SegAgent: Exploring Pixel Understanding Capabilities in MLLMs by Imitating Human Annotator Trajectories

Supplementary Material

A. Discussion

A.1. Computational complexity Trade-off

We agree that error accumulation is a potential issue, and longer sequences may lead to worse results if we perform fixed-length greedy decoding. However, the proposed PRM can effectively alleviate this issue. The experimental results are as follows.

Table 2. Performance comparison under different steps.

# Steps	1	3	5	7
w/o PRM	71.53	73.67	73.88	68.22
w/ PRM	71.53	72.98	75.21	75.43

A.2. Limitations

Failure cases. Failures mainly arise from (1) incorrect target localization in the first step and (2) inaccurate coordinate outputs in refinement steps, particularly at boundaries (wrong point reaching the background area), resulting in the wrong mask.(see Fig. 8)

We believe that the current bottleneck limiting the model's performance lies mainly in the fine-grained pixel-level localization ability of MLLMs. On the HRES dataset, the main reason for the model's failure is the inaccurate location of the output points, such as the wrong point being outside the object's boundary.

B. More Details about algorithm

B.1. Trace Generation

In Section 4.1 of the main text, we introduced how F_{sim} is used to simulate user trajectories. Here, we provide a more detailed illustration of this process in the form of Python pseudocode (see Alg. 3).

To further enhance the diversity of annotation trajectories and improve SegAgent's mask refinement capabilities, we introduced additional initial states beyond the empty mask. These include masks generated from bounding boxes and masks created by random sampling points.

B.2. Comparision with StaR

As mentioned in Section 4.3 of the main text, we have introduced the differences between our method StaR+ and the original StaR algorithm [70]. Here we first present the original StaR algorithm (see Alg. 4). In addition to the differences mentioned in the main text, another difference is that the original StaR algorithm will only fine-tune the model with the newly generated trajectory dataset D_n, while our StaR+ will fine-tune the model with the merged dataset of D_{traj} and D_n. This is because the original StaR algorithm is designed for reasoning tasks that lack a simulate function F_{sim} to generate trajectories. In our task, D_{traj} retains a lot of useful information because it generates approximately optimal trajectories. Therefore, we merge it with D_n to fine-tune the model.

Algorithm 3 Pseudo code of trace generation

```
# n: Maximum number of clicks
# image: The image to be segmented
# gt_mask: Ground truth mask of current image
# pred_mask: The predicted mask according to
    clicks in click_list

click_list = []
pred_mask = np.zeros_like(image)

# Iterate n times
for i in [1, n]:
   fn = gt_mask & (~pred_mask) # false negative
   fp = (~gt_mask) & pred_mask # false positive
   fn_dist = cv2.distanceTransform(fn_mask)
   fp_dist = cv2.distanceTransform(fp_mask)
   fn_max_dist = max(fn_dist)
   fp_max_dist = max(fp_dist)

   if (fn_max_dist > fp_max_dist): # Next click
       should be positive
      click_y, click_x = np.where(fn_dist ==
          fn_max_dist)
      is_positive = 1
   else: # Next click should be negative
      click_y, click_x = np.where(fp_dist ==
          fp_max_dist)
      is_positive = 0

   click_list.append((click_x, click_y,
       is_positive))
   pred_mask = model.predict(image, click_list)
```

Algorithm 4 SegAgent Policy Improvement with StaR

Input: The SegAgent model trained on the generated trajectory dataset $D_{\text{traj}} = \{(I^i, M^i_{\text{target}}, P^i, \mathcal{T}^i)\}_{i=1}^m$
$S_0 \leftarrow$ SegAgent
// Initialize the SegAgent model
$D_0 \leftarrow D_{\text{traj}}$
// Initialize the trajectory dataset
for $n = 1$ to N **do**
  $\hat{\mathcal{T}}^i \leftarrow S_{n-1}(I^i, M^i_{\text{target}}, P^i)$ for all $i \in [1, m]$
  // Perform trajectory generation
  $D_n \leftarrow \{(I^i, M^i_{\text{target}}, P^i, \textbf{Filter}(\hat{\mathcal{T}}^i)) \mid i \in [1, m]\}$
  // Filter trajectories based on the reward function
  $S_n \leftarrow \text{train}(S_0, D_n)$
  // Fine-tune the model on the filtered dataset
end for

B.3. More Details about Process Reward Model and Tree Search

In Section 4.4 of the main text, we have shown the process of PRM-guided tree search in the form of pseudocode. To facilitate readers' understanding, we further illustrate this process in Fig. 5

C. Implementation Details

C.1. Model Architecture and Hyperparameters

For SegAgent-LLaVA, we initialized the model with project weights provided by [69]. Subsequently, we performed a second-stage fine-tuning using the annotation trajectories generated in our framework. Following the implementation in [69], we adopted a ConvNeXt-L [34] CLIP model as the vision encoder, extracting image features from the "res4" stage. The model was trained using the AdamW [36] optimizer with a learning rate of 1×10^{-5} and a cosine annealing scheduler [35] for two epochs. We set the batch size to 16. During both training and inference, input images were resized to 512×512. The maximum sequence length was set to 2048 tokens.

For SegAgent-QWen, we initialized the model using the Qwen-VL-Chat weights provided in the official implementation [2]. Fine-tuning was conducted using the full-parameter fine-tuning script provided by the authors, with only the ViT module frozen. Specifically, input images were resized to 448×448, and 256 queries were used for the vision-language adapter. The model was trained using the AdamW optimizer with a learning rate of 1×10^{-5}, a cosine decay learning rate schedule, and a batch size of 128 for two epochs. The maximum sequence length was set to 2048 tokens.

For SAM [17] and SimpleClick [30], we used the official pre-trained weights provided by their respective repositories. Both models are based on a ViT-large architecture.

C.2. Prompt Design

Here we provide a detailed introduction to the specific design of the input prompt P for MLLMs, as shown in Fig. 6. The design of the prompt is to guide the model to generate more accurate annotations, including two operations: adding a positive point, adding a negative point. Adding a positive point is to expand the mask, and adding a negative point is to shrink the mask.

D. Visualization Analysis

D.1. Comparison of Dataset Quality

In Section 5.2 of the main text, we quantitatively analyzed the complexity of different datasets. Here we now provide a qualitative comparison of dataset quality through visualization. Fig. 7 illustrates examples of images and annotations from various datasets, allowing readers to gain a deeper understanding of the characteristics of each dataset.

From the visualization, we can observe that the annotation masks in ThinObject5k-TE and DIS5K are indeed more complex and precise. For instance, in the "Bridge" and "Sailboat" examples from DIS5K, the annotations exhibit intricate details such as hollow structures and fine lines. These characteristics highlight the high annotation quality and attention to detail in these datasets.

In contrast, RefCOCO primarily focuses on scenes with people and common objects. Although the captions are longer, the annotations contain more noise. For example, while the masks roughly cover the objects, there are significant issues with mislabeling and omissions at the edges. Additionally, RefCOCO struggles to handle intricate details such as hollow regions effectively.

In summary, ThinObject5k-TE and DIS5K offer higher-quality and more complex annotations, making them better suited for evaluating and exploring SegAgent's ability to refine masks over multiple steps.

D.2. Visualization of Predicted Trajectories

We visualized the original predicted trajectories of SegAgent, as shown in Fig. 8. Note that PRM and Tree Search were not used in this visualization. The first two rows show the results of using Qwen-box as the first action combined with SAM for mask refinement. Although we visualized the clicks at Iteration 0, the first click was not actually input to SAM. In subsequent iterations, we used the clicks predicted by SegAgent and Qwen-box together as input to SAM. The last two rows show the results of using only clicks as actions combined with SimpleClick for mask annotation. It can be observed that SegAgent has indeed learned the rules of annotation and acquired an understanding of objects. It can refine masks step by step through positive and negative points.

E. More Experiments

E.1. Annotation Filtering

In Section 5.3 of the main text, we analyzed SegAgent's capabilities in mask annotation and mask refinement. Here, we further explore and demonstrate its ability in annotation filtering.

We model annotation filtering as a regression task, where the model predicts the Intersection over Union (IoU) between the current mask and the ground truth (GT) mask. This functionality is a key feature of SegAgent's PRM. In practice, by setting a reasonable threshold, we can effectively filter out low-quality masks.

To evaluate this ability, we constructed a test set based on the validation set of RefCOCO. Specifically, we generated masks of varying quality by randomly sampling posi-

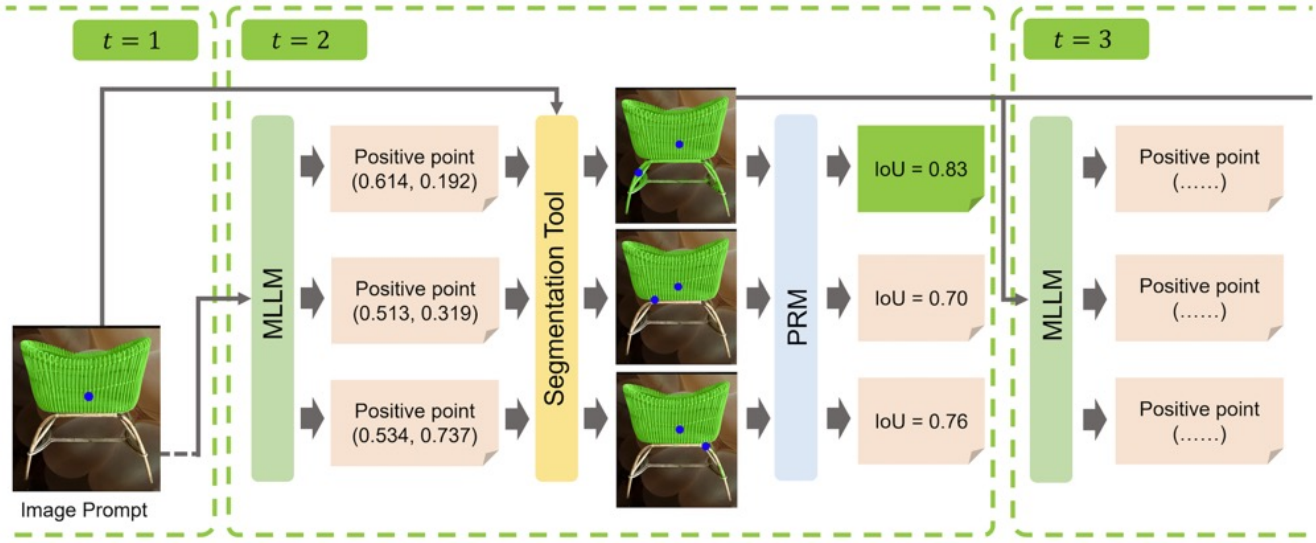

Figure 5. An illustrative example of PRM-guided tree search. The model predicts the reward at each step and selects the action with the highest reward to generate the next mask.

Prompt Design for SegAgent

You are a highly skilled segmentation annotator. We have provided you with an image and an initial mask marked by a **semi-transparent green mask** that roughly covers the object described below. Your task is to refine this mask to make it as accurate as possible. Based on the given image and the mask, perform the following actions:

1. Positive Point (x, y):
Add a positive point if any part of the object is not covered by the mask. This will expand the mask to include the missing area. *Example:* Add a positive point on any corner or edge of the object that the mask does not cover.

2. Negative Point (x, y):
Add a negative point if an area outside the object is incorrectly included in the mask. This will refine the mask by excluding unnecessary regions. *Example:* Add a negative point where the mask extends into the background or any non-object area.

The description of the object is as follows: <description>.

Figure 6. The prompt provides detailed instructions for refining a segmentation mask with three possible actions: adding a positive point, adding a negative point. The red part indicates user-specific input, such as object descriptions.

tive and negative points within the GT bounding boxes. The PRM was then used to predict the IoU of these masks.

We assessed the annotation filtering capability of the PRM using several standard regression metrics, including Mean Squared Error (MSE), Mean Absolute Error (MAE), Pearson Correlation Coefficient, and Spearman Correlation Coefficient.

Based on the results in Tab. 3, SegAgent-LLaVA outperforms SegAgent-Qwen across all evaluated metrics, indicating its superior ability in annotation filtering. These results are consistent with the analysis in Section 5.3 of the main text regarding the mask refinement capability. We hypothesize that the differences in performance may stem from the distinct model architectures. Specifically, the Q-former structure in SegAgent-Qwen might lead to some loss of detail, which could explain its slightly inferior performance compared to SegAgent-LLaVA.

However, from an overall perspective, both SegAgent-

Table 3. **Evaluation of SegAgent's Annotation Filtering Ability.** Lower MAE and MSE indicate better accuracy, while higher Pearson and Spearman correlation coefficients reflect stronger agreement with ground truth IoU.

Model	MAE ↓	MSE ↓	Pearson ↑	Spearman ↑
SegAgent-Qwen	6.88	193.98	0.90	0.87
SegAgent-LLaVA	5.58	175.35	0.91	0.90

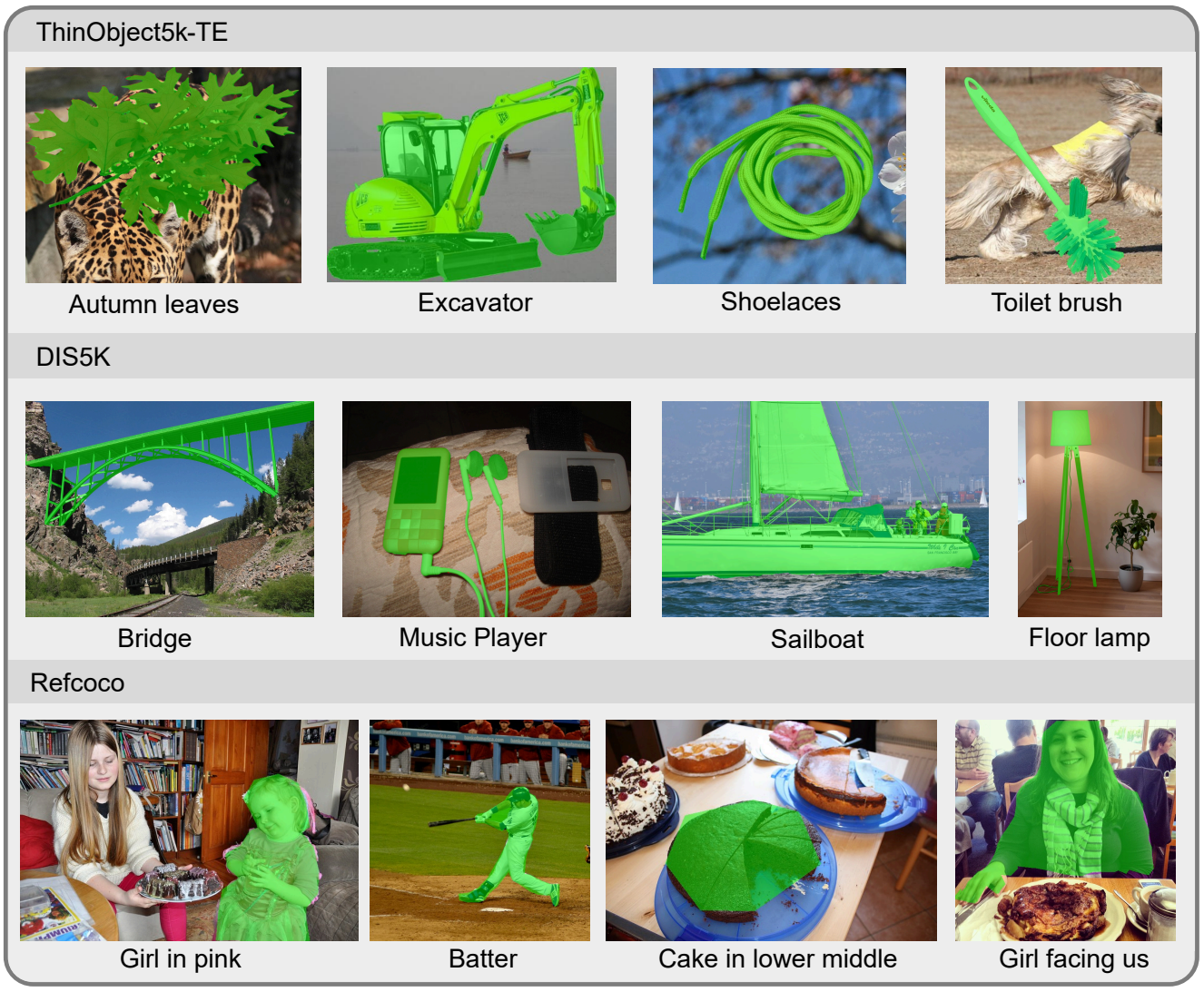

Figure 7. **Examples of Images and Annotations from Various Datasets.** The figure showcases representative samples from three datasets: ThinObject5k-TE, DIS5K, and RefCOCO. Each row represents a dataset, with images and corresponding annotations highlighting different objects and scenes. The annotations (green overlays) demonstrate the varying levels of detail and complexity across datasets.

LLaVA and SegAgent-Qwen exhibit high correlation coefficients and relatively low MAE and MSE. This indicates that the PRM is highly effective in predicting the mIoU of masks, enabling the filtering of low-quality masks with strong reliability.

E.2. Mask color

By default, we visualize the current segmentation results using a semi-transparent green mask. Here, we further investigate the impact of mask color on segmentation performance. The results in Table 4 show the impact of mask color on segmentation performance, measured by mean Intersection over Union (mIoU). The three tested mask colors—green, blue, and red—yield nearly identical performance. This suggests that the choice of mask color has minimal, if any, effect on segmentation performance. The consistent mIoU across different colors indicates that the model's segmentation capability is robust to visual variations in mask color.

Table 4. **Evaluation of Mask Color on Segmentation Performance.**

Mask Color	Green	Blue	Red
mIoU	0.749	0.750	0.749

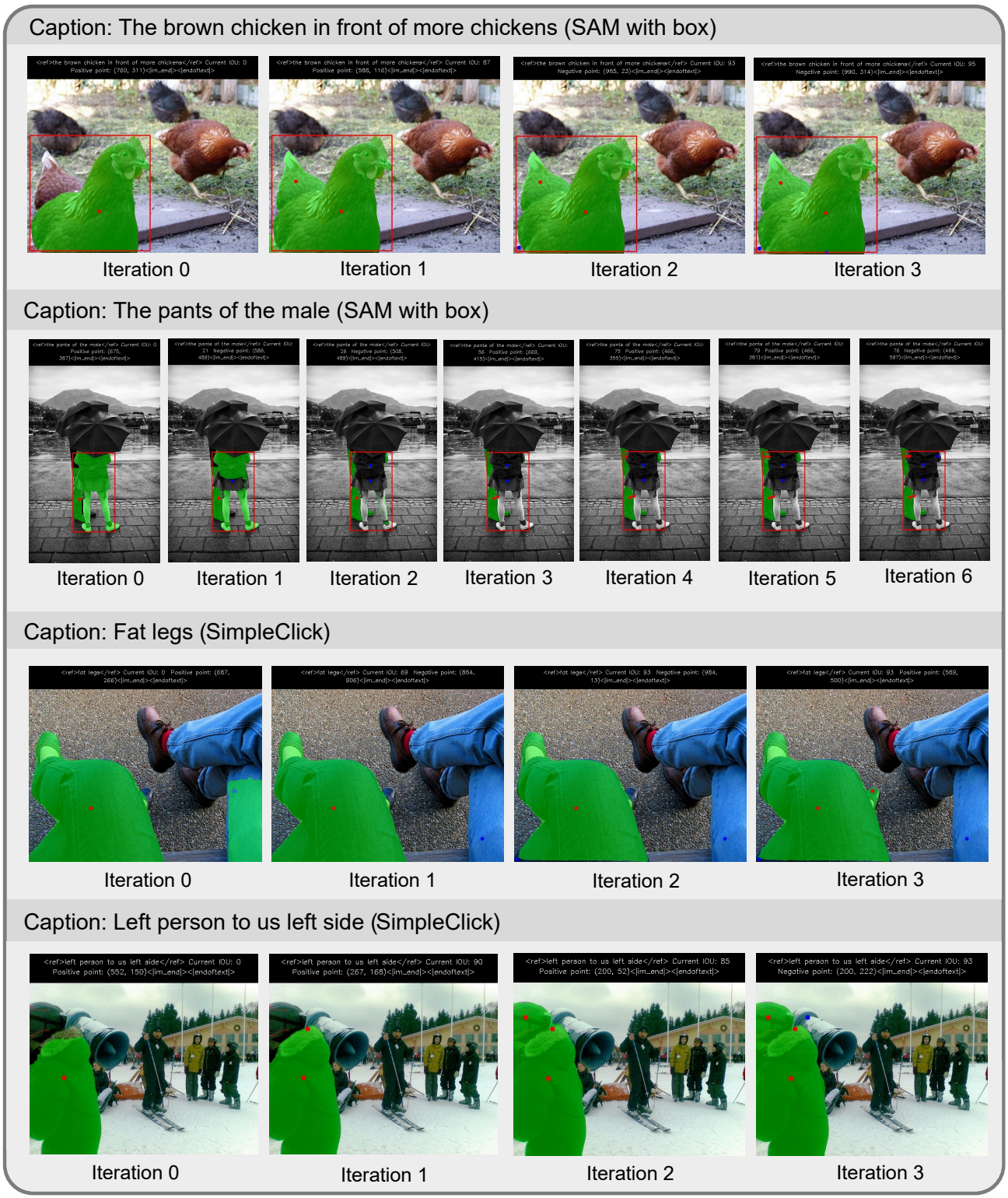

Figure 8. **Predicted trajectories of SegAgent using SAM and SimpleClick.** We visualize current action a_t and the resulting mask M_{t+1} in one image. Red points represent positive points, and blue points represent negative points.

Table 5. **Evaluation of Different Initial Actions on SegAgent Performance.**

Initial Action	NO box	Qwen Box	Self Box
refcoco(val)	77.81	78.01	77.85
refcoco+(val)	70.88	70.86	70.50
refcocog(test)	73.13	74.62	74.33

E.3. Init Action

Since SAM can accept both boxes and clicks as input, we investigated the impact of different initial actions on segmentation performance. In Table 5, "NO box" indicates using only clicks as actions, "Qwen Box" represents using the box predicted by Qwen-VL-chat as the action, and "Self Box" denotes using the box predicted by SegAgent-Qwen itself as the action (an additional task during training). The results indicate that the choice of initial action has a minimal impact on segmentation performance, suggesting that the model is robust to the selection of initial actions. Overall, using Qwen Box as the initial action yields slightly better performance than the other two initial actions. To ensure a fair comparison, we selected Qwen Box as the initial action for SAM.

E.4. Coordinate Format

We also investigated the representation of coordinates. For SegAgent-Qwen, we used the [0, 1000) format to represent the coordinates of bounding boxes, as Qwen itself has grounding capabilities. For SegAgent-LLaVA, we explored whether to use integers in the range [0, 1000) to represent relative positions or decimals in the range [0, 1). Table 6 shows the impact of relative position representation on segmentation performance. The results indicate that the two coordinate formats yield nearly identical performance, suggesting that the model is robust to the choice of coordinate format. For SegAgent-LLaVA, we selected decimals in the range [0, 1) to represent relative positions.

Table 6. **Evaluation of Coordinate Format on Segmentation Performance.**

Coordinate Format	[0, 1)	[0, 1000)
mIoU	0.749	0.747