A. Experimental Details

A.1. Model Overview

We extend Llama into a multi-modal foundation model dubbed **AgentDet**. Concretely, we instantiate AgentDet with three backbone LLMs: Qwen3-8B, Qwen2.5-7B, and Llama3.1-8B. The backbone is further augmented with:

1. two lightweight linear projections (`projection` and `projection_clip`) that map visual features into the language space;
2. a learnable knowledge base (KB) initialized offline from CLIP embeddings of fine-grained image patches (cf. Sec. A.1.2);
3. special delimiter tokens `<visual_sup/que>` . . . , which allow the model to *splice* visual tokens directly into the language sequence by the differentiable operator $\mathcal{I}(\cdot)$ in Eq. (29).

A.1.1. Splicing operator.

Given textual embeddings $\mathbf{T} \in \mathbb{R}^{L\times d}$, a start/end token pair (s, e), and M visual vectors $\mathbf{V} \in \mathbb{R}^{M\times d}$, we form

$$\mathcal{I}(\mathbf{T}, \mathbf{V}; s, e) = \left[\ \mathbf{T}_{1:s}\ ;\ \mathbf{V}\ ;\ \mathbf{T}_{e:L}\ \right], \tag{29}$$

where ; is concatenation along the sequence dimension. Labels corresponding to $\mathbf{V}$ are masked with -100 to avoid gradient leakage into the language loss.

A.1.2. Knowledge-Base Construction.

For each class, we parse an attribute sentence (e.g. *"horse: a long mane, . . . "*) into one global and several attribute phrases. CLIP (`ViT-B/32`) encodes these phrases as text keys $\mathbf{K} \in \mathbb{R}^{N\times 512}$. Image regions are obtained by multi-scale SLIC super-pixels; their CLIP embeddings serve as value vectors. A similarity threshold $\tau = 0.22$ filters noisy matches. The resulting tuples $\langle$attr, patch, embed$\rangle$ are serialized with `joblib` for fast retrieval.

A.1.3. Parameter-efficient fine-tuning.

We activate LoRA on {`q/k/v/up/down/o/gate_proj`} of all transformer blocks; visual projections and the Q-Former remain *trainable*, while the vision encoder is frozen.

A.1.4. Hardware and Software.

Experiments run on 4 NVIDIA A100 80GB GPUs. We employ Python 3.10, PyTorch 2.1, HuggingFace `Transformers` 4.41, PEFT 0.11, and SAM `vit_h` for over-segmentation operator. Training a single model takes $\sim$18 h wall-clock time. All random seeds are fixed to `42` (Python, NumPy, PyTorch).

A.2. Datasets Preparation

- **PASCAL-VOC**: 10 727 training images with object annotations; we convert them to JSONL where each line stores {*image_path*, caption, class, bbox, $w\times h$}.

Parameter	Value
Base model	Meta-Llama-3.1-8B
Optimizer	AdamW
Learning rate	2×10^{-5}
Batch size	2
Grad accumulation	4
LoRA rank r	8
LoRA α	32
LoRA dropout	0.05
Precision	bfloat16
Epochs	3 (COCO), 15 (VOC)

Table 6. Key hyper-parameters.

- **COCO**: COCO follows the same data organization as VOC in our implementation.

A.3. Training Protocol

Training is performed with the HuggingFace `Trainer`. Table 6 summarizes the hyper-parameters.

B. Additional Experiment

B.1. Training and Inference Efficiency

Table 7 reports the stage-wise runtimes for the VOC (small) and COCO (medium) benchmarks. Every stage scales *linearly* with the number of training samples, consistent with the $O(n)$ complexity of KB construction and modality alignment.

Dataset	Pre-KB	Stage 0	Stage 1	Stage 2	Stage 3	**Total**
VOC	0.3	1	4	10	2	**16**
COCO	12	1	18	44	2	**64**

Table 7. Stage-wise training time (hours).

Inference Throughput and Latency. During inference the model processes **1 object s**$^{-1}$ (Table 8). Our *last-place elimination* strategy filters low-quality image–text pairs from the KB, causing only a negligible $+4$ ms overhead (0.98 s $\rightarrow$ 1.02 s) while preserving retrieval accuracy.

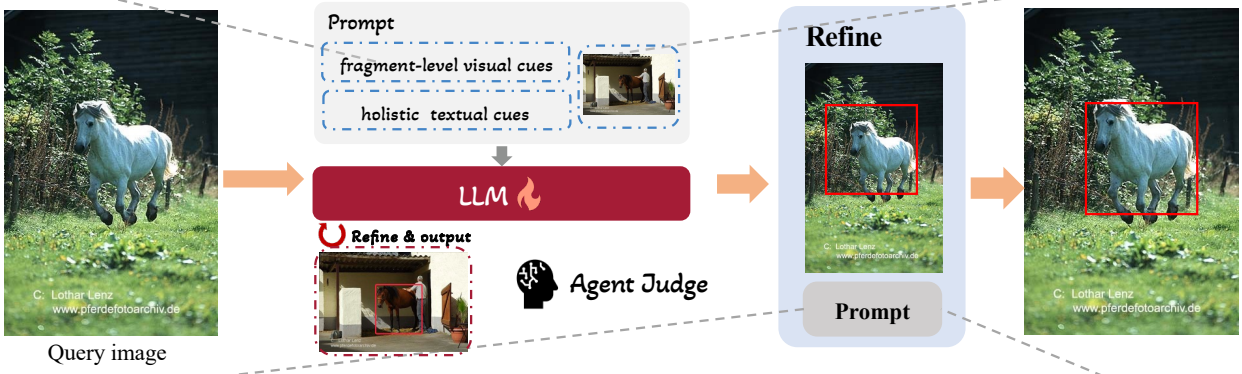
Metric	**w/o KB**	**w KB**
w/o Retrieval (obj/s)	0.98	0.98
w Retrieval latency (s/obj)	-	1.02

Table 8. Inference efficiency on a single A100.

B.2. Cooperation vs. Chain-Style Execution

To isolate the effect of the proposed cooperation mechanism, we construct a component-matched chain-style vari-

Instruction: You are a highly skilled data annotator, specializing in analyzing JSON-formatted data to extract relevant object annotations accurately. Your primary responsibility is to identify and annotate objects in a given image based on the provided data. Each image may contain multiple objects belonging to different categories, but your goal is to focus on objects of the class '{class_name}'. Annotations are expected to include the class name and bounding box coordinates for each object. The bounding box coordinates should be represented in the format [x_min, y_min, x_max, y_max] and must strictly adhere to the image size boundaries. Your task is structured as follows:
1. Analyze the input data to understand the image's size, description, and existing annotations."
2. Identify all objects of the class 'horse' and extract their bounding box coordinates."
3. Ensure that all detected objects of the specified class are annotated correctly, even when there are multiple instances of the same class in the image."
The input includes the image size, description, and annotations, provided in JSON format as follows:
Input:"
{{
"image": {{an image user give}},
"image_size": {{[img_x_max, img_y_max]}},
"description": "{{A sentence describing this image.}}",
}}
Your output should only include the bounding box coordinates for the class '{class_name}', formatted as follows:
Output: [[x_min, y_min, x_max, y_max], ...]
This image contains some reference information, which includes attributes of certain classes that you may refer to when identifying these categories, along with their appearance in the image. "
These references are one-to-one correspondences for your reference. "
The reference information you may use is as follows:
{{
"{class_name}'s\": <attr_embeds_start><attr_embeds_end>,
"visual tokens of {class_name}'s attribute\": <visual_sup_start><visual_sup_end>
}}



Prompt: You are an excellent bounding box refinement expert. Given an image and an object category, you can make slight adjustments to the initial annotation bbox_{llm} provided by another annotator, using your knowledge to make the bounding box more accurately enclose the object.
Image: {image vector}
Reference Annotation: {bbox_{llm}}
Output:

Figure 5. Qualitative illustration of a zero-shot detection case. Our AgentDet framework segments, retrieves, and grounds fine-grained attributes for accurate localization without category-specific supervision.

Backbone / Shot	Novel Split 1					Novel Split 2					Novel Split 3				
	0	1	3	5	10	0	1	3	5	10	0	1	3	5	10
Llama-3.1	29.8	52.3	59.2	63.5	63.9	25.2	35.4	46.9	50.3	52.1	30.2	54.7	53.1	53.9	56.5
Qwen-2.5	31.3	62.2	65.1	65.5	68.2	30.0	32.8	47.3	53.8	57.4	29.9	54.4	60.1	59.4	61.8

Table 9. mAP50:95 on VOC.

ant in which the same four agents are executed strictly in sequence, without shared-blackboard interaction. This comparison keeps the major modules unchanged and only alters the coordination strategy.

We observe that the chain-style pipeline achieves comparable final detection performance, but incurs substantially higher computational cost. In particular, compared with the shared-blackboard setting, chain-style execution increases training time by 156% and inference time by 73%. These results indicate that the main advantage of shared-blackboard cooperation lies in improved efficiency and scalability, while maintaining competitive detection accuracy.

B.3. Evaluation Protocol.

We report mean Average Precision (mAP) following FSOD conventions. At inference, beam size is set to 4 and maximum generation length to 128 tokens. Meanwhile, we provide more comprehensive experimental results, as illustrated in Table 10 for the COCO dataset and Table 9 for the VOC dataset, reporting mAP@50:95. These results further validate the robustness of our approach across varying object scales and category distributions.

Method	Shot					
	0	1	3	5	10	30
Llama-3.1	4.1	8.2	12.4	18.5	23.2	27.8
Qwen-2.5	4.5	8.7	11.3	17.2	23.7	28.1

Table 10. mAP50:95 on COCO.

B.4. Effect of Pseudo-Incremental Knowledge Base Updating

To evaluate whether progressive memory updating is necessary, we compare our pseudo-incremental knowledge base (KB) construction strategy with a static alternative that builds the KB only once before training and keeps it fixed thereafter. The comparison is conducted on PASCAL VOC Novel Split 1 under 0/1/3/5-shot settings.

Table 11 shows that pseudo-incremental updating consistently outperforms the static KB across all shot settings. This suggests that progressively refreshing the visual memory with newly consolidated evidence improves the quality of retrieved fine-grained cues, which in turn benefits downstream localization and classification of novel categories.

Method	Novel Split 1			
	0	1	3	5
Static KB (built once)	33.2	53.8	61.9	64.6
Pseudo-incremental KB (Ours)	**35.7**	**56.5**	**64.2**	**68.5**

Table 11. Effect of pseudo-incremental KB updating on VOC Novel Split 1.

C. Case Study

C.1. Example

Fig. 5 illustrates a qualitative example from the VOC2012 benchmark to demonstrate the effectiveness of our proposed AgentDet framework in a challenging zero-shot detection setting. Given a query image with cluttered background and occluded objects, our system performs multi-scale segmentation to extract candidate visual regions without relying on category priors. Visual features from both the full image and fine-grained patches are encoded via EVA-CLIP and Q-Former, and subsequently projected into a unified embedding space.

Leveraging the knowledge base, which stores attribute-level image–text pairs, relevant references are dynamically retrieved based on visual similarity. These retrieved attributes serve as structured priors to construct an interpretable prompt, which guides the LLM to regress bounding box coordinates directly. No explicit post-processing (e.g., NMS) is required.

As shown in Fig. 5, AgentDet successfully identifies and localizes a *horse* instance, despite significant intra-class variation and visual ambiguity. This case highlights the synergy between knowledge-grounded retrieval and instruction-following generation, and underscores the practicality of our approach in open-world object detection scenarios.

C.2. Bad Case

Fig. 6 shows several representative failure cases of AgentDet. Most errors arise in challenging scenarios involving fragmented candidate regions, repetitive visual patterns, or multiple similar instances in cluttered scenes. In such cases, the model may produce partial grounding or drift toward neighboring objects.

These examples suggest that the performance of AgentDet still depends on the quality of intermediate segmentation and the discriminativeness of retrieved visual evidence. Improving candidate consolidation and instance-aware retrieval may further enhance robustness in these difficult cases.

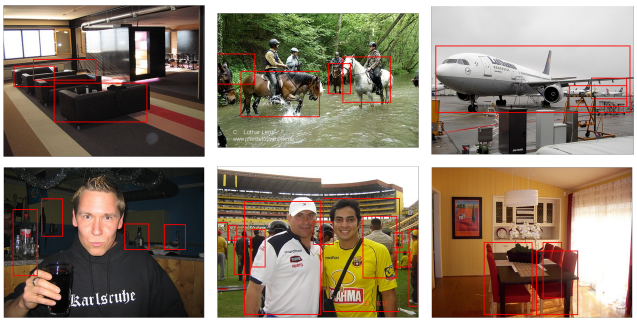


Figure 6. Representative failure cases of AgentDet. Typical errors occur when segmentation fragments the object excessively, retrieval returns weak or misleading attributes, or visually similar instances appear in cluttered scenes.

D. Limitations

While our method demonstrates strong performance on zero-shot and few-shot detection tasks, it currently relies on a knowledge base whose expansion is still limited and not yet fully open-world. This design limits its adaptability to entirely novel domains or evolving object categories not

present during KB construction. In future work, we plan to explore dynamic knowledge integration and continual KB expansion to better support open-world generalization.