves

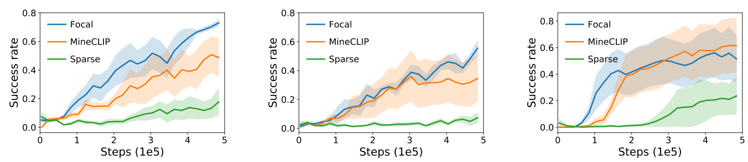
Learning curves of four methods on for Minecraft tasks are shown in Figure 9. Each curve shows the mean success rate of four runs with different seeds and shaded regions indicate standard error (the same applies hereinafter). We can observe that only our focal reward leads to the mastery of all four skills by guiding the agent to consistently approach the target.

C.3 Additional Experiments

We conduct additional single-task experiments on three harvest tasks including `milk a cow`, `shear a sheep`, and `chop a tree`, where MineCLIP reward achieves nonzero success rates [17]. The environment parameters for each task

Table 8: Single-task settings in additional experiments.

Task	Target	Initial Animals	Range	Inventory	Biome	Length
`milk a cow`	milk_bucket	cow, sheep, pig	10	bucket	plains	200
`shear a sheep`	wool	cow, sheep, pig	10	shears	plains	200
`chop a tree`	log	cow, sheep, pig	7	golden_axe	forest	200



(a) `milk a cow` **(b)** `milk a cow` **(c)** `milk a cow`

Fig. 10: Learning curves of PPO with focal reward, MineCLIP reward, and environmental sparse reward only, on three Minecraft tasks: (a) `milk a cow`, (b) `shear a sheep`, and (c) `chop a tree`.

can be found in Table 8. As shown in Figure 10, our focal reward outperforms MineCLIP reward on `milk a cow` and `shear a sheep`. Regarding `chop a tree`, our focal reward and MineCLIP reward achieve similar performance, both with 3 out of 4 runs having learned this skill. To break a wood block, the agent needs to continuously take *attack* actions for around 6 steps. Therefore, we believe that the main challenge for RL in this task lies in exploration. It is difficult for an RL algorithm, such as PPO, with a stochastic policy to explore and exploit a behavior pattern that requires consecutive actions over 6 steps, especially given the sparse environmental reward signal. Using an off-policy RL algorithm or self-imitation may help address this problem.

To quantitatively assess the accuracy of our focal reward as a proxy for estimating the distance between the agent and the target object, we calculate the correlation coefficient between the reward and the ground-truth distance to the target object at each step. The ground-truth distance to the target object is obtained through an internal function of the environment simulator [17], which is typically not available during normal gameplay and not available in all environments. Therefore, for generality, we cannot directly use the distance as a reward for training. We run our trained models on four tasks. For each task, we sample 10,000 steps and record focal rewards, MineCLIP rewards, and ground-truth distance. If the target object is not in the sight of the agent, the ground-truth distance is set to the *inf*. Given that the data distribution is not close to a normal distribution, we calculate the Spearman rank correlation coefficient instead of the Pearson correlation coefficient, as shown in Table 9. The results demonstrate that the correlation between our focal reward and the distance to the target object is higher than the correlation between the MineCLIP reward and

Table 9: Spearman's rank correlation coefficient between rewards and distance to the target object in sight.

Rewards	`cow`	`sheep`	`pig`	`chicken`
Focal	**-0.704**	**-0.701**	**-0.761**	**-0.835**
MineCLIP	-0.374	-0.585	-0.460	-0.544

the distance, suggesting that our focal reward better approximates the actual distance.

D Multi-Task Experiments

D.1 Settings

Hunting domain. The hunting domain consists of four instructions: "`hunt a cow`", "`hunt a sheep`", "`hunt a pig`", and "`hunt a chicken`". At the start of each episode, one instruction is randomly selected, and an environment is built with the parameters listed in Table 10. The agent will receive a +100 reward after successfully killing the target animal specified in the instruction. If the agent mistakenly kills the animal which is the target of other instructions, no reward is given and the episode ends. This setup encourages the agent to attack the correct animal rather than indiscriminately attacking any animal.

The object-level generalization evaluation for the hunting domain also contains four instructions: "`hunt a mushroom cow`", "`hunt a spider`", "`hunt a llama`", and "`hunt a horse`". The environment parameters can be found in Table 11. We slightly increase the maximum episode length for "`hunt a spider`", "`hunt a llama`", and "`hunt a horse`", given that killing them requires more attacks as a result of their higher health compared to other animals. For each instruction, we run the test model for 100 episodes to calculate its success rate and precision (same in the harvest domain).

Harvest domain. The harvest domain consists of four instructions: "`milk a cow`", "`shear a sheep`", "`harvest a flower`", and "`harvest leaves`". Same as the hunting domain, one instruction is randomly selected at the start of each episode, and an environment is generated with the parameters listed in Table 12. The agent will receive a +100 reward after successfully acquiring the target item. If the agent mistakenly acquires the target item corresponding to other instructions, no reward is given and the episode ends. Note that *the target item required to finish the task may not always be the same as the target object that the agent needs to approach.* For example, in the instruction "`milk a cow`", the target item is a milk_bucket, while the target object that the agent needs to approach is a cow.

The object-level generalization evaluation for the harvest domain contains four instructions: "`harvest water`", "`shear a mushroom cow`", "`collect sand`",

Table 10: Multi-task settings in the hunting domain.

Instruction	Target	Initial Animals	Range	Inventory	Biome	Length
`hunt a cow`	cow	cow, sheep, pig, chicken	10	diamond_sword	plains	500
`hunt a sheep`	sheep	cow, sheep, pig, chicken	10	diamond_sword	plains	500
`hunt a pig`	pig	cow, sheep, pig, chicken	10	diamond_sword	plains	500
`hunt a chicken`	chicken	cow, sheep, pig, chicken	10	diamond_sword	plains	500

Table 11: Generalization evaluation settings in the hunting domain.

Instruction	Target	Initial Animals	Range	Inventory	Biome	Length
`hunt a mush-room cow`	mushroom cow	mushroom cow, spider, llama, horse	10	diamond_sword	plains	500
`hunt a spider`	spider	mushroom cow, spider, llama, horse	10	diamond_sword	plains	800
`hunt a llama`	llama	mushroom cow, spider, llama, horse	10	diamond_sword	plains	800
`hunt a horse`	horse	mushroom cow, spider, llama, horse	10	diamond_sword	plains	800

and "`collect dirt`". For each instruction, there is a distraction item. If the agent mistakenly acquires the distraction item, no reward is given and the episode ends. For "`harvest water`", the distraction item is milk_bucket, as the agent can also acquire a bucket of milk with the given empty bucket from a cow. Similarly, the distraction items for the other three instructions are wool, dirt, and sand, respectively.

Here we briefly introduce the behavior patterns required by the harvest domain instructions. "`Milk a cow`" and "`harvest water`" require the agent to approach the target object (cow/water), aim at it, and take *use* action. "`Harvest a flower`", "`collect sand`", and "`collect dirt`" require the agent to approach the target object (flower/sand/dirt), aim at it, take *attack* action to break it, and move closer to pick up the dropped item. "`Shear a sheep`" and "`harvest leaves`" are the same except that they require taking *use* action instead of *attack* action. In all these tasks except "`collect sand`" and "`collect dirt`", the agent

Table 12: Multi-task settings in the harvest domain.

Instruction	Target[1]	Initial Animals	Range	Inventory	Biome	Length
`milk a cow`	milk_bucket	cow, sheep, pig	10	bucket	plains	200
`shear a sheep`	wool	cow, sheep, pig	10	shears	plains	200
`harvest a flo-wer`	red_flower	cow, sheep, pig	10	-	flower_forest	200
`harvest leaves`	leaves	cow, sheep, pig	10	shears	flower_forest	200

[1] Target here represents the parameter for making a MineDojo environment, *i.e.*, the target item required to finish the task. It differs from the target object specified in the instruction.

Table 13: Generalization evaluation settings in the harvest domain.

Instruction	Target	Initial Animals	Range	Inventory	Biome	Length
`harvest water`	water_ bucket	cow, sheep, mushroom cow	10	bucket	river	200
`shear a mushroom cow`	mushroom	cow, sheep, mushroom cow	10	shears	plains	200
`collect sand`	sand	cow, sheep, mushroom cow	10	diamond_shovel	river	200
`collect dirt`	dirt	cow, sheep, mushroom cow	10	diamond_shovel	river	200

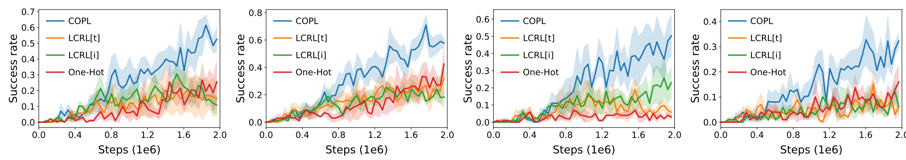
needs only one *attack* or *use* action to finish the task. Therefore, individually, the harvest tasks are easier than the hunting tasks. "`Collect sand`" and "`collect dirt`" require the agent to continuously *attack* multiple times to break a sand or dirt block.

D.2 Learning Curves

We show the learning curves of COPL, LCRL[t], LCRL[i], and One-Hot on each task. As illustrated in Figures 11 and 12.

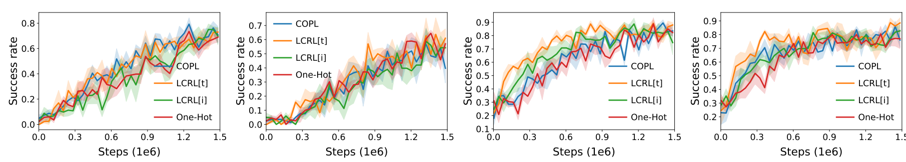
D.3 Precision

In the hunting domain, precision is defined as the number of correct kills on the specified target animal divided by the number of kills on any animal. The high precision, as reported in Table 14, proves COPL's ability to distinguish the target animal from other animals, rather than indiscriminately attacking them, even if these animals are all out of the training scope. As for the harvest domain, precision is related to the distraction item claimed in Appendix D.1, as harvesting the distraction item requires the same tool as the target item. Precision shown



(a) `hunt a cow` **(b)** `hunt a sheep` **(c)** `hunt a pig` **(d)** `hunt a chicken`

Fig. 11: Learning curves of COPL, LCRL[t], LCRL[i], and One-Hot on four hunting instructions: (a) "**`hunt a cow`**", (b) "**`hunt a sheep`**", (c) "**`hunt a pig`**", and (d)"**`hunt a chicken`**".



(a) `milk a cow` **(b)** `shear a sheep` **(c)** `harvest a flower` **(d)** `harvest leaves`

Fig. 12: Learning curves of COPL, LCRL[t], LCRL[i], and One-Hot on four harvest instructions: (a) "**`milk a cow`**", (b) "**`shear a sheep`**", (c) "**`harvest a flower`**", and (d)"**`harvest leaves`**".

in Table 15 is defined as the number of times correctly harvesting the specified target item divided by the total number of times harvesting the target item or the distraction item. Similar to the results in the hunting domain, COPL exhibits better identification ability on unseen targets, compared to LCRL[t] and LCRL[i], suggesting that the zero-shot object-level generalization of COPL emerges from grounding the unseen target object in a simple two-dimensional visual representation, given that all methods perform equally on training tasks.

E Hyperparameters

E.1 PPO Hyperparameters

In our experiments, we use PPO [48] as our base RL algorithm. Table 16 lists the hyperparameters for PPO across all tasks. Unlike MineAgent [17], our implementation does not include self-imitation learning and action smoothing loss. We find that vanilla PPO is able to achieve high performance in our experiments. For single-task experiments, we train RL models for 1,000,000 environment steps. For multi-task experiments in the hunting domain, we train RL models for 2,000,000 environment steps. For multi-task experiments in the harvest domain, we train RL models for 1,500,000 environment steps.

E.2 Intrinsic Reward Coefficient

To determine the optimal scale of intrinsic reward that can effectively guide reinforcement learning while avoiding conflicts with the environmental reward,

Table 14: Precision (%) on hunting test tasks.

Tasks	COPL	LCRL[t]	LCRL[i]	[7]	STEVE-1
`llama`	**88.8±7.5**	43.3±13.5	55.8±16.7	36.7	6.3
`horse`	**89.8±7.5**	13.3±2.2	14.8±11.2	28.0	9.4
`spider`	**96.8±1.9**	47.0±33.0	56.5±40.0	23.3	93.6
`mushroom`	**97.0±2.0**	69.0±26.9	0.0±0.0	55.3	6.8
Avg.	**93.1±3.4**	41.5±3.8	42.3±2.1	35.8	29.0

Table 15: Precision (%) on harvest test tasks.

Tasks	COPL	LCRL[t]	LCRL[i]	STEVE-1
`water`	55.8±11.9	31.8±11.6	28.5±3.0	**91.3**
`mushroom`	**74.8±10.5**	56.3±8.2	48.8±9.1	0.0
`sand`	**70.5±31.2**	10.5±12.6	14.3±10.9	12.0
`dirt`	95.3±2.6	57.8±43.1	60.8±23.5	**98.6**
Avg.	**74.1±10.4**	39.3±16.4	38.1±7.3	50.5

we conduct an experiment to evaluate the performance of our focal reward with different λ values. Figures 13a and 13b illustrates the performance of our focal reward with different λ, including 0.5, 5, and 50, on `hunt a cow` and `hunt a sheep`. Focal reward with $\lambda = 5$ outperforms $\lambda = 50$ and $\lambda = 0.5$ on two tasks. Therefore, we consistently set $\lambda = 5$ for all experiments in the main text.

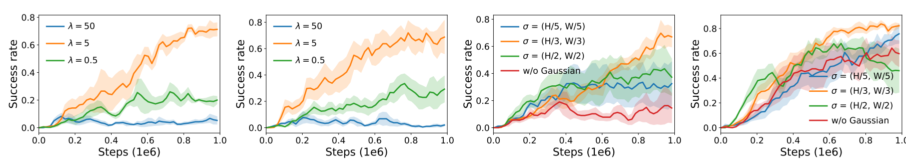
Regarding the MineCLIP reward, we set the coefficient to 1.0, following the original setting of MineAgent in [17]. The optimal coefficient of ND reward in [55] for *find* task is 0.003, and its sparse environmental reward is 1.0. Considering the environmental reward is set to 100 in our experiments, we decided to increase the coefficient for $\mathrm{ND_{CLIP}}$ from 0.003 to 0.3 in our implementation.

E.3 Gaussian Kernel

The introduction of a Gaussian kernel is to guide the agent to center a target object within its field of view. The Gaussian kernel should create a high contrast between the center and the edge, as well as between the edge and areas outside the field of view. Therefore, the variance of the Gaussian kernel would influence the performance of the focal reward. To evaluate the impact of different variances, we conduct an experiment with $\sigma = (H/5, W/5)$, $\sigma = (H/3, W/3)$, and $\sigma = (H/2, W/2)$. As illustrated in Figures 13c and 13d, $\sigma = (H/3, W/3)$ outperforms the others. We suppose that a wider Gaussian kernel with $\sigma = (H/2, W/2)$ fails to provide sufficient contrast between the center and the edge. Conversely, a narrower Gaussian kernel with $\sigma = (H/5, W/5)$ cannot provide sufficient contrast between the edge and areas outside the field of view.

Table 16: Hyperparameters for PPO across all tasks.

Hyperparameter	Value
num steps	1000
num envs	4
num minibatches	4
num epoches	8
GAE lambda	0.95
discounted gamma	0.99
entropy coef	0.005
PPO clip	0.2
learning rate	1e-4
optimizer	Adam
recurrent data chunk length	10
gradient clip norm	10.0
network initialization	orthogonal
normalize advantage	true



(a) `hunt a cow` (b) `hunt a sheep` (c) `hunt a cow` (d) `hunt a sheep`

Fig. 13: (a)(b) Learning curves of PPO using the focal reward with different λ on two Minecraft tasks. (c)(d) Learning curves of PPO using the focal reward with different Gaussian variances on two Minecraft tasks.

F Baselines Implementation

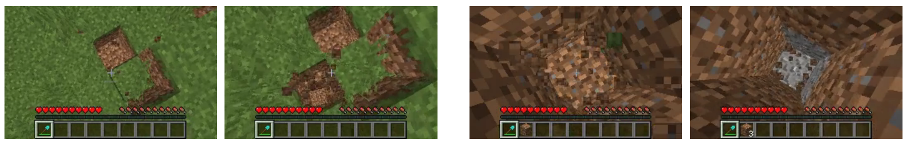
MineCLIP. We adopt the provided prompt templates in MineDojo to design task prompts for MineCLIP reward computation in single-task experiments. For hunting tasks, we use the prompt "`hunt a {animal} on plains with a diamond sword`". For additional harvest tasks in Appendix C.3, we use the prompts "`obtain milk from a cow in plains with an empty bucket`", "`shear a sheep in plains with shears`", and "`chop trees to obtain log with a golden axe`", respectively.

[7]. We use the released plains model[6] for evaluation. The goal is set to be the name of the target animal.

[6] `https://github.com/CraftJarvis/MC-Controller`

Table 17: Success rates (%) of STEVE-1 with different prompts.

Animals	"`kill`"	"`hunt`"	"`combat`"
`cow`	6	0	0
`sheep`	14	0	6
`pig`	9	0	4
`chicken`	6	0	7



(a) (b) (c) (d)

Fig. 14: (a)(b) Screenshots of the agent trained with MineCLIP reward. (c)(d) Screenshots of the agent trained with our focal reward targeting `hole`.

STEVE-1 [32]. We use the released model[7] for evaluation. However, STEVE-1 is designed for another simulator, MineRL [20], with a different action space from MineDojo. We build a wrapper to map STEVE-1's actions into the action space of MineDojo. As noted in the STEVE-1 paper, prompt engineering significantly impacts its performance. Therefore, we attempt three templates for the hunting domain tasks, including "`kill a {animal}`", "`hunt a {animal}`", and "`combat a {animal}`". As shown in Table 17, "`kill a {animal}`" achieves the highest performance and STEVE-1 cannot understand the original instruction "`hunt a {animal}`" at all. Consequently, we use "`kill a {animal}`" as prompts given to STEVE-1 for the experiments in the main text. For tasks in the harvest domain, we use prompts "`milk a cow`", "`shear a sheep`", "`break a flower`", "`break leaves`", "`collect water`", "`shear a mushroom`", "`collect sand`", and "`collect dirt`", respectively. The verbs `break` and `collect` are selected by referring to the prompts provided in the STEVE-1 paper. "`Milk a cow`", "`shear a sheep`", and "`shear a mushroom cow`" follow original instructions, as we find that "`collect {milk/wool/mushroom}`" does not work.

G Creative Tasks

For `dig a hole`, the agent spawns with a diamond shovel; for `lay the carpet`, the agent spawns with 64 carpets. For each task, we train an agent with MineCLIP reward and an agent with our focal reward. The prompts used to calculate MineCLIP reward are "`dig a hole`" and "`put carpets on the floor`", respectively. We run the trained models in the environment and record the agent's

[7] `https://github.com/Shalev-Lifshitz/STEVE-1`

depth and the number of placed carpets, averaged on 10 episodes. As illustrated in Figure 5b, the agents trained with MineCLIP reward and our focal reward targeting `carpet` show the same behavior pattern that keeps laying carpets. However, on `dig a hole`, the two agents learn different behaviors, as shown in Figure 5a: the agent trained with our focal reward targeting `hole` keeps getting deeper, while the elevation of the one trained with MineCLIP reward does not change too much. By examining their trajectories in the environment, we find that the agent trained with MineCLIP reward tends to dig one block and then stand beside this shallow hole and look at it, as shown in Figures 14a and 14b. In contrast, the agent trained with our focal reward stands inside the dug hole and continuously digs downwards, as shown in Figures 14c and 14d. Both behavior patterns are consistent with the description of "`dig a hole`" and can be considered reasonable.