



# PixelLM: Pixel Reasoning with Large Multimodal Model

## Supplementary Material

In this supplementary material, we first detail the training configuration, decoder structure, flops calculation, and MUSE evaluation process in Sec.A. We then present an additional experimental analysis of our segmentation codebook and decoder in Sec.B. Furthermore, we offer a more comprehensive analysis of MUSE and the multi-referring segmentation dataset in Sec. C.

### A. Implementation Details

**Training details.** We give the detailed training configuration in Tab. A.1, and we do *not* use color jittering, drop path or gradient clip. The gradient accumulation step is set to 10. **Structure of pixel decoder.** The decoder can be divided into three parts based on their functions: *i*) the attention block for each scale; *ii*) using the output mask from one scale to modulate the features in the next scale; *iii*) the fusion of masks from all scales to obtain the final results. We present the PyTorch-style pseudocodes for the overall decoder and each part in Alg. 1.

**Calculation of TFLOPs.** We compare models’ TFlops (trillion floating-point operations per second) in Tab. 1. The calculation follows the formula in [7] and the script in DeepSpeed [23]. Since the flops for LMMs vary with the generated token length, we standardize it at 512. This length aligns with the common default used by [17, 30] and suffices to accommodate dozens of target objects.

**Details of the evaluation metric.** Sec. 4.3 provides a concise overview of the MUSE evaluation pipeline. In this section, we delve into a more formal and detailed explanation of its design. Let us denote by  $M = \{M_g\}_{g=1}^G$  the ground truth set of  $G$  objects, and  $\hat{M} = \{\hat{M}_k\}_{k=1}^K$  the set of  $K$  predictions. Motivated by [4], assuming  $K$  is not equal to  $G$ , we use  $\emptyset$  (no objects) to pad the smaller set and both sets finally have size  $P = \max(G, K)$ .

(1) We find a bipartite matching between these two sets by searching for a permutation of  $P$  elements,  $\sigma \in \mathfrak{S}_P$ , with the lowest cost:

$$\hat{\sigma} = \arg \min_{\sigma \in \mathfrak{S}_P} \sum_i^P \mathcal{L}_{match}(M_i, \hat{M}_{\sigma(i)})$$

where  $\mathcal{L}_{match}(M_i, \hat{M}_{\sigma(i)})$  is a pairwise matching cost between ground truth  $M_i$  and a prediction with index  $\sigma(i)$ . We compute this optimal assignment efficiently with the Hungarian algorithm. We define  $\mathcal{L}_{match}(M_i, \hat{M}_{\sigma(i)})$  as  $\mathcal{L}_{bce}(M_i, \hat{M}_{\sigma(i)}) + \mathcal{L}_{dice}(M_i, \hat{M}_{\sigma(i)})$ .

(2) Based on the matching results, we modify the generated response  $y_{res}$  to  $y'_{res}$ : since each  $\hat{M}_i$  originates from a seg-

config	value
optimizer	AdamW
base learning rate	3.0e-4
weight decay	0
optimizer momentum	$\beta_1, \beta_2=0.9, 0.95$
batch size	16
learning rate schedule	WarmupDecayLR
warmup iterations	100
augmentations	None
$\alpha$	2.0
$\lambda_{ref}$	2.0
$\lambda_{dice}$	0.5

Table A.1. Training settings.

Codebook design	MUSE Val		refCOCO+			refCOCOg	
	gIoU	cIoU	val	testA	testB	val(U)	test(U)
$N$	41.0	48.3	64.0	69.8	57.5	67.9	68.4
$N \times L$	<b>42.6</b>	<b>50.7</b>	<b>66.3</b>	<b>71.7</b>	<b>58.3</b>	<b>69.3</b>	<b>70.5</b>

Table A.2. Sharing tokens across  $L$  scales. The first row corresponds to results of sharing tokens across all feature scales.

Selected layers	MUSE Val		refCOCO+			refCOCOg	
	gIoU	cIoU	val	testA	testB	val(U)	test(U)
baseline	40.1	47.2	61.1	65.4	54.7	64.3	65.6
20, 17, 14	42.0	51.5	66.0	71.4	58.5	68.8	70.6
20, 17	41.2	49.1	65.9	71.2	58.0	68.0	66.4
23, 14, 10	41.8	48.0	65.1	68.3	57.9	67.5	68.0
23, 14, 20	42.3	<b>51.4</b>	66.0	71.5	<b>58.5</b>	<b>69.8</b>	70.4
23, 14	<b>42.6</b>	50.7	<b>66.3</b>	<b>71.7</b>	58.3	69.3	<b>70.5</b>

Table A.3. Multi-scale layer selection. CLIP-ViT consists of 24 layers. “Baseline” only uses the penultimate (i.e., the 23rd) layer.

mentation token sequence in  $y_{res}$ , we replace each sequence with the GPT-generated description of  $M_i$ .

(3) We use a carefully designed prompt for GPT-3.5 to assign a score  $s_i$  to each  $\hat{M}_i$  in the answer in a single step. An example of this methodology is depicted in Fig. A.1. The empty predictions are directly scored with 0.

The above three steps assess the model’s capability to generate outputs where masks are intertwined with text descriptions and evaluate how accurately these masks correspond to their respective text descriptions. Then we evaluate the quality of the masks.

(4) The final IoU of each prediction is:

$$\text{Intersection}_i = \begin{cases} \text{Intersection}_i & s_i > 0.5 \\ 0 & s_i \leq 0.5 \end{cases}$$

$$\text{IoU}_i = \text{Intersection}_i / \text{Union}_i$$

And the final  $\text{IoU}_{img}$  of each image is:

$$\text{IoU}_{img} = \sum_i \text{IoU}_i / P$$

---

**Algorithm 1:** Pseudo codes of our pixel decoder.

---

```
# Inputs: f_img:image features from L scales [L,C,H,W];
# h_seg:segmentation tokens for L scales [L, C];
# Variables: lev_token: Learnable embeddings for L scales
[L,C]; out_token:Learnable embeddings [N, C];
image_pe:position embedding of image features; gamma:mask
weighting factors [L]
# Functions: SelfAttention();CrossAttention();MLP();
up_scale(); down_scale();
1 def feature_update(f, mask):
    # f:image feature of one scale.[CxHxW]
    # mask:output from the attention block above f.[Cx4Hx4W]
2     mask = down_scale(mask, size=(HxW));# mask: [H,W]
3     f = f * (sigmoid(mask) + 1) # update feature
4     return f
5 def attention_block(h, f, l):
    # h, f:segmentation token and image feature of one
    # scale.[C], [CxHxW]
    # l:scale index
6     token = cat([out_token, h], dim=0) + lev_token[l];
    # token:[N+1,C]
7     attn_out = SelfAttention(q=k=v=token);# self attention
    output:[N+1,C]
8     token = norm(token + attn_out)
9     key = f + image_pe; # key:[C,H,W]
10    attn_out = CrossAttention(q=token, k=key, v=f);# cross
    attention output:[N+1,C]
11    token = norm(token + attn_out + MLP(token)); # update
    token:[N+1,C]
12    attn_out = CrossAttention(q=key, k=token,
    v=token);# cross attention output:[C,H,W]
13    f = norm(f + attn_out); # update feature:[C,H,W]
14    f_up = up_scale(f) # f_up:Cx4Hx4W
15    token = MLP(token) # token:[N+1,C]
16    mask = token @ f # mask:[N+1,4H,4W]
17    mask = mean(mask, dim=0) # mask:[4H,4W]
18    return mask
19 def mask_fusion(mask_list):
    # masks:a list of masks from each scale
20    final_mask = zeros(4Hx4W); # initialize empty mask
21    for l, m in enumerate(mask_list):
22        final_mask = final_mask + gamma[l] * m
23    return final_mask
# Main Function
24 def pixel_decoder(f_img, h_seg):
    # masks:a list of mask from each scale
25    mask_list = []
26    for l, (f, h) in enumerate(zip(f_img, h_seg)):
27        if l < L-1:
28            f = feature_update(f, mask)
29            ;# update features after the scale L
30            mask = attention_block(h, f, l)
31            mask_list.append(mask)
32    final_mask = mask_fusion(mask_list)
33    return final_mask
```

---

Based on the IoU scores, we can calculate gIoU and cIoU metrics by referring segmentation dataset.

## B. More Ablative Experiments.

**Multi-scale tokens sharing.** To clearly demonstrate the necessity of our multi-scale segmentation tokens in the codebook  $C_{seg}$ , we continue using  $L$  multi-scale features, but reduce the original codebook shape from  $N \times L$  to  $N$  (recall that  $N$  is the number of tokens in each scale group). This reduction resulted in the remaining  $N$  tokens being identi-

cal (i.e., shared) across all  $L$  scales. As Tab. A.2 shows, using a dedicated segmentation token for each scale yields better performance.

**Multi-scale layer selection.** We experiment with different layer selections as presented in Tab. A.3. The CLIP-ViT-L used in PixelLM consists of 24 layers. To reduce the computational cost of the decoder, we avoid selecting features from all layers of ViT. Instead, we follow the multi-scale feature selection ratio from prior work [11, 18] and consider other selection options. Our experiments reveal that selecting features from layers before the middle does not yield benefits for our task (the fourth row in Tab. A.3). Therefore, we mainly focus on selections from the middle and rear layers. The results show that using features from layers 14 and 23 leads to the best outcomes.

## C. More Details about MUSE

### C.1. Data Filtering

**GPT-4V filtering.** Although GPT-4V can efficiently understand image content, there are still failure cases in the generated data, which can be summarized in the following two points:

- Questions are vague and open to multiple interpretations. For example, the question “What should I take with me on my outing?” is extremely vague because “outing” can mean a wide array of activities.
- Answers that omit semantically equivalent instances. For instance, a question might ask about “choosing a fruit for a snack”, but the answer may only suggest “an apple”, ignoring other fruit visible in the image.

Therefore, it is necessary to employ a stringent sample filtering process to guarantee the quality of the data. Toward this goal, we develop a GPT-4V assisted data filtering pipeline. This pipeline operates by prompting GPT-4V to evaluate all initial question-answer pairs, based on identified common failure modes. Pairs classified by GPT-4V as falling within these failure categories are removed. This procedure effectively excludes approximately 20% of the preliminary data. The specific prompts used in this process are detailed in Fig. A.1.

**Human verification.** To ensure high quality in the generated question-answer pairs during the evaluation stage, we further engage experienced human annotators to double-check our evaluation set. Our approach is driven by two primary objectives:

- The questions should follow an intuitive and logical sequence that a person would typically think of when viewing the image.
- The answers should correspond closely to the way a human would naturally respond to the question

The above filtering process effectively ensures that the questions in the MUSE dataset are sufficiently challenging for

**Prompt:** You are an intelligent chatbot designed to evaluate the correctness of generative outputs for question-answer pairs. You receive a list of objects, each described in the image you are observing. The image has a height of 480 and a width of 640. The image caption, objects in the image, and their respective bounding box coordinates are as follows:

**Global Caption;**  
**A grey cat with a collar is lounging on a closed laptop [115.88, 271.59, 187.77, 336.81];**  
 ...

Coordinates represent (top-left x, top-left y, bottom-right x, bottom-right y). The question should involve at least two of these objects and must be framed in a way that requires image reasoning for response. Additional requirements for the generated question include:

- 1.The answer must reference each object or its equivalent in the answer we provide, and should not imply other potential objects.
- 2.The question must be specific and meaningful, not overly broad.
- 3.The question should describe a complete activity, rather than just combining several sub-problems.
- 4.In the answer, rephrase the class name to indicate its location or shape.

**Question:** How can I comfortably listen to music while petting my cat when I get home from a long day at work?

**Generated Answer:** You can lie down comfortably on the large bed <SEG> covered with soft quilts <SEG> . Then take the silver laptop <SEG> out from under the chubby furry cat <SEG> next to you and connect it to the black wired headphones <SEG> next to you to listen to music.

**Modified Answer:** You can lie down comfortably on the large bed (A bed covered in a light blue quilt occupies the majority of the scene ) covered with soft quilts (A crumpled light blue quilt almost completely covers the bed). Then take the silver laptop (A closed laptop is positioned towards the foot of the bed under a resting cat) out from under the chubby furry cat (A grey cat with a collar is lounging on a closed laptop) next to you and connect it to the black wired headphones (A pair of over-the-ear headphones rests next to the cat) next to you to listen to music.




Figure A.1. **Evaluation example.** The left panel illustrates the prompt employed in our GPT evaluation pipeline. The right panel showcases an example of a predicted answer alongside its corresponding modified version, as input to GPT.

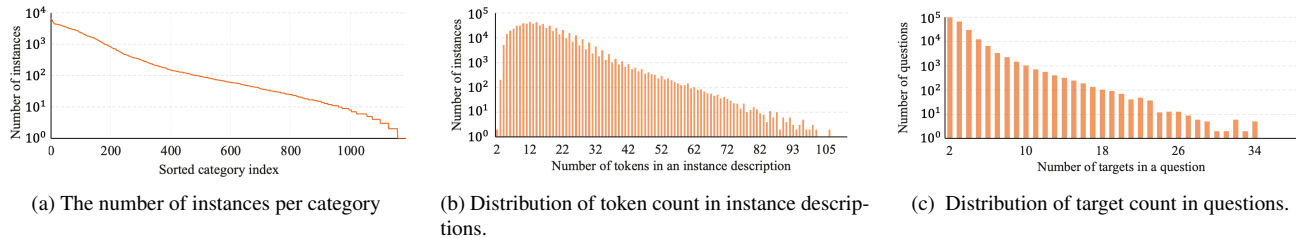


Figure A.2. **Dataset statistics.** Best viewed digitally.

reasoning, while the answers remain detailed and accurate.

### C.2. Dataset Statistics

In this section, we systematically analyze our dataset. First, our question-answer pairs are based on over 1000 categories, encompassing a wide spectrum of objects found in daily scenes. Additionally, the descriptions of objects in the answers go beyond mere category names limited to a few tokens. Instead, they offer context-specific descriptions extending to over 100 tokens. This demonstrates that our dataset is rich in perception information, crucial for real-world applications. Finally, we present the statistics regarding the number of objects involved in a data sample.

**Category statistics.** There are over 1000 categories in MUSE from the original LVIS dataset, and 0.9 million instances with unique descriptions that vary based on the context of the question-answer pairs. Fig. A.2a shows the number of instances per category on all question-answer pairs. The distribution inherits the low-shot nature of LVIS.

**Token count.** Fig. A.2b presents the distribution of instances by token count in their descriptions, highlighting

a wide range that exceeds 100 tokens in the most extensive cases. These descriptions are not limited to simple category names; rather, they are substantially enriched with detailed information about each instance, encompassing aspects like appearance, attributes, and relationships with other objects, thanks to our GPT-4V-based data generation pipeline. The depth and variety of information in the dataset bolster the trained model’s generalization capabilities, enabling it to effectively address open-set questions.

**Target count.** Fig. A.2c presents statistics on the number of targets in each question-answer pair. The average number of targets is 3.7, with the maximum number of targets in a single pair reaching up to 34. This number can cover most scenarios of target reasoning for a single image.

### C.3. Multi-referring Segmentation

As mentioned in Sec. 5.1, we transform conventional referring segmentation datasets into a multi-referring format for model training. In this subsection, we detail this process. The transformation involves selecting one to three distinct target objects from the annotations of each image.

**Prompt:** You are an intelligent chatbot designed to evaluate the correctness of generative outputs for question-answer pairs. You receive a list of objects, each describing an object in the image you are observing. The image has a height of 480 and a width of 640. The image caption, objects in the image, and their respective bounding box coordinates are as follows:

**Global Caption:** The image depicts a cozy scene with a cat lying on top of a laptop computer, which is situated on a bed. The cat is in the center of the image, covering a significant portion of the laptop. The bed serves as the main background element, with the laptop and cat as the primary subjects of the scene.

**Cat at [115.88, 271.59, 187.77, 336.81];**  
**Laptop at [569.87, 67.5, 70.13, 195.33];**  
 ...

Coordinates represent (top-left x, top-left y, bottom-right x, bottom-right y). The question must involve at least two of these objects and be framed to require image reasoning for a response. Additional requirements for the generated question include:

- 1.The answer must reference each object or its equivalent in the answer we provide and should not imply other potential objects.
- 2.The question must not be too broad and must be meaningful.
- 3.The question should describe a complete activity, not just a combination of several sub-problems.
- 4.Rephrase the class name in the answer to indicate its location or shape.

**Question:** Can you identify the object in the picture that is typically used in office, and the item often used to provide a comfortable resting environment ?

**Answer:** A laptop under the cat and the bed.

Figure C.3. **Example of GPT-4 data generation.** The corresponding image is the same as in Fig. A.1.

These objects are used to construct questions in the format: Please segment the <objects> in the image, where <objects> represents a list of comma-separated object descriptors. The response format is a list of comma-separated <object> is <SEG>, with <object> being the description of each object. We require that the order of predictions in the answer matches the order of object names in the question and calculate gIoU and cIoU for each prediction based on this.

**C.4. More Details about GPT-4 Generated Data.**

In Sec. 5.5, we create 30,000 additional multi-target question-answer pairs to compare GPT-4 and GPT-4V. The prompts for GPT-4 adhere to similar generation principles but incorporate detailed image captions to offset the absence of visual content. Fig. C.3 demonstrates an example of our GPT-4 data generation process and its typical failures. Given that image content is not directly perceivable, conveying as detailed an image description as possible into GPT-4 is crucial to compensate for this information gap. However, this method often leads to a lack of diversity in the generated question-answer pairs (refer to the question-answer pair at the bottom of Fig. C.3), characterized by: *i*) Numerous questions are composed of simple referring style sub-questions; *ii*) The question and answer content is limited to what the image caption describes; *iii*) Challenges in generating detailed object descriptions. To address these issues, it might be necessary to introduce more complex annotation details like object relationships, which significantly increases the complexity and burden of data generation.