




## A Conan Playground


 **Conan**'s playground is a computationally efficient 2D open-world environment with diverse items and rich tasks. The most distinctive feature of  **Conan**'s playground over the original Crafter environment is that agents in  **Conan** leave diverse traces when interacting with the environment. These traces serve as the foundation for abductive reasoning; the *detective* has to effectively connect the traces to figure out what the *vandal* has done.

### A.1 Items and Traces

**Land** Based on Crafter, there are three types of terrains that agents can walk on: *sand*, *grass*, and *path*. *Sand* and *grass* are soft surfaces where agents leave directional footprints after walking on them (see Fig. A1 first 2 rows in Columns 2 and 3 for examples). If a grid is left with more than one footprint, the footprints will become melded (Fig. A1 Column 4 in first 2 rows). Agents' actions will also leave traces on the terrain, *e.g.*, water on the ground (Fig. A1 Column 5 first 2 rows). If an agent gets injured, blood will be shed on the ground (Fig. A1 Column 6 first 2 rows).

**Creatures** There are four creatures in the playground: *plant*, *cow*, *zombie* and *skeleton*. *plant* grows from sapling to ripe plant. *Cow* randomly wander on the ground, whereas *zombie* and *skeleton* (monsters in general) will target agents in sight: *zombie* chases agents and *skeleton* shoots *arrow* at agents. Agents can fight with creatures and kill them. These actions will leave monster bodies on the ground.

**Tools** Agents can make tools on the *table*. There are 7 tools in total: *bucket*, *wood\_sword*, *wood\_pickaxe*, *stone\_sword*, *stone\_pickaxe*, *iron\_sword*, and *iron\_pickaxe*. These tools can be made using different materials and used for certain tasks. Both swords and pickaxes can be used to fight with creatures, but only pickaxes can be used in mining. Buckets can be used to collect water and lava.

**Actions**  **Conan**'s playground enables agents to interact with objects, non-playable characters, and even other agents in the playground. Agents can cut *tree* to get *apple* and *wood*, as well as collect *sapling* and grow *plant* (Fig. A1 Row 3). They can also mine with different tools to get *stone*, *coal*, *iron*, and *diamond*. Using these materials, agents can make *bed* for sleep, *furnace* for keeping monsters away and grilling food, *table* for making tools, *etc.* Of note, these items should be placed in an empty grid to use and they can be destroyed by monsters.

### A.2 Achievements and Tasks


There are 60 tasks and 39 achievements in  **Conan**'s playground. We list all achievements in Tab. A1. Tasks are composed achievements. We select 60 nontrivial and meaningful tasks from all compositions in  **Conan** as the final task set.

Table A1: Achievements in  **Conan**.

Type	Achievements			
Survive	drink_water	eat_apple	eat_beef	eat_steak
	sleep	sleep_on_bed	wake_up	eat_grilled_apple
Collect	drink_water_from_bucket	eat_plant		
	collect_wood	collect_apple	collect_water	collect_stone
	collect_iron	collect_diamond	collect_beef	collect_coal
Make	collect_water	collect_lava	collect_sapling	collect_plant
	make_steak	make_grilled_apple	make_bucket	make_fence
	make_wood_sword	make_wood_pickaxe	make_stone_sword	make_stone_pickaxe
	make_iron_sword	make_iron_pickaxe	place_table	place_bed
Defeat	place_furnace	place_plant		
	defeat_cow	defeat_zombie	defeat_skeleton	

### A.3 Observation and Action


 **Conan** offers both pixel representation and symbolic representation for training agents. For pixel representation, the environment returns a  $900 \times 900$  RGB image each time step for the *detective*'s



Figure A1: Items and related traces in  Conan.

$9 \times 9$  local view. For symbolic representation, the environment returns a  $9 \times 9$  tensor, with each entry an index representing one of 50 grid types, covering materials, resources, objects, creatures, and *etc.* The agent is always at the center of the observation.

🔍 **Conan** affords a larger action space. See [Tab. A2](#) for a detailed list of actions.

Table A2: Actions in 🔍 **Conan**.

Action	Details
Noop	Do nothing.
Move Left	Move left if the grid is walkable.
Move Right	Move right if the grid is walkable.
Move Up	Move up if the grid is walkable.
Move Down	Move down if the grid is walkable.
Do	Collect materials or fight with monsters. Use tools if possible.
Sleep	Sleep to restore energy. Sleep on bed can restore energy faster;
Place Stone	Place a stone if the grid is not occupied. Should have a stone.
Place Table	Place a table if the grid is not occupied. Should have a table.
Place Furnace	Place a furnace if the grid is not occupied. Should have furnace.
Place Plant	Place a plant if the grid is grass. Should have sapling.
Place Bed	Place a bed if the grid is not occupied. Should have bed.
Make Wood Pickaxe	Nearby table. Should have wood.
Make Stone Pickaxe	Nearby table. Should have wood, stone.
Make Iron Pickaxe	Nearby table, furnace. Should have wood, coal, iron.
Make Wood Sword	Nearby table. Should have wood.
Make Stone Sword	Nearby table. Should have wood, stone.
Make Iron Sword	Nearby table, furnace. Should have wood, coal, iron.
Make Bucket	Nearby table. Should have wood, stone.
Make Steak	Nearby table, furnace. Should have beef.
Eat Apple	Restore 2 health. Should have apple.
Eat Beef	Restore 4 health. Should have beef.
Eat Steak	Restore 6 health. Should have steak.
Collect Water	Collect water to bucket. Should have empty bucket.
Collect Lava	Collect lava to bucket. Should have empty bucket.
Drink	Drink water. Drink water from water bucket if not near the water.

## B 🔍 Conan Questions

### B.1 Question Generation

Questions in 🔍 **Conan** are generated based on *vandal*'s task-finishing process. To generate a question, (1) we initialize a playground and put the *vandal* in it; (2) the *vandal* is randomly assigned a task; (3) the *vandal* tries to finish the task with the help of the pre-build parser and planner, and generates logs along the way; (4) a question is generated based on a certain part of the log. We randomly select a template from the template pool and fill placeholders with related objects in it. The answer is also parsed from the log. Other choices are sampled based on the question and the context to avoid unrelated choices that can be easily excluded.

### B.2 Question Templates

[Tab. A3](#) lists all the templates we use for generating questions.

### B.3 Dataset Statistics

See [Tab. A4](#) and [Tab. A5](#) for details.

## C Explorer

The Explorer in the *detective* is an RL agent. The agent receives an observation of a  $[64, 64, 2]$  tensor. This tensor combines the  $9 \times 9$  symbolic local view of the *detective* and a  $64 \times 64$  question mask.

Table A3: Question templates in `Q Conan`. `[]` is the placeholder.

Type	Templates	
Intent	What was the <i>vandal</i> 's objective in these area?	What was the <i>vandal</i> 's current intent?
	What did the <i>vandal</i> do after this step?	What did the <i>vandal</i> do before this step?
	What did the <i>vandal</i> make on this table?	Why did the <i>vandal</i> make this table?
	What item did the <i>vandal</i> most likely craft using the table?	Why did the <i>vandal</i> make the <code>[]</code> ?
	What action did the <code>[]</code> perform immediately?	What was the <code>[]</code> used for?
	What did the <i>vandal</i> make on this furnace?	Why did the <i>vandal</i> make this furnace?
	What item did the <i>vandal</i> most likely craft using the furnace?	Why was tree cut?
	What was the intended use for the wood?	How was the tree cut?
	What was the purpose of mining <code>[]</code> ?	Why was the <code>[]</code> mined?
	What was the intended use for the <code>[]</code> ?	How did the <i>vandal</i> defeat the <code>[]</code> ?
Goal	What did the <i>vandal</i> use to defeat the <code>[]</code> ?	Why did the <i>vandal</i> defeat the <code>[]</code> ?
	What was the <i>vandal</i> 's final goal?	What was this <i>vandal</i> trying to achieve?
Survival	What did the <i>vandal</i> want to achieve?	
	What was the <i>vandal</i> 's survival intent for doing <code>[]</code> ?	why did the <i>vandal</i> collect/make <code>[]</code> ?
Survival	What was the <i>vandal</i> 's goal for survival currently?	Did the <i>vandal</i> die? Why?
	Why did the <i>vandal</i> die during the task?	How did the <i>vandal</i> die?
	What was the <i>vandal</i> trying to do when died?	What can the <i>vandal</i> do to avoid death?
	what helped keep the <i>vandal</i> away from hungry?	what food did the <i>vandal</i> eat?

Table A4: Dataset split and choice distribution.

Category	Train	Test	Val	Choice A	Choice B	Choice C	Choice D
Intent	71162	9152	8822	24.99%	25.20%	24.89%	24.93%
Goal	8000	1000	1000	24.89%	25.08%	24.87%	25.16%
Suvival	7365	1560	1596	25.13%	24.95%	24.95%	24.97%

Table A5: Task distribution.

Task Percentage	get_drink 2.47	defeat_cow 8.49	get_apple 2.52	make_stone_pickaxe 2.87	place_bed 8.44	place_furnace 8.23
get_lava 2.72	defeat_skeleton 8.7	make_iron_sword 2.64	get_coal 2.42	get_beef 2.7	get_diamond 2.39	get_stone 2.67
make_bucket 3.11	get_iron 2.44	get_water 2.2	make_iron_pickaxe 2.95	make_bed 2.71	make_steak 2.81	make_wood_sword 2.53
defeat_zombie 7.8	make_stone_sword 2.65	place_table 8.24	get_wood 2.67	make_wood_pickaxe 2.63		

The local view is zero-padded to  $64 \times 64$ . This ensures the agent knows its relative position on the map. Additionally, the mask is generated based on the question, with the area related to the question unmasked. The mask serves as the goal of the exploration policy.

All the RL baselines are trained for  $10^8$  steps. See more details below. Unless specified otherwise, parameters are set as default in Stable Baselines.

### C.1 Model Details

**DQN** The DQN baseline is trained using a  $\gamma$  value of 0.98, a  $\tau$  value of 1, a learning rate of 0.0001, a buffer size of  $10^7$ , and a batch size of 512. We leverage an MLP policy with two layers of 64 neurons each. The model is updated every 10 steps.

**TRPO** The TRPO baseline updates its policy with a special KL-divergence constraint on the distance between the new and old policies. We also leverage an MLP policy for TRPO, where the same multi-layer perceptron is used for both policy and value prediction.

**RecurrentPPO** The ReucrrntPPO baseline uses long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) as the recurrent policy. The LSTM layers' weights are initialized with standard Gaussian. We reset LSTM states at the end of the episode. The LSTMs for both the actor and the critic have the same architecture, with two LSTM layers of 256 neurons each.

## C.2 Training Details

Explorers are firstly trained on long-horizon tasks as explained in the main text. These long-horizon tasks include “get diamond,” “get lava,” “get water,” “make iron sword,” “make iron pickaxe” and “eat steak.” These tasks can be further broken down into over 20 subtasks and have an average episode length of more than 200 steps. We generate 10,000 unique scenes with traces given these tasks and train explorers on them for  $10^8$  steps. Then the explores are fine-tuned on all tasks in **Conan** for  $10^7$  steps.

We also show the frame rate per second (FPS) for different RL baselines during training in Fig. A2. As can be seen from the figure, DQN exhibits the highest training efficiency, reaching an FPS exceeding 3000. TRPO maintains a stable FPS of 2000. On the contrary, RecurrentPPO operates significantly slower, requiring over 96 hours to complete training with 128 subproc environments, whereas TRPO accomplishes the task in just 14 hours.

## D VL Reasoning

In this section, we describe the experimental details for the Vision-Language (VL) models used in the paper.

### D.1 Model Details

**Vanilla-Trans** For Vanilla-Trans, the visual features together with the text features are concatenated in the format of [frame\_1, frame\_2, ..., frame\_n, question, choice\_1, choice\_2, ..., choice\_4]. Visual features, if from the symbolic observation, are directly passed into the model. Otherwise, we utilize CLIP’s pre-trained image encoder (ViT-B/16) to extract features from pixel input. Text features are calculated using the text encoder of CLIP. These input features are then passed through a 6-layer Transformer model with an MLP head for classification.

**FrozenBiLM** We adopt the cross-modal FrozenBiLM for **Conan**, drawing inspiration from models used in Multiple-choice VideoQA benchmarks such as How2QA (Li et al., 2020) and TVQA (Lei et al., 2018)<sup>1</sup>. **Conan** can be formulated as a multiple-choice VideoQA problem given the fixed explorer. We concatenate all of the observation frames as the video input. The questions and choices are converted into the following format: [“{question} Is it {choice\_1}?”, ..., “{question} Is it {choice\_4}?”]. We then evaluate the probabilities of the model producing “Yes” and “No”. The visual features are processed in the same way as in Vanilla-Trans and then forwarded for visual-text projection. We utilize BERT-Large and DeBERTa as our frozen language backbones in this work; however, other general language models are applicable as well.

**Flamingo-Mini** Our Flamingo-Mini baseline is based on an open-source implementation of the Flamingo model<sup>2</sup>, as the original Flamingo model’s pre-trained weights are not accessible. Flamingo-Mini is built upon OPT-125M and CLIP pre-trained ViT-L/14 model. We also formulate **Conan** as a multiple-choice problem for Flamingo-Mini. The questions and choices are converted into the following format: [“Question: {question} Answer: {choice\_1}”, ..., “Question: {question} Answer: {choice\_4}”]. Each question-choice pair is fed into the model and then a binary classifier head is used on Flamingo’s last layer output to predict the final answer.

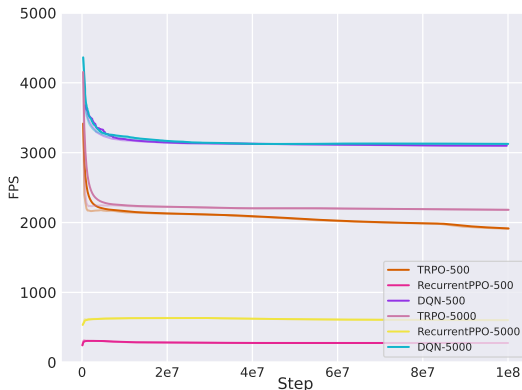


Figure A2: **Frame rate per second (FPS) curves of several RL explorers in training.** Results show that DQN and TRPO are significantly faster than RecurrentPPO.

<sup>1</sup><https://github.com/antoyang/FrozenBiLM>

<sup>2</sup><https://github.com/lucidrains/flamingo-pytorch>

## D.2 Training Details

Vanilla-Trans was trained for 100 epochs, with a batch size of 128. FrozenBiLM models were trained for 50 epochs, with a masking probability (for the MLM objective) of 0.15, a batch size of 32, a learning rate of  $3 \times 10^{-4}$ , a gradient clipping max norm of 0.1, and Adam as the optimizer ( $\beta_1 = 0.9, \beta_2 = 0.95, \epsilon = 1 \times 10^{-8}$ ). Flamingo-Mini was trained for 100 epochs, with a learning rate of  $5 \times 10^{-5}$ , a batch size of 8, and also Adam as the optimizer ( $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1 \times 10^{-8}$ ).

## E Additional Experiments

### E.1 Negative Control Baselines


We compare our VL reasoning results on the trained explorers with those on empty visual inputs as a negative control baseline. The results are shown in [Tab. A6](#).

Table A6: VL Reasoning models’ performance on explorers compared with empty visual inputs.

	Vanilla-Trans	F-BiLM-BERT	F-BiLM-DeBERTa	Flamingo-mini
Empty visual inputs	26.4	25.5	25.9	22.9
TRPO explorer	25.0	44.4	43.1	43.3
Ideal explorer	78.4	59.5	71.8	47.8

The results show that using empty visual inputs yields random performance across all settings. Besides, it also shows that the training QA pairs are unbiased. The TRPO explorer achieves higher performance, which suggests that the exploration strategy learned by TRPO helps gather some informative evidence for the reasoning process. The Ideal explorer is an oracle-like exploration policy that has access to perfect trace evidence and temporal information. It provides the most comprehensive information about the environment. This highlights the importance of effective exploration in improving reasoning performance. However, it does not mean that reasoning is less important, as even with the Ideal explorer, the model still could not achieve satisfactory performance. Based on all results, collecting informative evidence seems to be more important in the overall objective.

## F Abduction from Deduction (AfD)

As mentioned in [Sec. 4.4](#), we adopt a data-driven strategy to learn a model of  $P(g | S)$  and simultaneously answer the questions. To be more specific, we train the *detective* agent self-supervisedly. The *detective* is randomly assigned with one of all possible tasks. It then finishes the task by following the action policy  $\pi(\cdot)$ . Note that we assume the *detective*’s  $\pi(\cdot)$  is the same as the *vandal*’s in order to best implement the idea of AfD. Based on the task execution process, questions are generated. Since our ultimate goal is to have our models answer  Conan’s questions, we do not explicitly construct  $P(g | S)$ , but rather consider the question-answer process as the  $g$ . We then train  $P(g | S)$ , where  $S$  is the *detective*’s observation during the task execution, and the label can be derived from the assigned tasks together with the  $\pi(\cdot)$ .

Besides  $P(g | S)$ , we still need to learn a model of  $P(S | O)$ , which, intuitively, can be understood as inferring the true state of the environment from partial observation. In our experiment, we tried two ways to model  $P(S | O)$ . One approach is to directly train a model using multi-frame observations to predict the states. We employed a UNet (Ronneberger et al., 2015) and a multi-layer CNN as the network. However, this method did not work effectively. Reasoning based on the reconstructed states only achieved performance at a random level. The second approach, which was finally used to report performance, aligned the hidden feature spaces from true states and observations. When training  $P(g | S)$ , we added a head before the VL models, converting the input  $S$  into a 4096-dimensional vector. Then we trained a head on  $O$  with the same structure, minimizing the difference between features from  $O$  and features from  $S$ .

## G Conan Task Demo



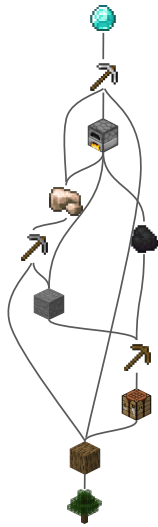


Figure A3: The task structure of “get diamond”.

To better illustrate the core components in **Conan**, We take the playground shown in Fig. 1 as an example. In this scenario, the assigned task is “get diamond” (Fig. A3 shows the task dependency). As shown in Fig. A4, once the vandal completes the task, it leaves behind traces in the playground. The vandal ends at the bottom of the figure. The detective then enters the playground, starting at the beginning of the traces. In this case, traces encompass footprints and remnants left after certain actions. Note that footprints cannot be left on sand or stone, and different footprints may overlap. The vandal will collect objects crafted on a table, making them invisible.

Let’s suppose the detective’s exploration begins by following footprints (note the context window size is  $9 \times 9$ ).

Firstly we can see some cut trees. As the footprints are not seriously overlapped and mostly one-directional, we can deduce the vandal did not return. After seeing the tool-making table, with the only resources being wood, we could say that the vandal could only make wooden tools, not stone swords or iron pickaxes, further restricting possible actions the vandal took.

Note that this is already critical reasoning in **Conan**.

Moving on, we note that footprints become missing on the sand surface. However, we note broken stones and coals. Therefore, the wooden tool to break stones and coals shall be a wooden pickaxe. So the agent should have made a wooden pickaxe on the table earlier. Despite the fact that the tool has been collected, we could still figure that out.

Following the reemerged footprints, we note blood and a zombie body on the ground, suggesting the vandal should have had a fight.

Searching on, we find the broken diamond. As an iron pickaxe is the only tool to collect diamonds. The vandal must have built an iron pickaxe with iron and coal in the furnace. With no other footprints around, we can safely conclude our search.

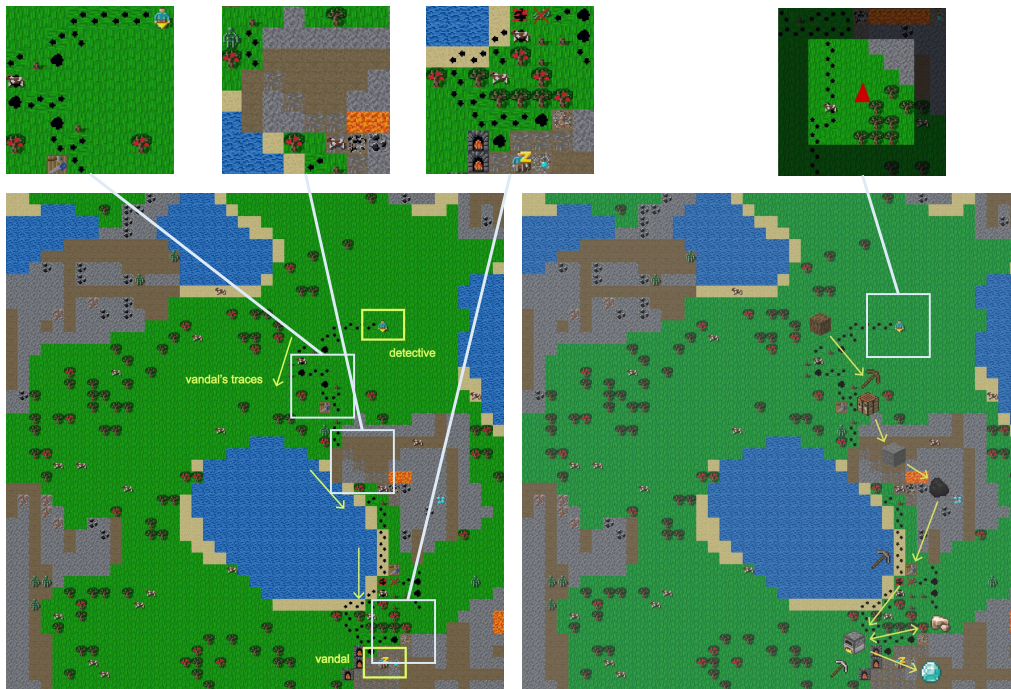


Figure A4: A demonstration of core components in **Conan**. We show how a *detective* can do reasoning based on the task structure and traces left in the playground. Zoom in for more details.