

Contents

A Dataset details	1
A.1 WebUI	1
A.1.1. Web page collection and preprocessing	1
A.1.2. System prompt p_{sys}	2
A.1.3. Evaluation	2
A.2 Sokoban	4
A.2.1. Level generation	4
A.2.2. System prompt p_{sys}	4
A.2.3. Evaluation	4
A.3 Football	5
A.3.1. Initial scene generation.	5
A.3.2. System prompt p_{sys}	6
A.3.3. Evaluation	6
A.3.4. Auto-rendering algorithm.	7
A.4 Environment selection	9
A.5 Human labeler details	11
B LMM-as-Agents	12
C More Results	13
C.1. MageBench results	13
C.2. Agent memory	13
C.3. Error Analysis	14
C.4. Failure of WebUI online	14
D Benchmark generalization capability	14
D.1. Robotics and embodied AI	15
D.1.1. Overall results	16
D.1.2. Visualization	16
D.2. Structured Visual Generation	16
E RL training details	20
F. Future researches and limitations	21
G Examples	21
G.1. WebUI	21
G.1.1. Labeled target webpage	21
G.1.2. Interactions	21
G.1.3. Task descriptions	21
G.1.4. Task prompts p_{cot}, p_{io}	21
G.1.5. Output examples	25
G.2. Sokoban	27
G.2.1. Task prompts p_{cot}, p_{io}	27
G.2.2. Output examples	27
G.3. Football	28
G.3.1. Task prompts p_{cot}, p_{io}	28
G.3.2. Output examples	28

A. Dataset details

A.1. WebUI

A.1.1. Web page collection and preprocessing

We searched and collected a large number of open-source, minimalist web design examples from GitHub. These small projects are often educational examples, and we filtered them to include only web projects composed of a small number of HTML, JavaScript, and CSS files. We ensure that the entire process of data collection, processing, application to our benchmark, and re-release as open source fully complies with the original open-source licenses of the GitHub repositories.

We then pre-process the projects with the following steps:

- **Resource Download:** First, we download images and other resources from the webpage to a local folder. We then rename these files and correspondingly update the image address names in the webpage.
- **Content Adjustment:** We made several content adjustments to the web pages, including layout and attributes. Overly long web pages were split into multiple pages. Additionally, some web page content was modified to prevent existing models from having encountered these examples during data scraping on GitHub for training purposes.
- **Atomic Element Annotation:** In the website’s HTML files (including parts of JavaScript that contain HTML code), we identify all atomic elements (as defined in the main part). For each corresponding HTML tag, we add two attributes: *data-filter-by* and *data-evalby*. The *data-filter-by* attribute contains a single HTML or CSS property (such as the text property of a paragraph or the URL property of an image tag). This attribute, present in only a few elements, highlights the characteristics of the atomic element and is used to optimize the precision of atomic element matching. The *data-evalby* attribute consists of multiple CSS properties (such as font, color, background, etc.). It displays the scoring criteria of the atomic element, and the evaluation of the element is based on the similarity of the CSS properties listed in *data-evalby*. You can find an example of the annotated origin webpage in [G.1.1](#).
- **Web Interaction Identification:** We identify and design meaningful web interactions (such as scrolling, clicking buttons, entering content, etc.) for each webpage. These interactions are implemented as Python functions using Google Chrome and the Selenium library. Additionally, we capture and store screenshots of the webpage after each interaction. An example interaction python code can be found here [G.1.2](#).
- **Task Description:** We write a website reconstruction task description in markdown format of interleaved

image-text based on the information of the website, which will include: 1. The function of the website. 2. Screenshots of the web page after the initial and each interaction. 3. The resources used in the website construction and the places where they are used (including image url, long texts, external CSS links, etc.). 4. Specify the class/id/-name of the elements that can be interacted with, This will be used for automated interaction testing. An example of task description can be found in [G.1.3](#)

A.1.2. System prompt p_{sys}

The system prompt for this environment is shown as follow:

```
Your job is to re-implement a Web page
UI given target description and screen
shot, by coding with `index.html`, and
probably `style.css` and `script.js`
if needed.
You will be provided a task description
and available actions. Actions including
writing to files and interact with
your generated webpage.
Here are the action ID and explanation:
```

```
'write_html' - Input: str, Write the input
str into 'index.html', if called second
times, it will override the file.
```

```
'write_javascript' - Input: str, Write the
input str into 'script.js', if called
second times, it will override the file.
```

```
'write_css' - Input: str, Write the input
str into 'style.css', if called second
times, it will override the file.
```

A.1.3. Evaluation

The evaluation process of WebUI is illustrated in Figure 1. In general, if there is an error in webpage parsing or initial page rendering, we directly assign a score of 0 to that webpage. When the webpage renders successfully, we treat the original page and each subsequent page after an interaction as independent static webpages for evaluation (excluding duplicated parts). For each static webpage, we perform two processes: atomic element matching and similarity calculation of the CSS properties of matched elements. Below, we will detail each part step by step.

Atomic Tags Matching. From a top-down perspective, atomic element matching is implemented using the Hungarian algorithm to establish a matching between the set

$$\mathbb{E}_{gen} = \{e : \text{all elements in the generated webpage}\}$$

and the set

$$\mathbb{E}_{gt} = \{e : \text{all atomic elements in the GT webpage}\}.$$

To do this, we need to assign a score $S(e_i, e_j)$ where $e_i \in \mathbb{E}_{gen}, e_j \in \mathbb{E}_{gt}$ for the Hungarian algorithm to maximize. In our implementation, the definition of $S : \mathbb{E}_{gen} \times \mathbb{E}_{gt} \rightarrow \mathbb{R}$ is as follows:

$$S(e_i \in \mathbb{E}_{gen}, e_j \in \mathbb{E}_{gt}) := GIoU(e_i.bbox, e_j.bbox) + e_j.filter(e_i) - \epsilon |e_i.children - e_j.children| \quad (1)$$

GIoU is the standard generalized Intersection over Union that well-known in object detection. The GIoU value ranges from $[-1, 1]$ and is used to describe the similarity of the positions of the bounding boxes of two elements within a fixed-size frame. There is also a filter function to help better matching accuracy:

$$e_j.filter(e_i) = \begin{cases} -1 & \text{if } L(e_i.attr, e_j.attr) < \lambda_{attr} \\ 0 & \text{else} \end{cases},$$

where $attr$ is the pre-defined *data-filter-by* attribute of the atomic element of the ground truth webpage, and L is a general attribute similarity scoring function which will be introduced later. λ_{attr} is a pre-defined fixed value for a kind of attribute. The attribute may be a CSS attribute, or a html attribute.

For example, let e_j be a text tag and its *data-filter-by* attribute is "text", and e_j is:

```
<!--this is ej-->
<span>example text tags </span>
```

Now suppose we have a generated webpage:

```
<!--this is ek-->
<span>example not match </span>
```

```
<div> <!--div is et-->
  <!--this is ei-->
  <h1>example text tags </h1>
</div>
<!--this is ew-->
<span>example not match 1 </span>
```

In this example, because:

$$L(e_k.text, e_j.text) = TIoU(e_k.text, e_j.text) < \lambda_{attr},$$

where $TIoU$ is term IoU metric for text. So $e_j.filter(e_k) = -1$, which means wherever the element e_k is incorrectly set, even its bounding box position coincide, it will not match with e_j . This filtered out a large amount of irrelevant element during the matching process.

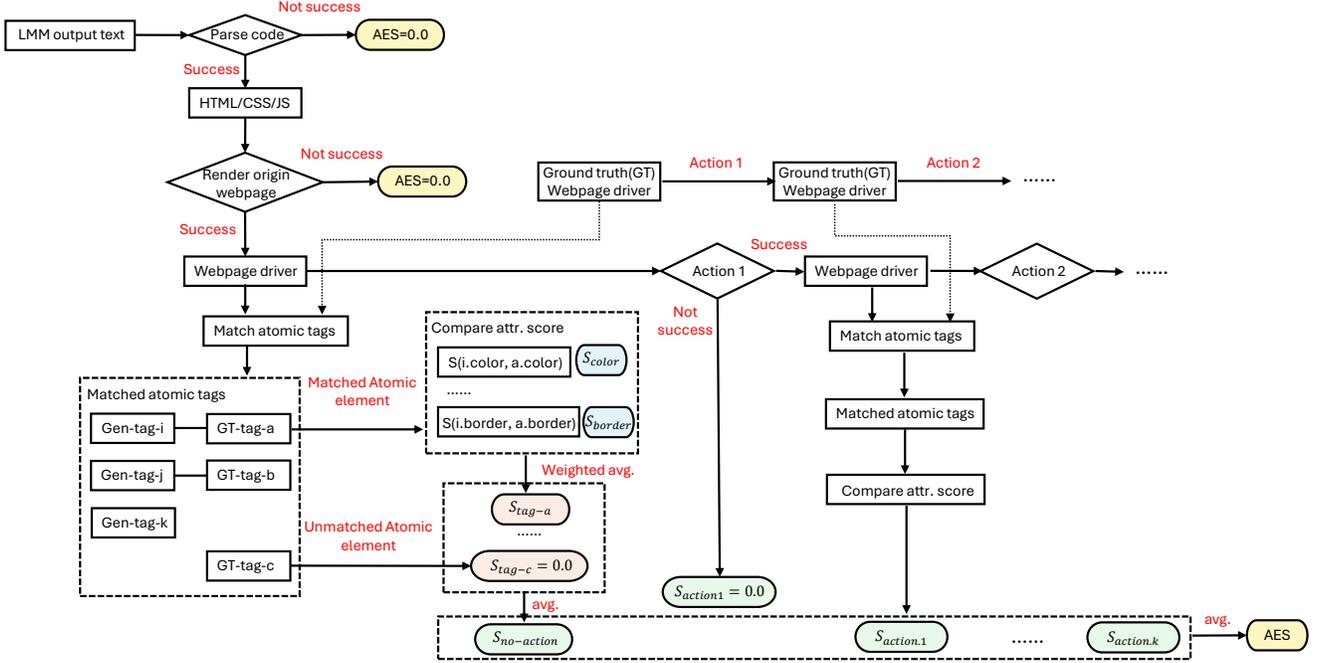


Figure 1. The complete evaluation pipeline of WebUI.

The last term of equation 1 is used to avoid this bug: using the same example above, you will find tag e_t (the div container) has the same bounding box and text attribute with e_i (the inner h1 tag). This is a property of html attribute. By adding the last term with a small punish coefficient $\epsilon = 10^{-3}$, e_j will match e_i instead of e_t .

Attribute Similarity Score. We need to implement a $L : \mathcal{A} \times \mathcal{A} \rightarrow [0, 1]$, where \mathcal{A} is all html and css attributes. We actually implement this function according to the attribute types. For text attribute, we just use the term IoU. For continuous variables, including *border-width*, *border-radius*, *font-size*, *height*, *width*, *letter-spacing* and etc, we just use the relative error:

$$L(e_j.\text{attr}, e_i.\text{attr}) = \frac{|e_j.\text{attr} - e_i.\text{attr}|}{e_j.\text{attr}}$$

For discrete variables with few choices, including *background-image*, *background-repeat*, *src*, *type*, *border-style*, *outline-style*, *font-family*, *font-style*, *tab-size*, *display* and etc, we use:

$$L(e_j.\text{attr}, e_i.\text{attr}) = \mathbb{1}(e_j.\text{attr} == e_i.\text{attr}).$$

For color attribute, including *color*, *background-color*,

border-color, *outline-color* and etc, we use:

$$L(e_j.\text{attr}, e_i.\text{attr}) = \frac{1}{3} \sum_{c \in \{r,g,b\}} \frac{1}{256} |e_j.\text{attr}.c - e_i.\text{attr}.c|.$$

Hyper-parameters Searching. As mentioned before, the final AES score can be written as the average of each static webpage score (after certain actions, including scrolling, click):

$$S_{AES} = \sum_{act} S_{act},$$

and S_{act} is weighted averaged with pre-defined attribute weight and element space:

$$S_{act} = \sum_{e_i, e_j \in \text{matched}} \left(\frac{\sum_{at} L(e_j.at, e_i.at) \alpha_{at}}{\sum_{at} \alpha_{at}} \right) (e_j.\text{space})^\beta.$$

This means we still need a series of pre-defined hyper-parameters $\{\alpha\}$ and β , which are difficult to set manually, because it need to consider not only the importance of the attribute, but also the range of variation of the property. Instead, we first generate a large amount of webpages using varies models, to re-implement a certain webpage of our benchmark. Then human annotators rank those model

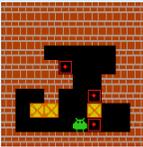
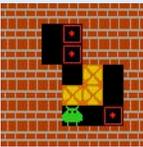
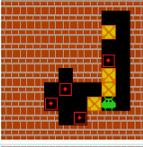
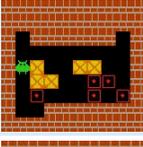
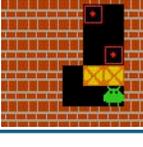
Example	Id	#Samp.	Example	Id	#Samp.
	Grid	#Boxes		Grid	#Boxes
	Steps			Steps	
	v0	15		small-v1	20
	v1	24		large-v0	14
	v2	22		large-v1	18
	small-v0	50		huge-v0	19

Figure 2. Details of different difficulties of the environments. ‘#Samp.’ refer to the number of the environments and ‘Steps’ for the averaged minimum number of steps to fulfill the task.

implementations. We then used a CUDA-accelerated GPU version of Particle Swarm Algorithm (PSO) to search the parameters to align with the human preference. As a result, the evaluation results with searched parameters agree 93% of the human preferences, from 68% with random parameters.

A.2. Sokoban

A.2.1. Level generation

Based on the environment configurations provided by deepmind, we generated environments of varying difficulty levels. By considering the grid size of the environment and the number of boxes, we categorized all environments into eight difficulty levels, resulting in a total of 182 environments. Examples and detailed information for each difficulty level are shown in Fig. 2. We employed a pruned breadth-first search (BFS) algorithm to determine the minimum number of steps required to complete each environment, ensuring that all generated environments can be completed within fifty steps. This constraint is aimed at reducing the computational budget required for model invocation.

A.2.2. System prompt p_{sys}

The system prompt for this environment is shown as follow:

You’re going to play a game of Sokoban,

where the goal is to manipulate the green character to push the yellow box into the target area (an area with a red dot in the center).

You’ll use text commands to manipulate your character and restart the game, and here are the possible commands and corresponding meanings:

- ‘Left’: The action character moves to the left, and if there is a box on the left, pushes the box to the left.
- ‘Right’: The action character moves to the right, and if there is a box on the right, pushes the box to the right.
- ‘Up’: Character moves up, and if there is a box above, pushes the box up.
- ‘Down’: The action character moves down, and if there is a box underneath, pushes the box down.

Note:

1. It is not possible to push two boxes side by side at the same time from the side-by-side direction.
2. The red brick pattern is the wall. The game can go into a lose-lose state, for example if you push the box into a corner where the corner is not the target area. In this case, you should restart the game.

A.2.3. Evaluation

Our evaluation metrics are adapted from those proposed in deepmind’s paper. Specifically, for a given time step t (assuming the task is not yet fully completed at this time), we allow the agent to take an action and define the reward at this time step as follows:

$$R^{(t)} = \begin{cases} +4.5 & \text{pushed box to target} \\ -5.5 & \text{pushed box out of target} \\ +54.5 & \text{task done} \\ -0.5 & \text{otherwise} \end{cases},$$

which is exactly the same as the reward defined in their work except that we scale the value by $5\times$. At the same time, we do not want the model’s output length (under the Global setting) or the number of steps set (under the On-line setting) to affect the evaluation results. Therefore, we

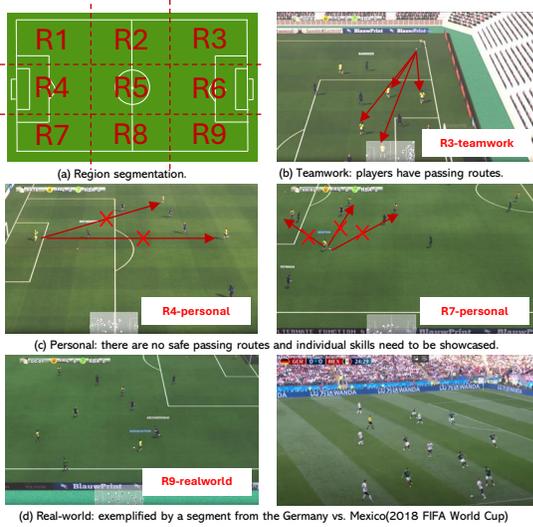


Figure 3. An overview of initial scene setting in the football environment.

choose the metric of the historically best cumulative reward value, rather than using the cumulative reward value directly. Secondly, due to the inherent differences in the difficulty levels of various environments, the variances across different environments also differ. Directly calculating the mean of the rewards may cause environments with larger variances to dominate the final results. To address this issue, we subtract the reward value corresponding to the optimal trajectory (as provided by BFS) from the final evaluation of each environment. The final evaluation metric is shown as follow:

$$R = \max_{t=1}^T \left(\sum_{\tau=1}^t R(\tau) \right) - R_{best} + 100,$$

where R_{best} is the reward corresponding to the trajectory that achieves the task in the shortest possible steps, and T is the lesser of the number of steps the model takes to successfully complete the task and 50. It is worth noting that, as mentioned earlier, all environments can be completed within 50 steps.

A.3. Football

A.3.1. Initial scene generation.

To enhance the diversity of tasks, we have designed different initial scenarios as levels. Each level operates under the same game rules but starts with different initial conditions (positions of players and the ball). To comprehensively demonstrate the model’s capability to handle various situations, our initial scenarios are divided into three major categories: Personal, Teamwork, and Real-world. The Personal category represents scenarios where the ball handler

does not have a clear passing route and needs to showcase individual skills to break through defenders, as illustrated in Figure 3(c). In contrast, the Teamwork category represents scenarios where the positioning of players allows for effective collaboration and coordination, as illustrated in Figure 3(a). To ensure the tasks are realistic, we have also extracted player positions from actual World Cup matches. This constitutes our Real-world category, as shown in Figure 3(d).

In addition to the aforementioned three major categories, we also evaluate the model’s ability to handle the ball in different areas of the soccer field. As shown in Figure 3(a), we have divided the soccer field into nine zones, and we will initialize various scenarios in each of these zones. In summary, we have three major categories and nine zones. For each category and zone, we will initialize four different scenarios, resulting in a total of $3 \times 9 \times 4 = 108$ unique scenarios. Table 1 presents the specific World Cup matches and times for the Real-world category across all zones. These moments, which occurred in actual games, will be featured in our scenarios.

Table 1. Details of 36 scenes in Real-world category.

Region	Game; Time
R1	1. Brazil vs. Belgium — 2018 FIFA World Cup; 69:32
	2. Germany vs. Mexico — 2018 FIFA World Cup; 82:11
	3. Brazil vs. Germany — 2014 FIFA World Cup; 12:19
	4. Brazil vs. Germany — 2014 FIFA World Cup; 26:55
R2	1. France vs. Croatia — 2018 FIFA World Cup Final; 46:53
	2. Belgium vs. Japan — 2018 FIFA World Cup; 4:01
	3. Germany vs. Mexico — 2018 FIFA World Cup; 53:08
	4. France vs. Argentina — 2018 FIFA World Cup; 29:28
R3	1. Brazil vs. Belgium — 2018 FIFA World Cup; 14:13
	2. Brazil vs. Belgium — 2018 FIFA World Cup; 24:36
	3. Germany vs. Mexico — 2018 FIFA World Cup; 13:11
	4. France vs. Argentina — 2018 FIFA World Cup; 47:25
R4	1. Germany vs. Mexico — 2018 FIFA World Cup; 62:58
	2. France vs. Argentina — 2018 FIFA World Cup; 18:06
	3. Portugal vs. Spain — 2018 FIFA World Cup; 82:02
	4. Brazil vs. Germany — 2014 FIFA World Cup; 26:23
R5	1. Brazil vs. Belgium — 2018 FIFA World Cup; 6:22
	2. France vs. Argentina — 2018 FIFA World Cup; 42:08
	3. France vs. Argentina — 2018 FIFA World Cup; 64:47
	4. Portugal vs. Spain — 2018 FIFA World Cup; 31:49
R6	1. Belgium vs. Japan — 2018 FIFA World Cup; 8:02
	2. Portugal vs. Spain — 2018 FIFA World Cup; 62:08
	3. Brazil vs. Germany — 2014 FIFA World Cup; 40:46
	4. Netherlands vs. Brazil — 2010 FIFA World Cup; 16:55
R7	1. Germany vs. Mexico — 2018 FIFA World Cup; 39:49
	2. France vs. Argentina — 2018 FIFA World Cup; 00:52
	3. Portugal vs. Spain — 2018 FIFA World Cup; 21:25
	4. Portugal vs. Spain — 2018 FIFA World Cup; 64:22
R8	1. France vs. Croatia — 2018 FIFA World Cup Final; 0:18
	2. Brazil vs. Belgium — 2018 FIFA World Cup; 85:34
	3. France vs. Argentina — 2018 FIFA World Cup; 61:53
	4. Portugal vs. Spain — 2018 FIFA World Cup; 18:43
R9	1. Belgium vs. Japan — 2018 FIFA World Cup; 20:53
	2. Germany vs. Mexico — 2018 FIFA World Cup; 24:29
	3. Portugal vs. Spain — 2018 FIFA World Cup; 26:31
	4. Brazil vs. Germany — 2014 FIFA World Cup; 01:51

A.3.2. System prompt p_{sys}

The system prompt of football environment is shown as follow:

You're going to play a football game. The character you control wears a yellow jersey with a blue name above his head.

Your goal is to attack the goal on the ****right side**** according to standard soccer rules. It's your teammates who wear the yellow jersey, and the opponents in the other colors.

You'll use text commands to manipulate your character. Here are the action ID and explanation:

- 'action_left', 'action_top_left', 'action_top', 'action_top_right', 'action_right', 'action_bottom_right', 'action_bottom', 'action_bottom_left': Run in the direction corresponding to the action ID's name.
- 'action_long_pass': perform a long pass to the player on your team. Player to pass the ball to is auto-determined based on the movement direction.
- 'action_high_pass': perform a high pass, similar to action_long_pass.
- 'action_short_pass': perform a short pass, similar to action_long_pass.
- 'action_shot': perform a shot, always in the direction of the opponent's goal.
- 'action_sprint', 'action_release_sprint': start and stop sprinting, sticky action. If sprinting, player moves faster, but has worse ball handling.
- 'action_dribble', 'action_release_dribble': start and stop dribbling, sticky action. Player moves slower, but it is harder to take over the ball from him.

A.3.3. Evaluation

The reward system is designed as follow:

$$R^{(t)} = \lambda_1 S_{move}^{(t)} + \lambda_2 S_{oppo}^{(t)} + \lambda_3 \delta_{scored}^{(t)} + \lambda_4 \delta_{stole}^{(t)} \frac{t}{T} + \lambda_5 \delta_{pass}^{(t)} S_{pass}^{(t)} + \lambda_6 \delta_{shot}^{(t)} S_{shot}^{(t)}, \quad (2)$$

where $\lambda_1 = 16, \lambda_2 = 20, \lambda_3 = 40, \lambda_4 = 20, \lambda_5 = 400, \lambda_6 = 100, T = 400$ are fixed numbers. δ is an event indicator:

$$\delta_{event}^{(t)} = \begin{cases} 1 & \text{if event happened at time step } t \\ 0 & \text{else} \end{cases}.$$

Now, lets introduce each term separately.

Advancement Reward $S_{move}^{(t)}$. This metric is used to reward the agent for advancing the ball towards the opponent's goal, which includes actions such as individual dribbling breakthroughs and passing. We only consider this metric when the ball is under the control of our team's players. Therefore, it can be expressed as:

$$S_{move}^{(t)} = \delta_{own-ball}^{(t)} (ball_x^{(t)} - ball_x^{(t')}),$$

where $ball_x$ is the x coordinate of ball and t' is the last time our team owned the ball. If one of our player is always controlling the ball, $t' = t - 1$. But if a player receive a ball at t from his teammate, then t' will be the frame that his teammate pass the ball.

Passing Opponents Reward $S_{oppo}^{(t)}$. This metric rewards the player for surpassing the opponent, incentivizing the ball to be positioned closer to the opponent's goal than the opponent's players:

$$S_{oppo}^{(t)} = \frac{1}{11} (Passed^{(t)} - Passed^{(t')}),$$

where t' is the same above and

$$Passed^{(\tau)} = \sum_{p \in opponents} \mathbb{1}[p_x^{(\tau)} < ball_x^{(\tau)}].$$

Scoring Reward and Ball Interception Penalty. If a goal is scored at this moment, the game will immediately end and a positive reward λ_3 will be received. If the ball is intercepted by the opponent (or goes out of bounds, or an offensive foul occurs), a reward linearly related to the moment of interception will be provided as $\lambda_4 \delta_{stole}^{(t)} \frac{t}{T} \in (0, \lambda_4]$. This means the later the player is intercepted, the larger the reward received. If the ball is not intercepted by the maximum time limit of $T = 400$ frames, the player will receive the maximum reward λ_4 .

Passing Ball Reward $S_{pass}^{(t)}$. We first note that $\delta_{pass}^{(t)}$ is the moment that the ball passing event ends (our team player

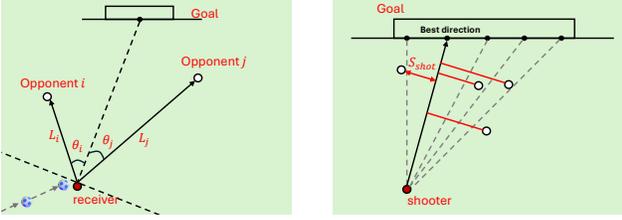


Figure 4. **Left:** Diagram illustrating the calculation of Passing Ball Reward. **Right:** Diagram illustrating the optimal shooting angle and opponent distance.

receives a ball), instead of start passing. We only consider successful passes, as unsuccessful passes have already been penalized in the fourth term. At the moment when a teammate successfully receives the pass, we will provide a reward based on the distance of the nearest opponent to the receiver and the level of threat posed by the opponent. This incentivizes the model to pass the ball to players who are in more open and advantageous positions for breakthroughs. It is defined as follow:

$$S_{pass}^{(t)} = \min_{i=1}^{11} \left[\frac{L_i}{\beta \max(\cos \theta_i, 0) + \epsilon} \right],$$

where $\beta = 10$ and $\epsilon = 1$. As shown in Figure 4 left, suppose the vector from the receiver to the opponent’s goal is \vec{a} . We define L_i as the straight-line distance from the i -th opponent player to the receiver, and θ_i as the angle between the vector from the receiver to this opponent and vector \vec{a} . We further interpret this metric as follows: First, we can easily observe the expression:

$$\frac{L}{\beta \max(\cos \theta, 0) + \epsilon} \in \left[\frac{L}{\beta + \epsilon}, \frac{L}{\epsilon} \right].$$

As L increases, meaning the opponent is farther away from the receiver, the reward increases, indicating that the threat from the opponent is reduced. Conversely, as $\max(\cos \theta, 0)$ increases, it means the opponent’s position is more obstructive to the player’s path towards the goal, representing a higher threat. In this case, the reward decreases.

Shot Rewards $S_{shot}^{(t)}$. Given that scoring a goal is extremely challenging and carries a significant degree of randomness in the current model, we instead assign a score to a shot based on the quality of the opportunity at the moment the shot is taken. As shown in Figure 4 right, we divide the goal into five target points for shooting. Assuming there are five shooting paths from the current player to these points, we first determine the optimal shooting path by calculating the distances of opponent players to these shooting paths. The optimal path is selected based on these distances, as indicated by the solid line in the Figure 4 right. We then define $S_{shot}^{(t)}$ as the shortest distance from these opponent players to the chosen shooting path.

A.3.4. Auto-rendering algorithm.

Since we set the rendering frame rate to 400 frames in the simulated football environment, completing a single scenario requires 400 API calls. This results in significant time and cost expenditures for evaluation. In fact, each time an action is input and a frame is rendered, the characters in the game only move a very small distance, which implies that there will be a large number of repetitive actions in between. Our idea is to employ certain algorithms to automatically repeat these actions, allowing players to make decisions at critical moments. We have identified two scenarios where we can automate rendering to reduce redundant calls: 1. Passing sequences. 2. Straight-line running.

Passing sequences. According to the rules of the rendering environment, when a character initiates a pass or a shot, any control during the ball’s time in the air is inherently ineffective, and the controlled player will switch during this process. Therefore, any control input during the entire duration when the ball is in the air is meaningless, and we will directly skip this period (corresponding to the process from $T = 10$ to $T = 13$ in the left panel of Figure 5). During this period, the player’s action is defaulted to using the built-in AI.

Straight-line running. When a player inputs the same directional movement for two consecutive frames, we enable auto-rendering for straight-line running. The maximum number of frames for automatic rendering is set to 10 frames. For each of these 10 frames, we execute the following algorithm: assuming the current moment is T , we pre-render 5 frames based on the current positions and directions of all players. If, during these pre-rendered 5 frames, the distance between an opponent and the ball carrier is less than a preset threshold δ , we will terminate the automatic rendering at moment T and return control to the player. Otherwise, we will continue rendering in the current direction and speed until moment $T + 1$.

The details of the algorithm are presented in Pseudocode 1. In Figure 5, the automatic rendering corresponds to the periods from $T = 1$ to $T = 6$ and from $T = 14$ to $T = 18$. It is worth noting that within the maximum 10 frames of automatic rendering, the player is defaulted to move in a certain direction, which might prevent the player from passing the ball or performing other actions during this interval. However, in the rendered game visuals, the duration of 10 frames may only correspond to one or two steps for the player, covering a real-world distance of about two to three meters, which is still very short. Additionally, the algorithm ensures that there are no nearby opponents causing interference when the automatic rendering ends. Therefore, the impact can be considered negligible.

We also conducted numerical experiments to demonstrate the effectiveness and correctness of our algorithm. As shown in Figure 6, we used an AI bot as the player. The line

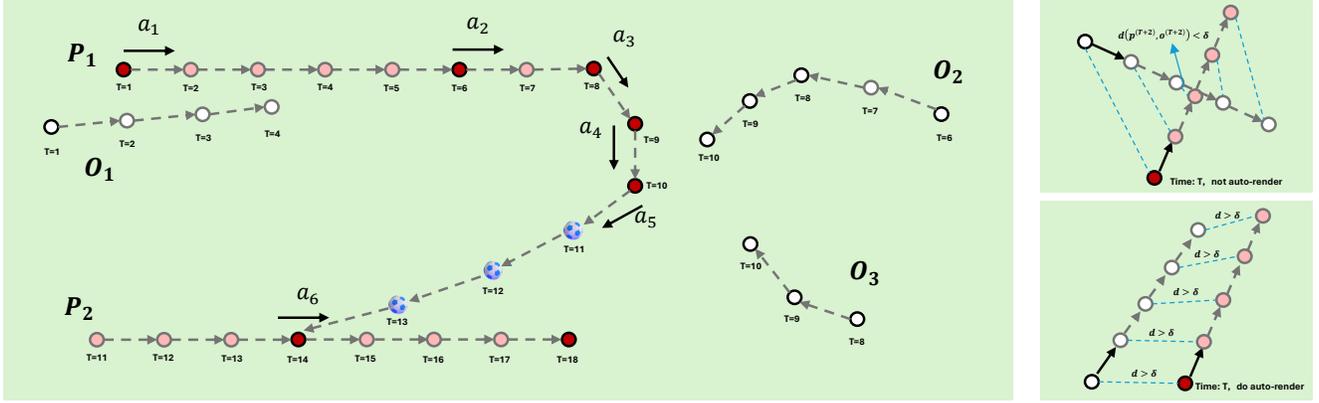


Figure 5. **Left:** Illustration of our auto-rendering algorithm. P_1 and P_2 represent the trajectories of two players on our team, while O_1 , O_2 , and O_3 represent the trajectories of three opponent players. All moments shown in light color indicate that they are automatically rendered, whereas the moments shown in dark color indicate the times when players need to make decisions. **Right:** Pre-rendering technology ensures that control is promptly returned to the player when the opponent’s position and orientation pose a threat (**upper figure**), while reducing the amount of player control when there is no threat (**lower figure**).

Algorithm 1 Automatic Linear Running

```

Require: player.pos, player.v, opponents
Ensure: Automatic rendering of linear running
1: max_frames  $\leftarrow$  10
2: current_frame  $\leftarrow$  0
3: threshold  $\leftarrow$   $\delta$ 
4: while current_frame < max_frames do
5:   collision_detected  $\leftarrow$  false
6:   for  $i \leftarrow 1$  to 5 do
7:     player_future  $\leftarrow$  player.x +  $i \times$  player.v
8:     for all oppo in opponents do
9:       oppo_future  $\leftarrow$  oppo.x +  $i \times$  oppo.v
10:      if d(player_future, oppo_future) < threshold then
11:        collision_detected  $\leftarrow$  true
12:        break
13:      end if
14:    end for
15:    if collision_detected then
16:      break
17:    end if
18:  end for
19:  if collision_detected then
20:    break
21:  end if
22:  renderFrame( $T$ )
23:   $T \leftarrow T + 1$ 
24:  current_frame  $\leftarrow$  current_frame + 1
25: end while
26: return control to player

```

graph represents the rewards obtained by the player in different subsets, which serves as the evaluation result. The bar chart indicates the average number of operations per

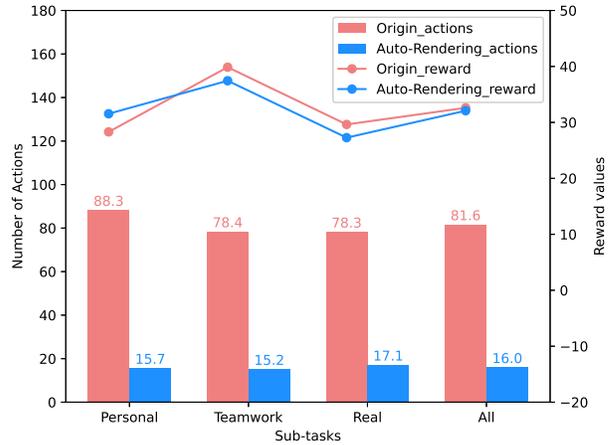


Figure 6. Effectiveness of auto-rendering technique. It reduce nearly 80% of the actions but maintaining the similar evaluation results.

scenario required to complete all tasks (if the player is a model, this corresponds to the number of API calls). The red bars represent the results without using the aforementioned auto-rendering technique, while the blue bars represent the results with the technique applied. It is evident that, without significantly affecting the evaluation results, the use of the auto-rendering technique reduced the number of operations by nearly 80%. This reduction lowered the average number of operations per scenario to 16, substantially saving testing costs for users of our benchmark.

Table 2. Comparison between environments. The term “Dependency” encompasses the test samples from the test set, the databases required for environment setup (such as all web pages and the index for WebShop), as well as the size of newly added pip and apt libraries during installation. We monitor the changes in resource usage before and after the installation in a fixed starting docker to provide an indication. When evaluating latency, we utilized a virtual machine equipped with 96 CPU cores and four 80GB NVIDIA A100 GPUs. For each environment, we experimented with various numbers of processes, such as 1, 2, 4, 8, and so forth, and selected the configuration that yielded the optimal speed. In the last row, + means require some dependency and – means not supported on some platform.

Metric	VirtualHome	Habitat	VAB-OmniGibson	VisualWebArena	WebShop	Notes
Dependency size (MB)	3,174	6,938	25,743	293,953	7,756	size of virtual env. and database
Min RAM required (GB)	8.4	9.8	7.7	14	12	single process
GPU required ?	T	T	T	F	F	T include both supported
1000 steps latency (second)	226	127	217	102	83	best processes, exclude env. loading
Platform	W/L/M	L	L	L	L	W=Windows, L=Linux, M=MacOS
Limitations	+Unity	-A/H100	+11G RTX GPU	-	text only	Special limitation or requirements.

Table 3. Expansion of Table 2

Metric	VAB-Minecraft	WuKong	ours-Sokoban	ours-WebUI	ours-Football	Notes
Dependency size (MB)	3,028	133,120	53	1,780	1,350	size of virtual env. and database
Min RAM required (GB)	5.2	16	1.2	1.7	4.8	single process
GPU required ?	T	T	F	F	F	T include both supported
1000 steps latency (second)	72	582	0.1	40	5.7	best processes, exclude env. loading
Platform	L	W/M	W/L/M	W/L/M	W/L/M	W=Windows, L=Linux, M=MacOS
Limitations	+4G GPU	+ steam	-	-	-	Special limitation or requirements.

A.4. Environment selection

In this section, we will elaborate on the factors considered during our selection process for the environment and discuss the environments that were excluded from our study.

Simplicity. One of the critical requirements in evaluating environments is simplicity and ease of use. In the assessment of large language models (LLMs), tools like evalharness enable the evaluation of dozens of datasets within a unified framework with minimal modifications, and their installation can be accomplished with a single command. Most LMM evaluations based on VQA datasets follow a similar pattern. However, this is not the case with existing agent environments, which are often highly complex or have significant hardware dependencies.

Many physical simulation platforms, such as Habitat, VirtualHome, and OmniGibson in the VisualAgentBench suite, have stringent requirements for GPU types and CUDA versions, often conflicting with the typically high-version torch and CUDA dependencies of LLM code. Some agent frameworks propose controlling UI operating systems such as Windows and Ubuntu, and software applications on these systems like office suites. These environments impose high demands on the platform, presenting prohibitive costs for researchers seeking to quickly validate model capabilities. Additionally, certain environments are inherently large and costly to install. For instance, WebArena, VisualWebArena, and WebShop require complex installation processes and extensive web databases. Some environments aim for realism by running on large games via platforms like Steam, such as “Black Myth: Wukong,” which can be tens

or even hundreds of gigabytes in size, making them impractical. A more detailed quantitative comparison is presented in Tables 2 and 3.

Each of these unmet requirements can lead to slow environment simulation speeds, inability to perform parallel validations, and hinder rapid dissemination of findings.

Visual Feedback. We require that our environment must include visual feedback, and that such visual feedback is irreplaceable. In certain web-based environments, aside from images, the web pages themselves contain substantial information such as HTML and captions. Our observations indicate that tasks can be effectively accomplished using this information. In fact, in real-world scenarios, web automation testing and agent embedding tend to interact directly with HTML elements rather than relying on visual information as an intermediary. Consequently, we exclude such environments, along with those that do not necessitate visual participation, such as LLMAgentBench, from the scope of this paper.

Representativeness on Reasoning. For each environment we select, we must be able to clearly identify the required capabilities. This implies that the environment must have substantial planning space and the possibility for errors, with a high degree of freedom at both the high-level and the action-level. Most existing environments offer significant freedom at the action-level but lack substantial freedom at the high-level. We will illustrate this point with the following examples.

For instance, in our Sokoban environment, high-level planning primarily involves deciding which box to push to

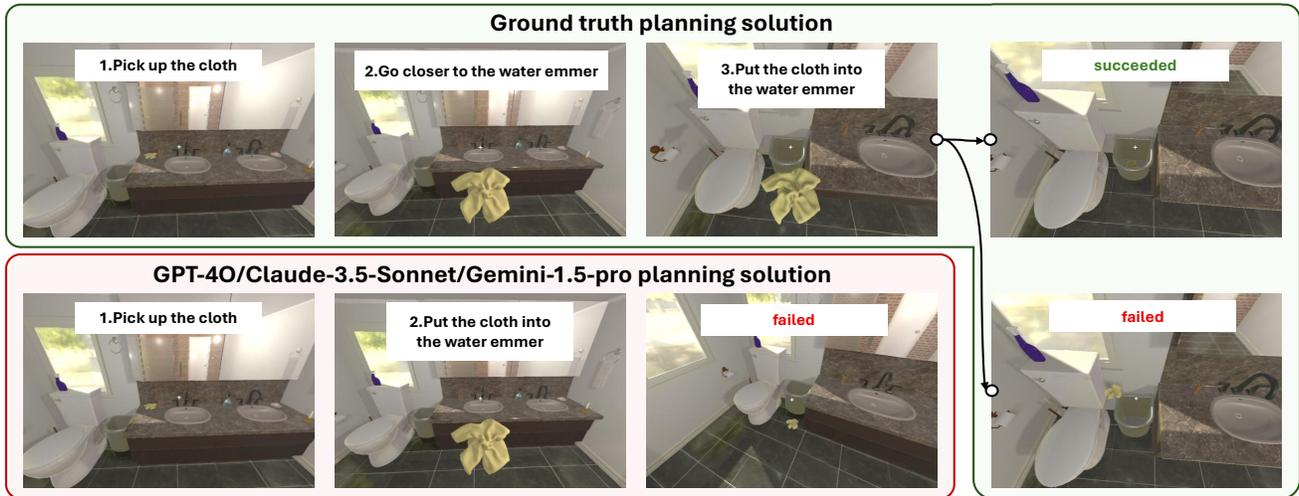


Figure 7. Simple task “Put the cloth into the water emmer” in ALFRED.

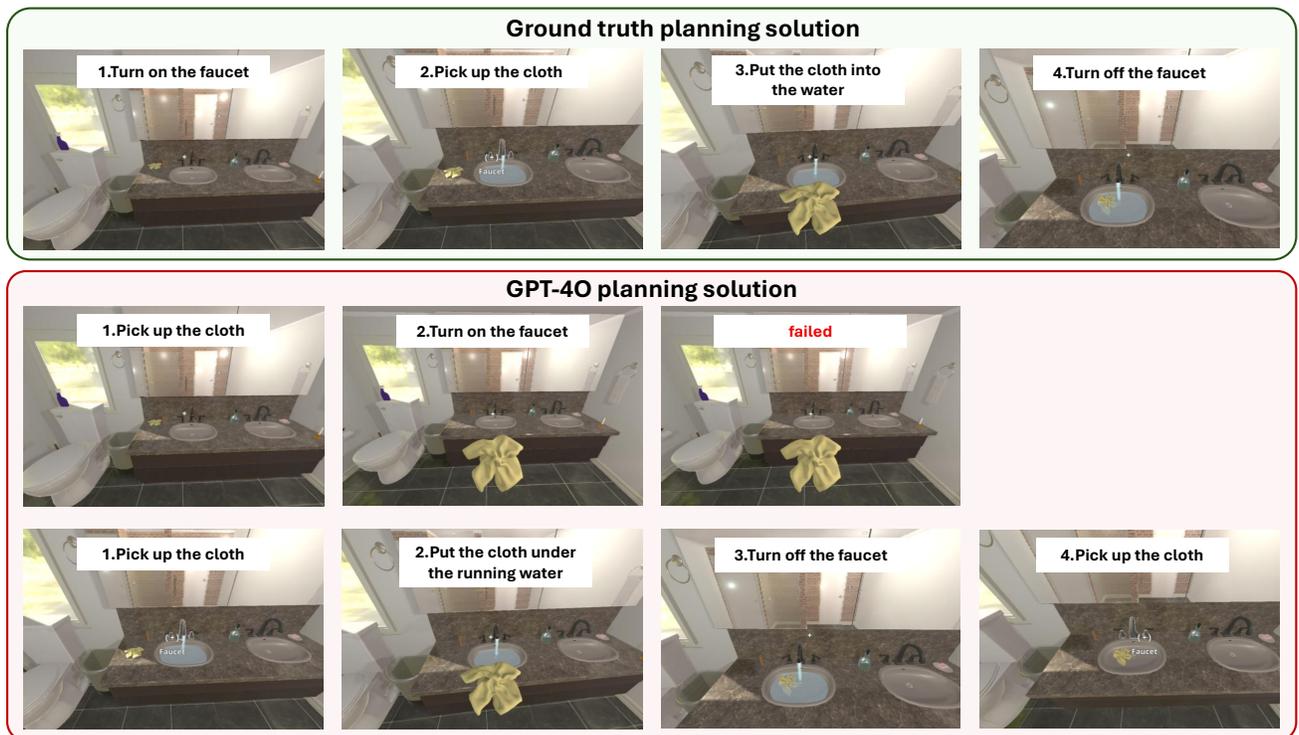


Figure 8. Simple task “Wash the cloth and the put into the water emmer” in ALFRED.

which area and the sequence in which to do so. In this game, pushing a box to the wrong area or in the incorrect sequence can result in a deadlock. In the context of Football, high-level planning refers to the various tactical strategies that can be employed. In WebUI, the HTML part mainly constructs the layout of the webpage, while CSS determines attributes such as the color and font size of elements. At the high-level, HTML can have multiple layout configurations that achieve the same visual outcome, and the corresponding CSS properties may differ. This illustrates that the

generation of HTML reflects the model’s understanding of spatial layout during the structured generation process.

Contrary to our approach, some works impose limitations at the reasoning level. For example, in the CSS environment of VisualAgentBench, the HTML is fixed, and the CSS of the original webpage is modified, requiring the model to debug the corresponding CSS properties. Aside from a small amount of CSS knowledge, this task essentially becomes one of comparing images and identifying differences. Such tasks significantly undermine the gener-

alization capability of structured visual generation, as we have demonstrated through qualitative experiments in Section D.2.

Additionally, various physical simulation platforms like Ravens, RLFRED and VirtualHome also exhibit limitations in high-level planning space. For instance, tasks such as placing a red ball into a red bowl on a cluttered desk or retrieving a cloth from a drawer can fail due to the model’s lack of common sense or perception errors. However, high-level planning tasks like “pick up the red ball, place it in the red bowl, ...” are quite naive and straightforward. Similarly, in web search tasks, the high-level planning typically follows a fixed process: “search for keywords, click the search button, observe the results, click the relevant item, ...”. Errors in such tasks are often due to perception and decision-making mistakes, leaving little room for adjustment at the high-level planning stage.

To better illustrate the aforementioned points, we conducted several case studies on the ALFRED physical simulation platform. We employed LMMs to generate step-by-step plans, utilizing ALFRED’s officially pre-trained PDDL controller to execute specific actions. Figure 7 depicts a simple task: picking up a cloth and placing it into a water emmer. The results from three well-known models were consistent: their high-level planning was accurate, yet they failed to complete the task due to unfamiliarity with the distance parameters in the simulation environment. Even when we initially adjusted the robot’s starting position, using ground truth planning did not guarantee task completion. This is attributable to the randomness inherent in the placement actions within the environment, which resemble tossing objects. Figure 8 illustrates a more complex task where GPT-4O failed due to turning on the faucet while holding the cloth, although this did not constitute an error in high-level planning. When we allowed GPT-4O to begin from the second step of the ground truth plan, the subsequent planning proceeded without errors. By combining the two examples mentioned above, we can conclude the following: **The challenge is not in our inability to design complex tasks using these physics simulation engines, but rather in the fact that as tasks become increasingly complex, the final simulation results are dominated by failures in low-level planning and controller operations.** This dominance obscures the effectiveness of high-level planning.

In fact, beyond the three environments we propose, there are several others that meet our criteria. For instance, DetToolChain, which uses given tools to annotate data, as well as games like Sudoku, puzzle games, and various board games (such as Gomoku and Go), could all serve as robust evaluation environments. However, they may have similarities to those we have already introduced. If we can garner interest from the community, we intend to expand our

project to include more such environments.

A.5. Human labeler details

In the main section, we discussed employing human experts to conduct tests aimed at achieving human-level results. For various environments and settings, we provide a detailed explanation of our labeler selection process and the methods used to guide them in completing the annotation tasks.

Sokoban-Global

- **Candidate selection:** The candidates were selected from thirteen PhD students in STEM fields. They were chosen based on the criteria that they are not long-term enthusiasts of Sokoban and have not extensively played the game in the past year.
- **Testing process:** For each scenario of Sokoban-mini, we provide only the initial scene (a static image) to the annotators, the annotators are required to return a list like [left, up, ...] to us.
- **Prompt to the annotators:** We first provided the goal and output rules of Sokoban. Building on this, for each scenario, we required annotators to directly record the sequence of actions in the document while observing the initial state of the scene. During the entire process of completing a scenario, annotators were not allowed to mark on the image or use their fingers for marking, nor were they permitted to use scratch paper or other tools for note-taking.

Sokoban-Online

- **Candidate selection:** Same as Sokoban-Global.
- **Testing process:** For each scenario of Sokoban-mini, we provide the interact-able environment (like a video game) directly to the annotators, and they will play the game and stop when stuck or finish, and the environment will record the reward.
- **Prompt to the annotators:** We provided the goal and output rules of the environment. The annotators should proceed as if they are playing a regular game, and they can stop when they either get stuck or complete the task.

WebUI-Global

- **Candidate selection:** The candidates include eight individuals, comprising undergraduate students majoring in software engineering with expertise in front-end programming, as well as front-end developers already working at internet companies.
- **Testing process:** For each scenario of WebUI-mini, we provide the webpage description document, the same as we provide to models. Annotators are required to reproduce the webpage without using a browser.
- **Prompt to the annotators:** We provided the goal, webpage description and output format. Since reproducing

Table 4. Evaluation on MageBench with unified prompt.

model	WebUI AES (%)	Sokoban Reward		Football Reward
	Global ($\delta = \pm 0.8$)	Global ($\delta = \pm 0.5$)	Online ($\delta = \pm 0.4$)	Online ($\delta = \pm 2.0$)
InternVL2-1B	10.61	46.01	46.06	-2.47
Xcomposer-2.5-1.8B	16.14	45.76	45.60	IFE
InternVL2-2B	17.33	45.94	46.11	2.00
Qwen2-vl-2B	9.26	46.43	46.52	0.64
InternVL2-4B	2.92	46.32	46.28	-0.03
Phi-3.5-V-4.2B	1.36	46.62	46.74	11.04
Yi-VL-6B	8.08	IFE	IFE	IFE
DeepSeek-VL-7B	18.22	45.60	46.64	IFE
Xcomposer-2.5-7B	15.32	46.84	46.39	IFE
LLaVA-v1.5-7B	11.42	45.60	46.19	-0.80
Llava-1.6-mistral-7B	9.07	46.37	46.23	-3.64
Qwen2-vl-7B	11.98	46.38	46.41	2.68
MiniCPM-V2.6-8B	16.06	46.53	46.86	3.61
InternVL2-8B	27.43	46.56	46.35	7.90
Llama-3.2-11B-Vision-Instruct	39.22	46.54	47.33	11.58
LLaVA-v1.5-13B	13.30	46.65	46.53	16.59
InternVL2-26B	31.68	46.77	46.50	15.62
Llava-1.6-34B	1.65	46.06	46.48	3.04
Yi-VL-34B	7.21	IFE	IFE	IFE
InternVL2-40B	7.20	45.65	46.82	12.99
Qwen2-vl-72B	11.70	47.14	47.84	15.02
NVLM-72B	10.36	46.39	46.68	18.43
InternVL2-76B-LLaMA3	28.45	46.61	47.09	8.04
Llama-3.2-90B-Vision-Instruct	42.21	47.14	IFE	8.32
GPT-4o	40.02	47.46	48.63	17.78
Gemini-1.5-pro	52.10	47.65	48.33	19.61
Idle Baseline	0.00	45.60	45.60	2.36
Random Baseline	0.00	47.40	47.40	16.64

20 webpages is not an easy task, we permit annotators to use LLMs to generate initial drafts. However, they are required to only observe and adjust the relevant code, such as settings in the CSS, without running debugging or rendering the webpages. Excluding the time spent using LLMs, we mandate that the manual verification and editing time for each web page should exceed ten minutes.

WebUI-Online

- **Candidate selection:** Same as WebUI-Global
- **Testing process:** We allow annotators to use the code from the WebUI-Global settings as the initial code. Without using LLMs, they are permitted to freely utilize code executors and browser rendering to debug and modify code and CSS properties. This process continues until they believe the output matches the provided webpage description precisely.
- **Prompt to the annotators:** Annotators are required to reproduce the webpage according to the description and as perfect as possible.

Football-Online

- **Candidate selection:** The candidates were selected from thirteen PhD students in STEM fields. They were chosen based on the criteria that they are not long-term enthusiasts of the FIFA Football Game and have not extensively played the game in the past year.
- **Testing process:** For each scenario of Football-mini, we provide the interact-able environment (like a video game) directly to the annotators, and they will play the game till an end, and the environment will record the reward.
- **Prompt to the annotators:** We provided the goal and output rules of the environment. The annotators should proceed as if they are playing a regular game.

B. LMM-as-Agents

We represent our Agent-Environment system and its evaluation using a Partially Observable Markov Decision Process $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O})$, indicating that the agent π_θ cannot directly access the complete state from state space $\mathbf{s}_t \in \mathcal{S}$. Instead, the agent has to observe vision feedback (and prob-

ably few text feedback for WebUI) in observation space $\mathbf{o}_t \in \mathcal{O}$, and leverage its planning ability to generate actions within a discrete action space $\mathbf{a}_t \in \mathcal{A}$. The environment will serve as the transition function to update the state based on the agent’s actions: $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. Finally, our designed reward assigning function \mathcal{R} will provide the evaluation results.

There will also be some pre-defined prompts involved in the agent. System prompt p_{sys} introduce the rule, target, available actions of the system. In WebUI, task prompt p_{task} is a specific description of a Webpage. For other environment, $p_{task} = \mathbf{o}_1$. CoT prompt p_{cot} and IO prompt p_{io} designate the model’s inner thought flow and output format so that we can parse the text-based output to actions.

In addition to the basic formulation mentioned above, we will also formalize the following two fundamental agent designs, which will serve as baseline agents for the dataset proposed in this paper:

- **Global Planner Agent.** We consider the following basic Agent design: it only observes the initial environment once (in the task prompt). Then, it continuously makes all subsequent decisions, formulated as

$$\pi_{\theta}(p_{sys}, p_{task}, p_{cot}, p_{io}) \rightarrow \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_T.$$

- **Online Planner Agent.** This baseline agent analyzes each step and takes actions online while possessing two attributes: action-memory (AM) and observation-memory (OM), where $OM \leq AM$, which are hyper-parameters of the agent. For example, when $AM = 3$ and $OM = 2$, the decision-making process of the agent can be formalized as follows:

$$\pi_{\theta}(p_{sys}, \mathbf{a}_{t-2}, \mathbf{a}_{t-1}, \mathbf{o}_{t-1}, \mathbf{a}_t, \mathbf{o}_t, p_{cot}, p_{io}) \rightarrow \mathbf{a}_{t+1}.$$

We separate these two kinds of memory because the number of input images is limited for most of the LMMs. For WebUI, we will also add p_{task} after the p_{sys} .

Global planner can better reflect the planning capabilities of the model itself. It is similar to the process used to generate zero-shot CoT results for existing QA-based benchmarks. The online planner represents another batch of agents that receive feedback from the system and adapt themselves, such as Reflexion, ReACT, ViLA, etc.

C. More Results

C.1. MageBench results

In Table 4, we present the test results of additional models on the larger MageBench full set. We selected models of varying sizes from 14 different model families, totaling 26 models. Given the large number of scenarios, each model and scenario only needed to be tested three times to compute the mean, resulting in a very small variance. The related conclusions are similar to those discussed in the main part, so they will not be reiterated here. Notably, we did

not measure the results for Claude-3.5-Sonnet and human participants, as we were unable to bear the associated economic costs for the full dataset. Additionally, as mentioned in the main text, the Online setting in the current WebUI environment is essentially non-functional. Consequently, we did not allocate resources to testing in this setting.

C.2. Agent memory

We selected the most robust model in the Sokoban environment under the online setting, specifically GPT-4o, to conduct memory ablation experiments. In this context, “AM” refers to action memory, which denotes the number of historical actions, while “OM” signifies observation memory, representing the number of observations inputted for generating the next action. Here, observations refer to the game’s visual frames. We have chosen AM values of 1, 5, and 10, and OM values of 1, 2, and 3 for our combinations. For instance, when AM=10 and OM=2, the model’s input and output can be described as follows (For the sake of brevity, we have omitted p_{sys}, p_{cot}, p_{io}):

$$\pi_{\theta}(\mathbf{a}_{t-9}, \dots, \mathbf{a}_{t-2}, \mathbf{a}_{t-1}, \mathbf{o}_{t-1}, \mathbf{a}_t, \mathbf{o}_t) \rightarrow \mathbf{a}_{t+1}.$$

Table 5. Ablation on memory length.

reward	AM=1	AM=5	AM=10
OM=1	53.508	53.417	52.458
OM=2	-	52.242	54.350
OM=3	-	53.392	53.325

In our expectations, as the context length increases, the model, having access to more preceding information, should be able to perform the task better. However, in reality, we have not observed a stable correlation between context length and performance, as shown in Table 5. The values presented in the table are all within the range of variance, thus it cannot be concluded that the impact is stable. This result indicates that the model’s performance, based solely on the previous frame and action, does not differ from its performance when more memory is added. This is inconsistent with human behavior.

To identify potential causes, we experimented with several other cutting-edge and powerful models:

- **Reasoning model:** They possess enhanced reasoning capabilities and the ability to process lengthy content through test-time scaling, we selected O4-mini and GPT-5 as examples.
- **End-to-end agent model:** Recently, there have been numerous agent models, such as DeepResearch, which may have been trained on trajectory-based data. We chose to focus on Computer Use Agent Models, using OpenAI Operator and the open-source UI-TARS-1.5-7B as examples. These models have been trained not only on

ViC-type data but also on task types that closely resemble our benchmark. Note that for these models, we will slightly modify our format prompt to adapt their interaction method to keyboard input.

Table 6. Impact of memory with different models.

O4-mini			GPT-5		
Reward	AM=1	AM=5	Reward	AM=1	AM=5
OM=1	58.425	58.992	OM=1	57.313	58.244
OM=3	-	59.739	OM=3	-	58.302
OAI Operator			UI-TARS-1.5-7B		
Reward	AM=1	AM=5	Reward	AM=1	AM=5
OM=1	59.231	60.425	OM=1	50.128	53.305
OM=3	-	63.071	OM=3	-	55.711

The results in 6 indicates that even a pre-trained base model with stronger reasoning or long-context processing capabilities (O4-mini and GPT-5) can only ensure that performance does not degrade with longer memory, but it cannot fully leverage historical information. However, recent models aimed at general-purpose agents, despite differing tasks, have incorporated trajectory and ViC-type data. When transferred to our benchmark, they not only capitalize on the advantages of the online setting but also significantly enhance the model’s capability with memory. This strongly supports our view: the most crucial aspect of developing general-purpose agent models is the inclusion of substantial ViC-type data during the pre-training phase. This capability is independent of general reasoning abilities (for instance, the single-turn reasoning capability of UI-TARS-1.5-7B is not very strong) and can be generalized. This also confirms the feasibility of training general-purpose agents.

C.3. Error Analysis

We have categorized the types of errors for different models on MageBench. For Sokoban and Football, since most errors stem from model decision failures, we additionally categorize parsing errors during the parsing process, as shown in Table 7. Among these, “Invalid Actions” refer to errors arising from outputs that do not conform to the required format, leading to parsing errors or incorrect actions. “Repeating Actions” is a common comprehensive error in LLMs, which refers to the phenomenon where the same token is repeated multiple times. If such output is successfully parsed, it results in a large number of repetitive actions.

As shown in Table 7, we observed that open-source models exhibit a significant number of IA and RA type errors in the online setting, and Claude’s performance is similar. This phenomenon is highly consistent with the online results presented in Table 1 of the main paper. Regarding

RA type errors, some studies have pointed out that they are commonly found in scenarios where the training and testing domains do not match. This indicates that both the open-source models and Claude have deficiencies in training with the Vision-in-the-Chain type of data studied in this paper.

Table 7. Error type summarization. “IA” stand for “Invalid Actions” and “RA” for “Repeating Actions”. Both types of error are the reflection of instruction following failure. The values are presented in percentages (%). Football results with mark † mean these models sometimes present a trajectory only consists of long-pass, short-pass and high-pass, we assign this situation as “RA” too.

Model	Sokoban-G		Sokoban-O		Football-O	
	IA	RA	IA	RA	IA	RA
GPT-4o	0.0	5.0	0.0	6.7	0.0	7.5
Gemini	15.0	3.3	25.0	0.0	0.0	3.0
Claude	0.0	6.7	0.0	1.7	0.0	43.0 [†]
InternVL2	0.0	5.0	5.0	80.0	0.0	70.0
LLaVA-1.5	0.0	3.3	5.0	35.0	0.0	25.0 [†]
MiniCPM	0.0	6.5	0.0	33.3	6.7	50.0 [†]
Phi-3-V	0.0	0.0	0.0	23.3	0.0	24.5

C.4. Failure of WebUI online

In Figure 9, we evaluated the online agent capabilities of three product-level models in modifying code based on rendered images across different prompts. Specifically, “0s-CoT” refers to the model being asked to analyze and adapt the code without specifying how to analyze it; “HD-CoT” stands for human-designed analysis processes, such as instructing the model to first observe the differences in the rendered images and then describe how to modify the code; “HD-CoT-NoIMG” does not provide the rendered images, only prompts the models to observe errors in the code. In Table 1 of the main part, We used “HD-CoT”. Unfortunately, these models almost uniformly failed to perform well across all settings. Although they may show slight and unstable gains under self-reflection due to correcting some syntax errors.

D. Benchmark generalization capability

Given that our benchmark is ability-oriented, we aim for the knowledge it provides to have a certain degree of generalizability and to offer insights for future engineering developments. To illustrate this point, we conducted some qualitative experiments to demonstrate that our metrics continue to generalize well when the model is transferred to other similar tasks with engineering significance. Our explanation is divided into two parts: the first part addresses how ViC-type reasoning, as indicated by Sokoban-Online and Football-Online metrics, generalizes to potential applications in robotics and embodied AI. The second part dis-

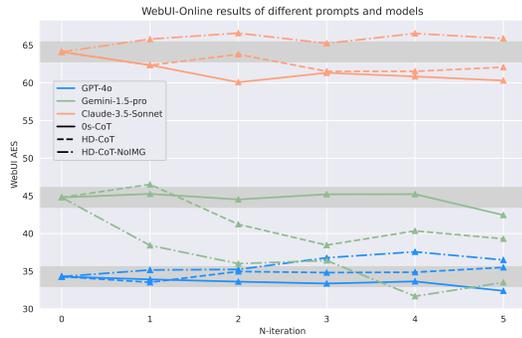


Figure 9. WebUI-Online results. Different line styles represent different prompt types, while different colors denote different models. The horizontal axis shows the number of iterations the model takes to modify the webpage code based on feedback. The gray-shaded areas indicate regions of variance.

cusses how WebUI generalizes across various structured visual generation tasks. We will elaborate on each part below.

D.1. Robotics and embodied AI

We hope to assert that our ViC-type reasoning could be significant in fields such as robotics. However, real-world mechanical arm environments are extremely costly, and the generalizability across different mechanical systems is often poor. In virtual physical environments, the input for mechanical arms typically consists of numerical values like torque and angles, and it is evidently unrealistic for LMMs to generate these values directly. One potential alternative is for LMMs to generate only high-level planning, with the execution of actions relying on predefined action models such as PDDL. However, this approach does not effectively evaluate the planning capabilities of LMMs. This is because the predefined actions are very limited, and if the actor model makes an error, LMM has no means to correct it.

To perfectly decouple planning and control while obtaining a rich planning space, we have designed a human-in-the-loop testing method. Simply put, this involves using humans as robots, with the LMM providing detailed natural language instructions for actions. This way, humans can accurately execute the actions as directed by the LMM. Consequently, the quality of the final outcome is solely determined by the effectiveness of the LMM’s planning. We have designed two tasks for LMM in real-world scenarios:

Clean-up table. As shown in Figure 10, we first replicate the scene depicted in the image each time. This scene features a cluttered desk with various objects scattered about. The LMM is required to provide instructions to the robot to organize the desk so that it appears clean and tidy. The specific details of the task are described as follows.

Task: Your job is to manipulate the

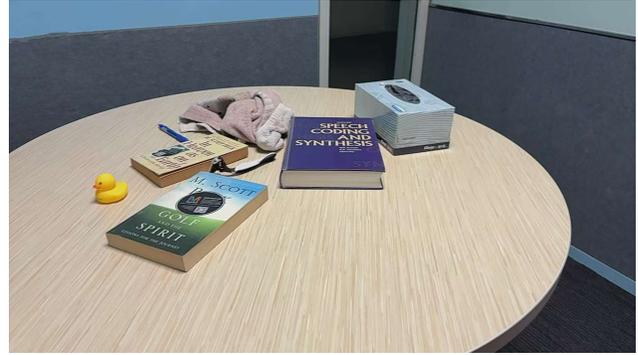


Figure 10. The initial scene for Clean-up table task, which is carefully replicated for each model before testing. This replication is done manually to ensure the scene is as consistent as possible across different tests.

Task:

There are only 3 regions can put the book: 'A', 'B' and 'C' (You can see it is marked on the table), and there are 3 books with different size.

Your job is to put all of the book on the region 'C' with increasing book size.

Your target scene should look like this:



However, during the whole process of operation, you cannot put a larger book on a smaller book (in a region).

If you break this rule, no action will be taken.

Your order should be like:

Put book in to region

(You have to satisfied the format and do valid order even if you not sure or can not see clearly.)

Figure 11. Task description for Book Hanoi tower task.

items on the table, put all the items orderly on the table and make the table looks good.

You cannot put anything out of the table.

Book Hanoi Tower. Similar to the Tower of Hanoi game, we divided the desk into three areas. There are a total of three books of different sizes. The model needs to ultimately stack the three books in area “C” in order from smallest to largest. Apart from the initial setup, during the moving process, larger books cannot be stacked on top of

model	GPT-4O	Claude-3.5-Sonnet	InternVL2-7B
S-mini-O	53.03	45.35	42.89
F-mini-O	21.20	16.94	8.74
Clean up the table			
Book Hanoi Tower			

Figure 12. The Sokoban-Online, Football-Online result on MageBench-mini of the 3 models, and the last frame of their performance on our real-world planning evaluation.



Figure 13. Initial scene for Book Hanoi Tower task.

smaller books. Figure 11 illustrates the task description provided to the model, while Figure 13 shows the initial setup. This initial scene is specifically designed to prevent the model from being overly familiar with the Tower of Hanoi algorithm.

D.1.1. Overall results

In Figure 12, we directly present the performance of three models, *i.e.*, GPT-4o, Claude-3.5-Sonnet, and InternVL2-7B, under the two newly constructed tasks. The figure illustrates the final environmental scenarios for these tasks. Overall, it can be observed that the planning capabilities in real-world scenarios are highly correlated with the test scores from our MageBench.

GPT-4o successfully achieved the preset objectives in

both scenarios, albeit with a number of seemingly meaningless actions in between. In contrast, Claude demonstrated some effective actions but often got stuck at a certain point within a scenario. As discussed in our paper, open-source models still have a considerable gap to close.

The aforementioned results indicate that ViC-type reasoning is indeed representative. In the next section’s visualizations, we will also observe that common errors in Agent environments are similarly prevalent in robotic planning. This suggests that our understanding of their error types is generalizable.

D.1.2. Visualization

Figures 14 and 15 illustrate the outputs of GPT-4o and the environment feedback across two tasks, encapsulating our proposed Vision-in-the-Chain (ViC) framework. It is evident that GPT-4o exhibits a significant number of repetitive and meaningless actions in the Book Hanoi Tower task. Furthermore, other models, such as InternVL2-7B, predominantly demonstrate errors characterized by the initiation of repetitive and identical analyses and actions after performing two or three initial actions correctly. This pattern aligns with the error types analyzed in MageBench, suggesting that due to the lack of ViC-type and multi-turn chat data, these models encounter instruction-following failures when faced with tasks in an online setting.

D.2. Structured Visual Generation

In MageBench, our WebUI environment is designed to reconstruct a target website. Our goal is to assess whether this



Analyze
The current observation does not show the table or any items on it. The view is focused on the floor, a chair, and part of a laptop.

Action
Adjust the camera to focus on the table and its contents.



Analyze
The table has several items: a pen, a pair of socks, three books, a rubber duck, and a box of tissues. These items need to be arranged neatly.

Action
Pick up the pen and place it parallel to the edge of the table near the top right corner.



Analyze
The pen has been moved to the top right corner. The remaining items still need to be organized.

Action
Pick up the pair of socks and place them neatly folded next to the tissue box on the right side of the table.



Analyze
The socks have been placed neatly next to the tissue box. The remaining items still need to be organized.

Action
Pick up the book titled "Speech Coding and Synthesis" and place it parallel to the edge of the table, next to the tissue box on its left side.



Analyze
The book titled "Speech Coding and Synthesis" has been placed neatly next to the tissue box. The remaining items still need to be organized.

Action
Pick up the book titled "Golf and the Spirit" and place it parallel to the edge of the table, next to the book "Speech Coding and Synthesis" on its left side.



Analyze
The book titled "Golf and the Spirit" has been placed next to the book "Speech Coding and Synthesis". The remaining items still need to be organized.

Action
Pick up the book titled "In Heaven as on Earth" and place it parallel to the edge of the table, next to the book "Golf and the Spirit" on its left side.



Analyze
The book titled "In Heaven as on Earth" has been placed next to the book "Golf and the Spirit". The remaining items still need to be organized.

Action
Pick up the rubber duck and place it near the bottom left corner of the table, ensuring it is aligned with the edge.



Analyze
The rubber duck has been placed near the bottom left corner of the table. The remaining items still need to be organized.

Action
Pick up the black and white keychain and place it next to the rubber duck, ensuring it is aligned with the edge of the table.



Analyze
The black and white keychain has been placed next to the rubber duck. The remaining items still need to be organized.

Action
Pick up the blue pen and place it parallel to the edge of the table, near the top right corner of the table.



Figure 14. GPT-4o on Clean-up table.



Analyze
The current observation is too dark to see the books or regions clearly.

Action
Put book in region A to region B



Analyze
There are two books visible on the table, one in region B and one in region C. Region A is currently empty.

Action
Put book in region B to region A



Analyze
Now, there is one book in region A and two books in region C. Region B is currently empty.

Action
Put book in region C to region B



Analyze
Now, there is one book in region A, one book in region B, and one book in region C.

Action
Put book in region C to region B



Analyze
Now, there is one book in region A and two books in region B. Region C is currently empty.

Action
Put book in region A to region C



Analyze
Now, there is one book in region C and two books in region B.

Action
Put book in region B to region A



Analyze
Now, there is one book in region A, one book in region B, and one book in region C.

Action
Put book in region B to region C



Analyze
Now, there is one book in region A and two books in region C.

Action
Put book in region A to region B



Analyze
Now, there is one book in region A, one book in region B, and one book in region C.

Action
Put book in region B to region C

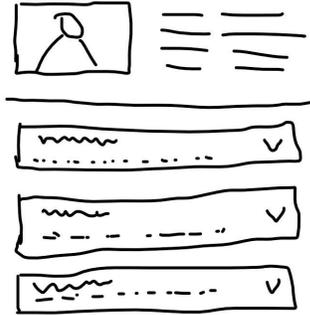


Figure 15. GPT-4o on Book Hanoi Tower.

I would like you to help me generate a web page for my academic resume, and I will provide you with the requirements and a draft diagram of the web page.

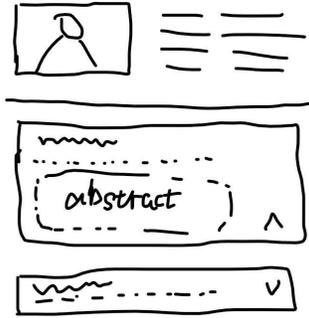
Layout description

The webpage should look like this original webpage:



The top left part is the photo, and its right are some basic information like name, birthday and etc, you can generate several item there. Below is a paper list, you need a separate line between the basic information part and the paper list part as shown in the image. Then, each paper should be in a box. On the right side there is a button that shows the drop-down box. On each box there should be 2 lines, the first line with larger font size shows the paper title and the second line shows the author list.

When user click on the button on the box, it will show the larger box and show the abstract of the paper, it should look like this:



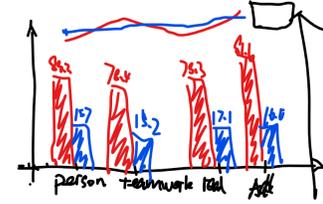
Resources

We will provide all information for you to generate the webpage:

- photo path: (images/photo.png)
- paper list:
 - Title: "How to make American great again", author: "Donald J. Trump, Vans, Elon R. Musk, Robert Francis Kennedy Jr.", Abstract: "This paper researched how to make American great again."
 - Title: "What should we do to make American great again", author: "Donald J. Trump, Vans, Elon R. Musk, Robert Francis Kennedy Jr.", Abstract: "This paper researched what should we do to make American great again."
 - Title: "Why we should make American great again", author: "Donald J. Trump, Vans, Elon R. Musk, Robert Francis Kennedy Jr.", Abstract: "This paper researched why we should make American great again."

You are free to design any CSS attributes and animations to make the webpage look better. Now, please generate the HTML code (and CSS and JavaScript code if needed).

I need you to help me visualize the data as shown in the figure below:



original data is a python dict:

```
data = {
  "person": {"Q1": 88.204, "Q3": 15.652, "Q_reward": 28.303, "AR_reward": 31.543},
  "framework": {"Q1": 78.403, "Q3": 15.16, "Q_reward": 39.886, "AR_reward": 37.444},
  "tool": {"Q1": 78.27, "Q3": 17.139, "Q_reward": 29.633, "AR_reward": 27.273},
  "job": {"Q1": 81.64, "Q3": 15.98, "Q_reward": 32.601, "AR_reward": 32.085},
}
```

1. In the graph, the x-axis is the key of the dictionary, that is, the four values of "person", "framework", "tool", and "job", and the bar chart shows the variables of the "Q1" property, where the red part is "Q1" and the blue part is the "AR_reward".
2. Above each bar, the specific value of the bar is marked with the corresponding color.
3. A line chart is also drawn above the entire bar chart, again red for "Q_reward" and blue for "AR_reward".
4. The y-axis on the left side of the entire chart is the value of the bar chart, while the y-axis on the right represents the value of the line chart.
5. Finally, you also need to display the legend correctly and reasonably.

You can generate python code and draw this visualization using any python library.

Figure 16. Task description for webpage generation (left and middle) and data visualization (right).

I will provide you a png format image, I want you to transform it into a svg image by writing svg code to re-implement the image.

Here is the image:



Now please generate complete svg code.

Figure 17. Task description for image to SVG task.

testing environment can accurately reflect the model's capability in structured visual generation, with an eye towards future applications in engineering practice. To this end, we have devised three structured visual generation tasks:

- Generate a webpage based on hand-drawn sketches and descriptions.
- Generate SVG images (code) from observed pictures.
- Create Python code for data visualization.

The specific task descriptions for these three tasks, which serve as the model inputs, are presented in markdown format in Figures 16 and 17.

In Figure 18, we evaluate six models: Claude-3.5-Sonnet, Gemini-1.5-pro, LLaMA-3.2-90B, GPT-4o, InternVL2-72B, and DeepSeek-VL. These models represent the top six based on WebUI scores. The figure includes their WebUI scores alongside the visualized results (Best of 5) from the three aforementioned generation tasks. Overall, the qualitative analysis results are highly correlated with the WebUI scores. We elaborate on this correlation in the following.

Firstly, regarding web page generation, the Claude model not only perfectly meets the requirements but also includes additional details such as dropdown animations, hover animations, and container shadow enhancements (although these may not be very apparent in the figure). Gemini correctly implements all functionalities, though the visual appearance is relatively less dynamic. The similarly scored LLaMA-3.2-90B and GPT-4o exhibit certain deficiencies; for instance, LLaMA-3.2-90B has discrepancies in the shape and position of the dropdown button compared to the task description, while GPT-4o has errors in implementing the dropdown functionality. The two lower-scoring models clearly lag behind the others.

In the SVG generation task, it is evident that Claude stands out from the other models, producing results that most closely resemble the specified fonts and logo shapes. The middle three models form a distinct group, while the last two models exhibit noticeable differences that set them apart from the others.

For the data visualization task, we have four core requirements (each worth one point): correct display of the bar chart, correct display of the legend, the line chart should be scaled to the top, and numerical labels in the same color as the bar chart. Claude did not meet the bar chart requirement but addressed all other aspects, earning a score of 3 out of 4. Gemini failed to scale the line chart correctly and had errors in font color and placement, resulting in a score of 1.5. Similarly, LLaMA-3.2-90B and GPT-4o scored between 0.5 and 1 point. The last two models continued to perform poorly in this task.

The aforementioned analysis indicates that our results can be robustly generalized to other structured visual generation tasks, including engineering applications. Notably, in the VisualAgentBench, they introduced an environment

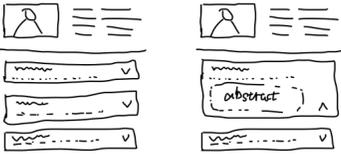
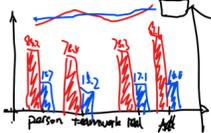
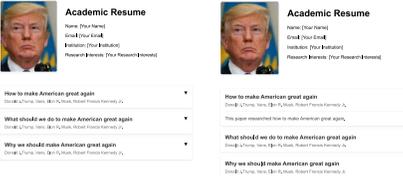
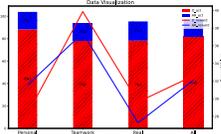
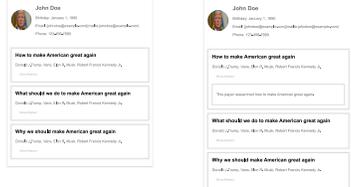
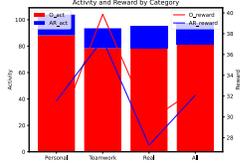
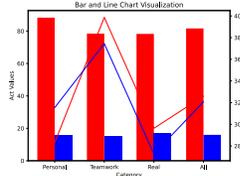
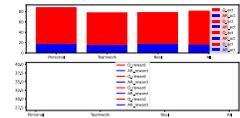
Model	WebUI Result	Webpage generation with description & sketch	Image to SVG generation	Data visualization
Claude -3.5- Sonnet	64.11			
Gemini -1.5- Pro	44.79			
LLaMA-3.2 -90B- Vision- Instruct	35.09			
GPT-4o	34.28	 <p data-bbox="704 1150 915 1205">Incorrect code No button implemented</p>		
InternVL2 -76B- LLaMA3	16.22		<p data-bbox="958 1297 1192 1352">Buggy code, below is the Correct part of code</p> 	<p data-bbox="1250 1360 1451 1415">Buggy code Cannot render figure</p>
DeepSeek -VL- 7B	13.78	 <p data-bbox="678 1570 915 1625">Incorrect code No button implemented</p>		

Figure 18. The cross-domain results demonstrate that our WebUI environment can robustly generalize to other engineering-level structured visual generation tasks.

called VAB-CSS, where models debug CSS properties. In their results, the performance ranking was as follows: GPT-4o > InternVL2-8B > LLaVA-Next-8B > Claude-3.5-Sonnet > Gemini-1.5-pro > Qwen-VL-9B. This differs from our results and qualitative analysis. The discrepancy arises because debugging CSS properties and end-to-end webpage generation are fundamentally different tasks and may not be strongly correlated. As previously mentioned, GPT-4o demonstrated the best performance in the Online setting, which might give it an edge in the CSS debugging task. However, for selecting a model for end-to-end structured visual generation tasks, our environment provides more relevant insights.

E. RL training details

The model is trained in the Global settings. We chose to train in the Sokoban-Global environment for several reasons: Firstly, the Global setting supports efficient implementation of On-policy RL. Secondly, as shown by the best-of-N curve in the main part, Sokoban-Global is the most challenging setting for RL, thus providing the best representativeness. Lastly, the Sokoban environment is highly streamlined and offers excellent scalability—we can easily generate ten thousand random levels, strictly removed duplication with the testing set.

We require the model to output its analysis and results within the `<think>`, `</think>` and `<answer>`, `</answer>` tags. In addition to the task reward provided by the system, we also include a format reward of 0.5 (the model will receive this portion of the reward if its outputs satisfy the format requirements). Furthermore, inspired by the work of OpenReasonerZero, we attempted and found that KL regularization is not necessary in our scenario. Finally, we observed that directly training the model leads to an increase in meaningless and repetitive thinking. However, when we introduced a repeat penalty, the model quickly collapsed into a state where it produced very few outputs. We discovered that adding a small encouragement term significantly aids in the cold start of the model, and combining both the encouragement term and the repeat penalty yields the best performance.

We utilized the original PPO implementation and adapted the OpenRLHF codebase to efficiently deploy various models and their optimizations. As detailed in the main text, we did not employ KL regularization, which allowed us to forgo deploying a reference model, thereby improving code efficiency.

Regarding the training data, specifically the Sokoban levels, we utilized the gym-sokoban repository to efficiently generate a large number of random levels. Given the high difficulty of the task, we set the majority of levels to a lower difficulty to ensure that the model could learn through accumulated experience. Specifically, we generated 5000 lev-

els each for Sokoban-small-v0 and Sokoban-small-v1, and 500 levels each for Sokoban-v0, Sokoban-v1, and Sokoban-large-v0, totaling 11,500 scenarios. Additionally, we ensured strict deduplication of the 200 levels in the test set, which was achieved based on the level configuration.

The reward definition during the training process is as follows: The task reward is based on the reward provided by the gym-sokoban system (e.g., pushing a box to the target location yields a reward of 1.0, and successfully completing all boxes yields a reward of 10.0). The format reward is 0.5 if the model’s output adheres to the format “< think > xxx < /think > < answer > xxx < /answer >” and if the actions within the answer can be correctly identified; otherwise, it is 0. The length encouragement is defined as follows:

$$len_reward = \begin{cases} len(output) * 0.001 & \text{if } len(output) < 200, \\ 0.2 & \text{else.} \end{cases}$$

This ensures that the model output reaches a length of 200, preventing any reward hacking phenomena from occurring. At last, the repeat penalty is -0.2 , when repetition detected, it will be removed from the total reward.

The table 8 details the hyper-parameters used for training our model. Due to the high instability inherent in reinforcement learning, we stored multiple checkpoints and selected the best one for reporting the results.

Table 8. Hyper-parameters for RL training.

Hyper-parameters	Value
pretrain model	Qwen-2.5-vl-Instruct
model size	3B
train_batch_size	32
rollout_batch_size	32
temperature	1
n_sample/prompt	16
num_episodes	4
gen_max_len	8192
advantage_estimator	gae
lambda	1
gamma	1
precision	bf16
deepspeed	O3
actor_lr	1.00E-06
critic_lr	5.00E-06
kl_coef	0
normalize_reward	TRUE

F. Future researches and limitations

In the process of proposing MageBench and conducting our evaluations, we have identified some capabilities that existing models lack, feasible research directions, and open questions. We hope that these summaries can assist and inspire future work:

- Existing open-source models exhibit a significant deficiency in ViC-type reasoning capabilities. How to obtain large amounts of ViC-type training data is an open question worthy of research.
- The models exhibit a lack of imagination and long-term planning abilities in visual spatial reasoning tasks (such as Sokoban). The Best-of-N results indicate that the current models do not possess such potential, necessitating fundamental, mechanism-level research.
- For most tasks that exhibit potential, the models show considerable improvement in the Best-of-N results. How can we design a reinforcement learning approach to enhance the models, enabling them to accumulate explainable, text-expressible experiences, similar to how humans accumulate tips and tricks for playing games?
- Theoretically, an Online Agent should benefit from having a longer memory when completing tasks such as Sokoban and Football, or when making multiple rounds of modifications to a generated webpage. However, our observations show counterintuitive results where not only is there no benefit, but there is even a slight decline in performance. From an engineering perspective, this could be due to the model’s lack of ability to handle long contexts, multi-turn dialogues, and tasks involving multiple images.

Limitations. Currently, we have a limited number of environments, and the environments are relatively simple, which might not fully capture the nuance and messiness of real-world scenarios. Additionally, to validate the model’s capabilities, we use a standardized and simple agent setup. If this work garners attention within the community, we plan to substantially increase the number of environments in future work. This will allow us to explore technical details at the agent level, such as long-short term memory, retrieval-augmented generation, code involvement, and more.

G. Examples

G.1. WebUI

G.1.1. Labeled target webpage

In this subsection, we will present specific examples of webpage processing. First, regarding the preprocessing of the target webpage’s HTML, as mentioned in Section A.1.1, we will annotate two new attributes to the atomic elements of the webpage: ‘data-filter-by’ and ‘data-eval-by’. Figure 19 shows a screenshot of a sample webpage, and the processed code is shown in Figure 20.

G.1.2. Interactions

In order to simulate webpage interactions, we have implemented automated interaction scripts for each target webpage, as illustrated in Figure 22. These scripts consist of predefined interaction code. In the example provided, the code demonstrates interactions with the first and fifth small containers on the webpage depicted in Figure 19.

It is noteworthy that the webpage interactions are element-based. For instance, identifying the fifth element with the class name “cup” (see Figure 22) necessitates that the model correctly assigns the specified class name, ID, or other attributes. In the task description, we explicitly instruct the model to set the appropriate attribute names to facilitate automated testing.

If the model incorrectly sets these class names, it will result in a failure of the automated interaction. Consequently, this action will be considered a failure when calculating the score, corresponding to an “interaction error” as defined in the main text.

G.1.3. Task descriptions

Figure 21 illustrates the task description for reproducing the webpage shown in Figure 19. We visualize this using markdown format. When provided as input to the large model, the task description is an interleaved image-text document. It details the functionality of the target webpage, includes webpage screenshots, various resources, and specifies class name requirements for automated testing.

G.1.4. Task prompts p_{cot}, p_{io}

In this section, we will present several examples of input-output prompts, as well as the detailed and complete input format used during the model invocation process.



Figure 19. A sample web page is provided where, in this example, clicking the small container button below will inject the corresponding water into the large container.

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Drink Water</title>
</head>

<body>
  <h1 data-filter-by="text" data-evalby="text|font-size|font-weight|color">Drink Water</h1>
  <h3 data-filter-by="text" data-evalby="text|font-size|font-weight|color">Goal: 2 Liters</h3>

  <div class="cup" data-filter-by="border-bottom-right-radius"
    data-evalby="border-bottom-right-radius|border-right-color|border-right-width|background-color|height|width">
    <div class="remained" id="remained">
      <span id="liters" data-filter-by="has_text" data-evalby="font-size|font-weight|color"></span>
      <small data-filter-by="text" data-evalby="text|font-size|font-weight|color">Remained</small>
    </div>

    <div class="percentage" id="percentage" data-filter-by="has_text"
      data-evalby="font-size|font-weight|color|background-color|height"></div>
  </div>

  <p class="text" data-filter-by="text" data-evalby="text|font-size|font-weight|color">Select how many glasses of water
    that you have drank</p>

  <div class="cups">
    <div class="cup cup-small" data-filter-by="text"
      data-evalby="text|font-size|font-weight|color|background-color|border-bottom-right-radius|height|width">250 ml
    </div>
    <div class="cup cup-small" data-filter-by="text"
      data-evalby="text|font-size|font-weight|color|background-color|border-bottom-right-radius|height|width">250 ml
    </div>
  </div>

  <script src="script.js"></script>
</body>
</html>

```

Figure 20. An example of preprocessed HTML file.

Task Description: Drink Water Webpage

Your job is to design a webpage that helps users track their daily water intake. The webpage should have interactive elements that allow users to select the number of glasses of water they have consumed. Below are the detailed requirements and resources needed to re-implement the webpage.

Initial Webpage

The initial webpage should look like this:



This screenshot is rendered under a resolution of 1920x1080.

Resources

Fonts

- Use the Montserrat font from Google Fonts:
`!font_uf1(https://fonts.googleapis.com/css?family=Montserrat:400,400i,400i,400i,400i,400i,400i,400i)`

Text Content

- The main heading text is "Drink Water".
- The subheading text is "Goal: 2 Liters".
- The instruction text is "Select how many glasses of water that you have drank".
- Each small cup has the text "250 ml".

Layout and Styling

Body

- Use a flexbox layout to center the content vertically and horizontally.
- Use the Montserrat font family.

Cups Container

- Use class name `cups` for the container of small cups.
- The container should use flexbox to align and justify the small cups in the center.

Remained Section

- Use ID `remained` for the remaining water section.
- The section should be centered with flexbox and should have a transition effect of `0.3s ease`.

Percentage Section

- Use ID `percentage` for the percentage section.
- The section should have a transition effect of `0.3s ease`.

Interactions

JavaScript

- Use the provided JavaScript code to handle the interactions.
- The small cups should be clickable and should fill up when clicked.
- The main cup should update to show the percentage of water consumed and the remaining liters.

Screenshots After Interactions

1. After clicking the first cup:



2. After clicking the fifth cup:



These screenshots are rendered under a resolution of 1920x1080.

Summary

- Design a webpage with a main heading, subheading, and instruction text.
- Create a main cup and eight small cups for tracking water intake.
- Implement the provided JavaScript to handle interactions.

Figure 21. An example of task description document.

```
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
import time

def click_first_cup(driver):
    first_cup = driver.find_element(By.CSS_SELECTOR, '.cup:nth-child(1)')
    first_cup.click()
    time.sleep(1)

def click_fifth_cup(driver):
    fifth_cup = driver.find_element(By.CSS_SELECTOR, '.cup:nth-child(5)')
    fifth_cup.click()
    time.sleep(1)

actions = [click_first_cup, click_fifth_cup]

def interact(driver, dir):
    # Save the original UI
    driver.save_screenshot(f'{dir}/_images/origin.png')

    # Perform actions and save screenshots after each action
    for i, action in enumerate(actions):
        action(driver)
        driver.save_screenshot(f'{dir}/_images/after_action_{i+1}.png')
```

Figure 22. An example python file for interaction.

Global setting. The prompt format for WebUI-Global setting is shown as follow:

```
{SYSTEM PROMPT}

{TASK DESCRIPTION}
```

Generate all necessary codes at a time. Your output should contain code blocks in markdown format, e.g.,
``html\nxxxx``. generate javascript and css code if necessary. They will be written to `index.html`, `script.js` and `style.css`.

The content of {SYSTEM PROMPT} should be replaced with that from Sec. A.1.2. Conversely, the {TASK DESCRIPTION} is similar to the document depicted in Figure 21. Upon generating the model output, we directly identify code snippets, using regular expressions. These

code snippets are then stored in the corresponding project folder.

Online setting. Under this setting, we start with the webpage generated by the model under the Global setting. This approach allows us to fairly compare the improvements made by different online prompts to the same initial code, ensuring a fairer comparison. Specifically, when running the Online setting, we already have an initial code-base. The model then modifies this code based on the initial code and the images rendered by the browser. There are 3 online prompt methods mentioned in the main part: "0s-CoT", "HD-CoT" and "HD-CoT-NoIMG". We will show their prompt implementation separately.

0s-CoT:

```
{SYSTEM PROMPT}

{TASK DESCRIPTION}

Current implementation:
``html
{PREV HTML CODE}
``
``css
{PREV CSS CODE}
``
``javascript
{PREV JAVASCRIPT CODE}
``

{RANDER FEEDBACK IMAGE}
```

Please adapt the code according to the current rendered screen shot of previous implementation, with this output format:

```
# Analyze
Your Analyze
# Regenerate
re-generated code
```

If some of the files (e.g., javascript) do not need to be changed, just write '*javascript do not need to change*' and do not generate any javascript code, the same for html and css.

The {RENDER FEEDBACK IMAGE} is a PNG image that represents the result of the current code being rendered in the browser. As you can see, the prompt ask the model to analyze, but do not point out how to, thus we call this prompt as "0-shot".

HD-CoT:

```
{SYSTEM PROMPT}

{TASK DESCRIPTION}

Current implementation:
```html
{PREV HTML CODE}
```

```css
{PREV CSS CODE}
```

```javascript
{PREV JAVASCRIPT CODE}
```

{RANDEr FEEDBACK IMAGE}
```

The above is an image of the previous implementation webpage screen shot. Adapt the previous code with the following step by step flow and output format:

```
# Analyze
answer: 1. What is the difference between the required webpage screen shot and the previous implementation webpage screen shot.
```

2. which parts of the previous code need to change and why.

```
# Regenerate
re-generate the files (html, css or javascript) that need to be changed.
```

If some of the files (e.g., javascript) do not need to be changed, just write '*javascript do not need to change*' and do not generate any javascript code, the same for html and css.

HD-CoT-NoIMG:

```
{SYSTEM PROMPT}

{TASK DESCRIPTION}

Current implementation:
```html
{PREV HTML CODE}
```

```css
{PREV CSS CODE}
```

```javascript
{PREV JAVASCRIPT CODE}
```

The above is the previous implementation webpage. Self reflect on the previous implementation and adapt the previous code with the following step by step flow and output format:
```

```
# Analyze
Analyze and self-reflect: 1. Are all the requirements in the description correctly fulfilled.
2. If not, which parts of the previous code need to change and why.
```

```
# Regenerate
re-generate the files (html, css or javascript) that need to be changed.
If some of the files (e.g., javascript) do not need to be changed, just write '*javascript do not need to change*' and do not generate any javascript code, the same for html and css.
```

Sure! Below are the necessary code blocks for `index.html`, `style.css`, and `script.js` to create the Drink Water webpage as described.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Drink Water</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <h3>Drink Water</h3>
    <h3>Goal: 2 Liters</h3>

    <!-- Main Cup -->
    <div class="cup-container">
      <div class="cup" id="remained">2L<br />Remained</div>
      <div class="percentage" id="percentage"></div>
    </div>

    <!-- Instruction Text -->
    <p>Select how many glasses of water that you have drank</p>

    <!-- Small Cups -->
    <div class="cups">
      <!-- Eight small cups each representing 250ml -->
      ${[...Array(8)].map((L, i) => `<div class="cup-small">250ml</div>`),join("")}
    </div>

    <!-- JavaScript File Link -->
    <script src="script.js"></script>
  </body>
</html>
```

style.css

```
@import url('https://fonts.googleapis.com/css?family=Montserrat:400,600&display=swap');

* {
  box-sizing: border-box;
}

body {
  background-color: #3498db;
  color: white;
  font-family: 'Montserrat', sans-serif;
```

```
display: flex;
justify-content: center;
align-items: center;
height: 100vh;
}

.container {
  text-align: center;
}

.cup-container {
  position: relative;
  margin-bottom: 20px;
}

.cup {
  width: 150px;
  height: 150px;
  background-color: white;
  border-radius: 50px;
  margin: auto;
}

.percentage {
  position: absolute;
  bottom: 0;
  left: 0;
  right: 0;
  height: 0;
  background-color: #3498db;
  transition: 3s ease;
}

#remained {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}

.cups {
  display: flex;
  justify-content: center;
  flex-wrap: wrap;
}

.cup-small {
  width: 60px;
  height: 60px;
  background-color: white;
  border-radius: 10px;
  margin: 5px auto;
}
```

script.js

```
const smallCups = document.querySelectorAll('.cup-small');
const liters = document.getElementById('liters');
const percentage = document.getElementById('percentage');
const remained = document.getElementById('remained');

updateBigCup();

smallCups.forEach((cup, idx) => {
  cup.addEventListener('click', () => highlightCups(idx));
});

function highlightCups(idx) {
  if (smallCups[idx].classList.contains("full") && !smallCups[idx].nextElementSibling.classList.contains("full")) { idx--; }
  smallCups.forEach((cup, idx2) => {
    if (idx2 <= idx) { cup.classList.add("full"); } else { cup.classList.remove("full"); }
  });
  updateBigCup();
}

function updateBigCup() {
  const fullCups = document.querySelectorAll('.cup-small.full').length;
  const totalCups = smallCups.length;

  if (full === total) { remained.style.visibility = 'hidden'; remained.style.height = '0'; } else { remained.style.visibility = 'visible'; liters.innerText = `${(250 * full / 100)}L`; }

  if (full === total) { percentage.style.visibility = 'hidden'; percentage.style.height = 'auto'; } else { percentage.innerText = `${(100 * full / total)}%`; percentage.style.height = `${(330 * full / total)}px`; }
}
```



Figure 23. GPT-4o’s output text(Upper image) and screen shot of GPT-4o’s implementation(Lower image). This implementation has a score of $AE_S = 29.23$

G.1.5. Output examples

Figures 23 and 24 illustrate the outputs of the GPT-4o and LLaMA-3.2-90B models, respectively, along with the cor-

responding website renderings. Please refer to the previous task description in conjunction with this.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Drink Water</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <h1>Drink Water</h1>
    <h2>Goal: 2 Liters</h2>
    <div id="remained"></div>
    <div id="percentage"></div>
    <div class="cups">
      <div class="cup cup-1">
        <span>250ml</span>
      </div>
      <div class="cup cup-2">
        <span>250ml</span>
      </div>
      <div class="cup cup-3">
        <span>250ml</span>
      </div>
      <div class="cup cup-4">
        <span>250ml</span>
      </div>
      <div class="cup cup-5">
        <span>250ml</span>
      </div>
      <div class="cup cup-6">
        <span>250ml</span>
      </div>
      <div class="cup cup-7">
        <span>250ml</span>
      </div>
      <div class="cup cup-8">
        <span>250ml</span>
      </div>
    </div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

style.css

```
@import url('https://fonts.googleapis.com/css?
family=Montserrat:400,600&display=swap');
```

```
body {
  font-family: 'Montserrat', sans-serif;
  background-color: #f0f0f0;
}

.container {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100vh;
  margin: 0 auto;
  max-width: 400px;
}

h1 {
  font-size: 36px;
  font-weight: 600;
  margin-bottom: 10px;
}

h2 {
  font-size: 18px;
  color: #666;
  margin-bottom: 20px;
}

#remained {
  font-size: 24px;
  font-weight: 600;
  margin-bottom: 10px;
}

#percentage {
  font-size: 18px;
  color: #666;
  margin-bottom: 20px;
}

.cups {
  display: flex;
  justify-content: space-between;
  width: 100%;
}

.cup {
  background-color: #fff;
  border-radius: 10px;
  padding: 20px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

.cup span {
  font-size: 14px;
  color: #666;
}
```

```
.cup:hover {
  background-color: #ccc;
}

.cup-filled {
  background-color: #007bff;
}

.cup-filled span {
  color: #fff;
}
```

script.js

```
const cups = document.querySelectorAll('.cup');
const remained = document.getElementById('remained');
const percentage = document.getElementById('percentage');

let totalCups = cups.length;
let filledCups = 0;

cups.forEach((cup, index) => {
  cup.addEventListener('click', () => {
    if (!cup.classList.contains('cup-filled')) {
      cup.classList.add('cup-filled');
      filledCups++;
      updateRemainedAndPercentage();
    } else {
      cup.classList.remove('cup-filled');
      filledCups--;
      updateRemainedAndPercentage();
    }
  });
});

function updateRemainedAndPercentage() {
  const remainedLiters = 2 - (filledCups * 0.25);
  const percentageFilled = (filledCups / totalCups) * 100;

  remained.textContent = `${remainedLiters}L`;
  percentage.textContent = `${percentageFilled}%`;
}
```

This code creates a simple webpage with a main heading, subheading, and instruction text. It also includes a main cup and eight small cups for tracking water intake. The JavaScript code handles the interactions, filling up the small cups when clicked and updating the main cup to show the percentage of water consumed and the remaining liters.

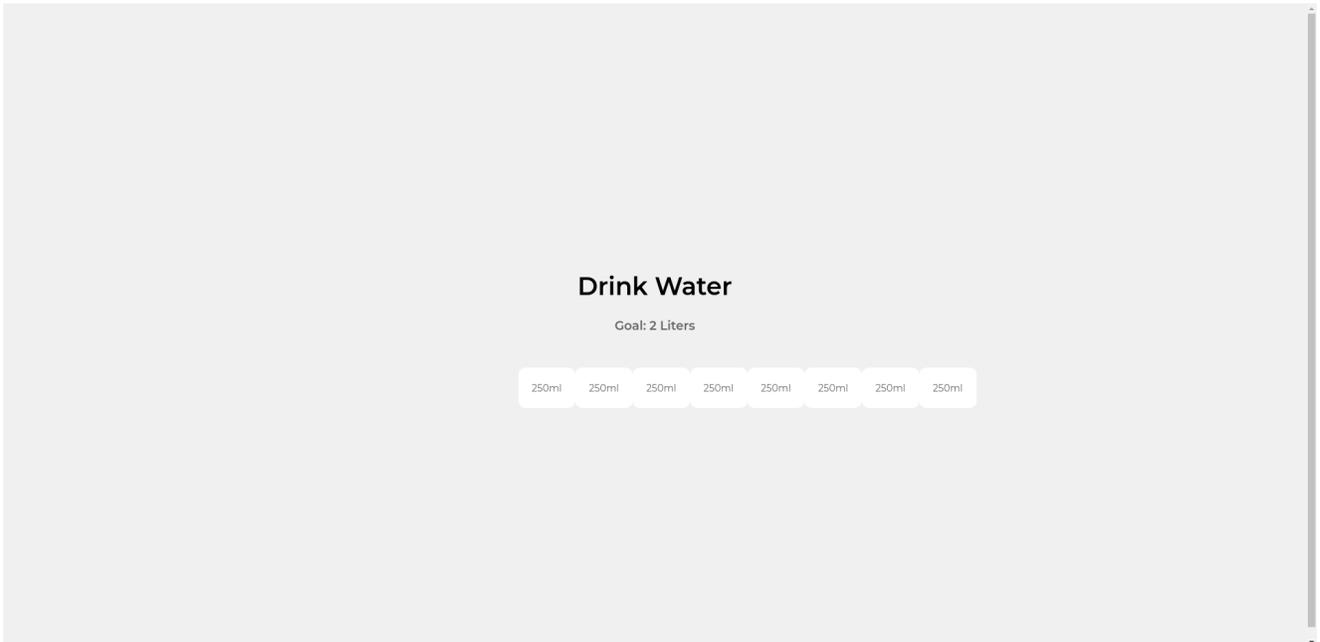


Figure 24. LLaMA-3.2-Vision-Instruct's output text(Upper image) and screen shot of LLaMA-3.2-Vision-Instruct's implementation(Lower image). This implementation has a score of $AES = 9.59$

G.2. Sokoban

G.2.1. Task prompts p_{cot}, p_{io}

Global setting. The prompt format of Sokoban-Global setting is shown as follow:

```
{SYSTEM PROMPT}
```

```
{INITIAL FRAME}
```

```
Please analyze and directly output a list of actions to take.
```

```
Generate all actions to fulfill the task at a time. Your output should follow this format:
```

```
### Analyze
your analyze.
### Actions
A long sequence of Left, Right, Up, Down, separate by ','
```

The same as WebUI, the content of {SYSTEM PROMPT} should be replaced with that from Sec. A.2.1. The {INITIAL FRAME} will be replaced by the image of the initial game scene. After the model generate the output, we just parse out the actions sequence and input to the environment.

Online setting. Differ from WebUI, we use a memory larger than 1 in Sokoban and football, and the memory in our codebase is implemented using multi-turns chat. In the following prompt, '—User:' and '—Assistant:' are not part of the prompt but an indicator of roles in the multi-turns chat. Now let's suppose we are at the time step T , the model needs to generate its response for action at time step $T + 1$, and we use the standard settings: $AM = 5, OM = 1$.

```
-----User:
{SYSTEM PROMPT}
image not available.
```

```
You should perform 1 action for each generation, with this format:
```

```
# analyze
some analyze
# action
action ID
```

```
For example:
# analyze
...
# action
Up
```

```
-----Assistant:
{output at T-4}
```

```
-----User:
Please continue to act according to the current game scene.
image not available.
```

```
-----Assistant:
{output at T-3}
```

```
-----User:
Please continue to act according to the current game scene.
image not available.
```

```
-----Assistant:
{output at T-2}
```

```
-----User:
Please continue to act according to the current game scene.
image not available.
```

```
-----Assistant:
{output at T-1}
```

```
-----User:
Please continue to act according to the current game scene.
image not available.
```

```
-----Assistant:
{output at T}
```

```
-----User:
Please continue to act according to the current game scene.
{FRAME T}
```

Here, as $OM = 1$, the only game scene inputted is the {FRAME T} part.

G.2.2. Output examples

Figures 25 and 26 present several examples within the Sokoban-Global environment. Figure 25 illustrates both successful and unsuccessful cases of the GPT-4o model within the same scenario. Figure 26, on the other hand, compares the performance of different models within the identical scenario. Figure 27 depicts the entire process under the Online setting.

GPT-4o

Analyze
The green character needs to push the yellow box into the target area with a red dot in the center. The current position of the green character is to the right of the yellow box, and there are two target areas: one above and one below. To solve this puzzle, we need to move around and push the box into either of these target areas.

Actions
Up, Left, Down

Overall reward: 41.0

GPT-4o

Analyze
The green character needs to push the yellow box into the target area with a red dot in the center. The current position of the green character is to the right of the yellow box, and there are two target areas: one above and one below. To solve this puzzle, we need to move the yellow box downwards into either of these target areas.

Actions
Left, Down

Overall reward: 100.0

Figure 25. An example of GPT-4o’s performances on Sokoban-Global settings, failed case (Left) and success case (Right).

GPT-4o

Analyze
The green character needs to push the yellow box into the target area with a red dot in the center. The current position of the green character is above one of the boxes, and there are two other boxes blocking direct access to any target areas. The strategy will involve moving these boxes out of the way and then pushing one box into a target area.

Actions
Down, Left, Up, Right, Down

Overall reward: 44.5

Qwen2-VL-72B

Analyze
The current state of the game shows the green character positioned next to a yellow box. The target area is marked by red dots, and there are walls surrounding the play area. To push the box into the target area, we need to maneuver it carefully without getting stuck in a corner or against a wall.

Actions
Left, Left, Up, Up

Overall reward: 41.5

Figure 26. An example of GPT-4o’s and Qwen2-VL-72B’s performances on Sokoban-Global settings.

G.3. Football

G.3.1. Task prompts p_{cot}, p_{io}

Because football is a multi-agent system, and there are 21 players controlled by the built-in AI bot, which bring a giant amount of randomness, it is meaningless to setup Global setting for this environment. Thus, this environment only supports **Online setting**, whose prompt is exactly the same as Sokoban-Online, except that the IO prompt part is replaced with:

```
You should perform 1 action for each
generation, with this format:
# analyze
some analyze
# action
action ID
```

```
For example:
# analyze
...
# action
action_left
```

G.3.2. Output examples

Figure 28 shows the output and images of GPT-4o under the Football-Online setting. In the video, the player successfully dribbles past opponents, then passes the ball, and finally scores a goal.

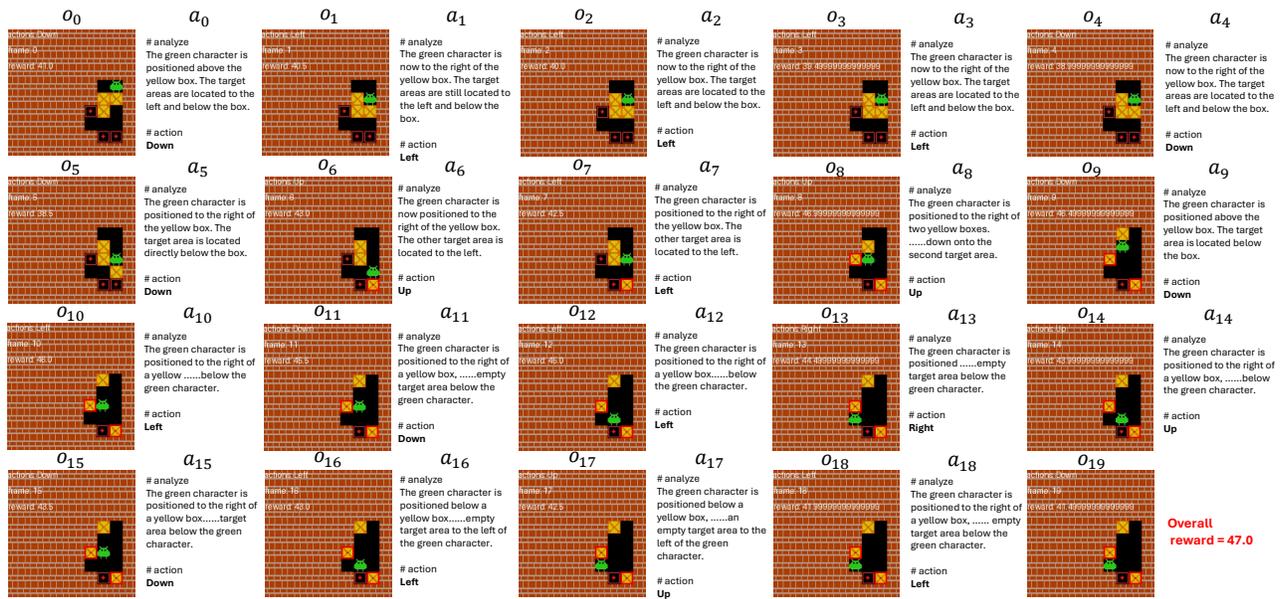


Figure 27. An example of GPT-4o's performances on Sokoban-Online settings with max-loop=20.

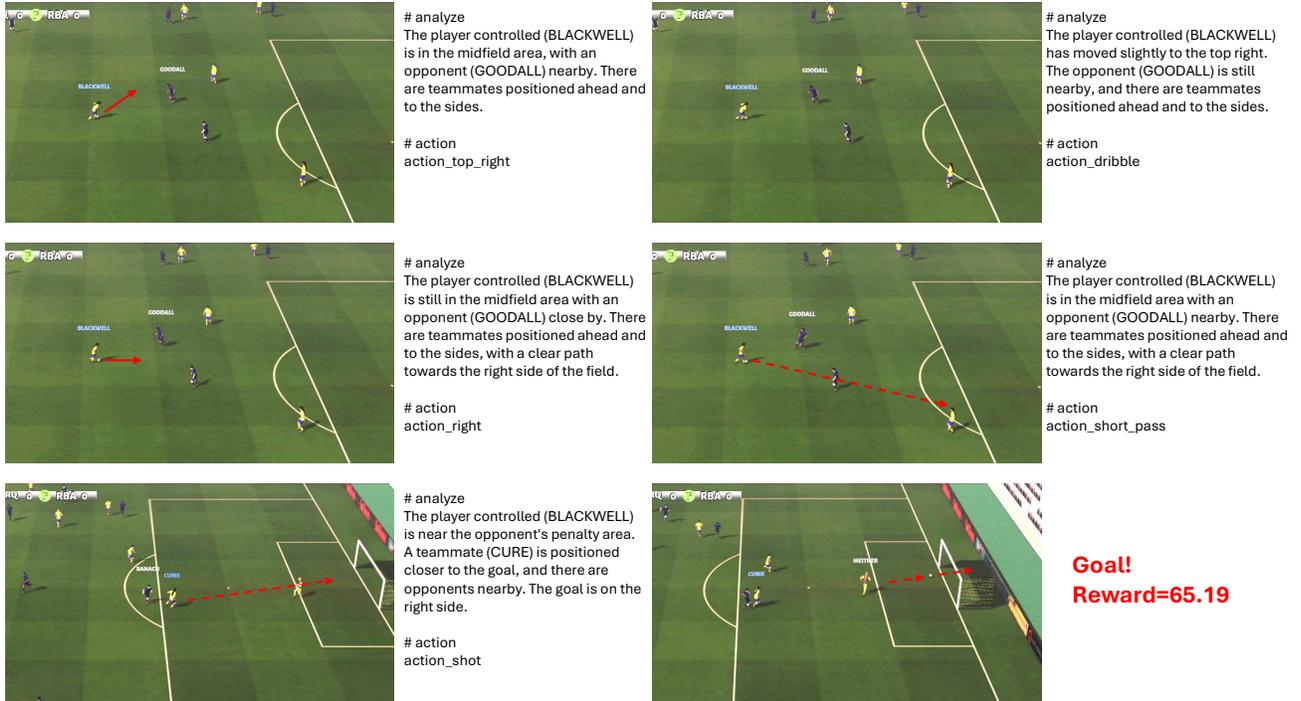


Figure 28. An example of GPT-4o's performances on Football-Online settings.