Visual Para-Thinker++: A Single-Policy Multi-Agent Framework for Visual Reasoning

Anonymous ACL submission

Abstract

Visual reasoning requires integrating evidence distributed across regions, attributes, and relations, making single-chain reasoning prone to early perceptual commitment and hallucination. We propose Visual Para-Thinker++, a single-policy multi-agent framework in which one shared MLLM policy is instantiated as role-conditioned Main, Worker, and Summary Agents. The Main Agent decomposes the task with fixed allocation patterns; Worker Agents reason in parallel under context isolation; and the Summary Agent reconciles full Worker reasoning traces rather than majority-voting on final labels. The shared policy is trained by Multi-Agent Capability Injection and Role-Decoupled Multi-Agent Optimization, which assign role-specific rewards and advantages to corresponding token segments to reduce gradient conflict among collaborative roles. A native inference engine enables efficient multi-agent rollout through shared visual prefix and KV cache reuse. Across V*, CountBench, Pixmo, MMVP, the RefCOCO family, and HallusionBench, Visual Para-Thinker++ consistently outperforms single-trajectory and inference-time parallel baselines, with especially strong gains on hallucination-sensitive visual reasoning.

1 Introduction

Test-time scaling has become the dominant lever for improving the reasoning ability of large language models (LLMs) (Wei et al., 2023; OpenAI et al., 2026): lengthen a single chain of thought, encourage self-reflection, and let one trajectory revise its own intermediate conclusions. Reinforcement Learning with Verifiable Rewards (RLVR) further amplifies this single-agent recipe by optimizing one policy against an outcome signal. The recipe has been effective in text-dominant domains such as mathematics and code, but its returns are markedly weaker on visual reasoning, where relevant evidence is distributed across regions, attributes, and relations rather than concentrated in a single linguistic argument. A single trajectory typically commits to one perceptual interpretation early; once an interpretation is selected, lengthening the chain tends to deepen it rather than diversify it, and an early perceptual error tends to be justified rather than corrected.

This observation motivates a different scaling axis: rather than making one agent's chain longer, organize visual reasoning as multi-role collaboration. Different reasoning roles, including question decomposition, local evidence gathering, global count verification, and intermediate finding reconciliation, are naturally assigned to different agents. Based on above, we propose a multi-role collaboration framework, termed Visual Para-Thinker++. Rather than using multiple independent models—which is computationally expensive—we introduce a single-policy multi-agent system. By conditioning a shared MLLM on specific role tokens, our approach enables subagents to collaborate.

Specifically, Visual Para-Thinker++ multi-agent framework consists of three types of roles:

- **Main Agent** that, given the image and the question, the role decomposes the visual task into sub-tasks based on two distinct allocation patterns, assigning each to worker agents.
- **Worker Agents**, upon receiving sub-tasks from the main agent, worker agents execute their respective assignments in a decoupled and isolated manner.
- **Summary Agent** that aggregates the outputs from various sub-agents, integrating their respective reasoning results to formulate a coherent final response.

To realize this multi-agent collaboration within a single policy, we propose Role-Decoupled Multi-Agent Optimization. First, we employ Multi-Agent

Capability Injection, a role-aware SFT objective that enables a shared policy to instantiate distinct Main, Worker, and Summary roles. We utilize a context-isolation mask to ensure that Worker Agents operate independently, while the Summary Agent maintains full visibility of all preceding traces. Subsequently, the policy is refined via Role-Decoupled Optimization, where rewards are decomposed into role-specific signals: Worker rewards evaluate local evidence and sub-task accuracy, while the Summary reward assesses the final output. By mapping these signals to role-specific token segments, we mitigate gradient conflicts arising from the divergent objectives of the agents—specifically, the tension between local evidence gathering and global synthesis—while maintaining a unified set of model weights.

Empirically, Visual Para-Thinker++ consistently outperforms both single-trajectory (e.g., greedy decoding, long CoT) and parallel inference baselines (e.g., self-consistency, multi-agent debate) across six visual benchmarks. Finally, to enhance inference efficiency, we optimize the system's execution performance by leveraging KV cache reuse across the multi-agent reasoning process (Ye et al., 2025).

Contributions. Our contributions are summarised in three points:

- We propose Visual Para-Thinker++, a single-policy multi-agent visual reasoning framework where one shared MLLM policy is instantiated as Main Agent, Worker Agents, and Summary Agent through role conditioning.
- We introduce a unified two-stage optimization for Visual Para-Thinker++: Multi-Agent Capability Injection and Role-Decoupled Multi-Agent Optimization to reduce the gradient conflicts among different roles.
- Across V*, CountBench, Pixmo, MMVP, the RefCOCO family, and HallusionBench, Visual Para-Thinker++ improves visual reasoning more effectively than extending a single chain of thought. Practical evaluation is made efficient by a native vLLM-based multi-agent rollout engine that reuses KV cache across roles.

2 Related Work

Reasoning MLLMs. Since GPT-4V, MLLMs have been asked to produce visible chains-of-thought for visual question answering (Zhang et al., 2024; Liu et al., 2023). Recent work either post-trains MLLMs with reasoning data (LLaVA-o1 (Rafailov et al., 2024), Visual-CoT (Shao et al., 2024)) or activates reasoning through test-time prompting (Mitra et al., 2024). These are fundamentally single-agent, single-trajectory systems; they do not explicitly assign reasoning roles to different agents.

Parallel and self-consistent reasoning. Self-consistency (Wang et al., 2023) samples K independent chains and majority-votes on the final answer. Tree/Graph-of-Thought (Yao et al., 2023) generalize to search trees, and Para-Thinker (Wen et al., 2025) explores shared prefixes with divergent suffixes in a text-only setting. These methods are inference-time heuristics drawn from the same single role distribution: every chain inherits the same single-agent bias, and the system aggregates only at the final-answer level. Our framework instead learns role-conditioned agents into a single shared policy and aggregates at the trace level via a Summary Agent.

Multi-agent LLMs. Multi-agent debate (Du et al., 2023) and role-play systems (Chen et al., 2023) instantiate multiple distinct models or prompts to encourage role separation. The dominant cost is K-fold inference on heterogeneous models, and gradients rarely flow through the team. Our single-policy multi-agent design collapses these K models into one shared set of weights, makes role identity a function of role tokens and visible context rather than of separate models, and admits end-to-end optimization within a unified RL stage.

3 Method

3.1 Overview

Visual Para-Thinker++ instantiates one shared MLLM policy as three role-conditioned agents under a fixed four-path collaboration protocol (Fig. 1). The Main Agent receives the image and the question and decomposes the visual task into complementary sub-problems through a small set of fixed task-allocation patterns. A fixed set of Worker Agents, role-conditioned instances of the same policy, then explore these sub-problems along parallel reasoning paths under context isolation. A Summary Agent reads the complete Worker reasoning traces and produces the final answer through trace-level evidence reconciliation.

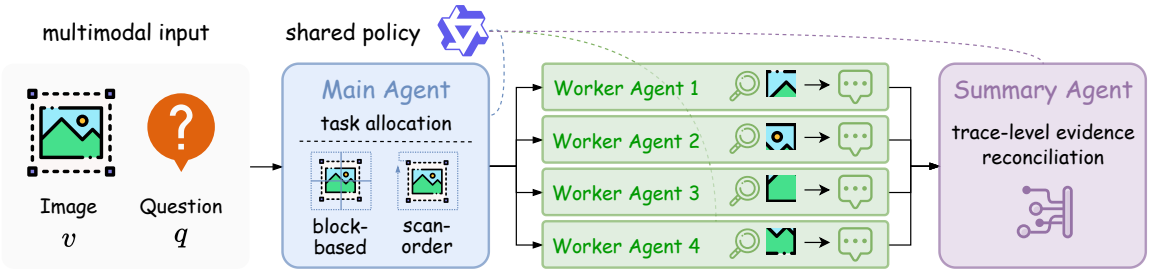

Figure 1: Visual Para-Thinker++ framework. A single shared policy π_θ is instantiated as three role-conditioned agents under a fixed protocol. Given a multimodal input (v, q), the **Main Agent** emits one of two fixed task-allocation patterns (block-based or scan-order); four **Worker Agents**, role-conditioned instances of the same policy, then explore the dispatched visual sub-problems along parallel reasoning paths under mutually isolated contexts; a **Summary Agent** performs trace-level evidence reconciliation to produce the final answer. All three agents share the same policy weights (see the shared-policy icon on top); only the role embedding and the visible context differ.

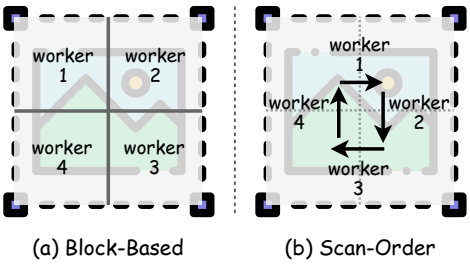

Figure 2: The two fixed task-allocation patterns used by the Main Agent under the four-Worker protocol. (a) **Block-based**: the image is partitioned into four disjoint quadrants, each dispatched to one Worker Agent for local evidence collection. (b) **Scan-order**: each Worker Agent retains a global view but traverses the image along a distinct scan order.

We deliberately frame the Main Agent as a fixed-pattern dispatcher rather than as an open-ended planner, and the Summary Agent as a learned reconciler rather than as a majority voter. There are no independently trained policies: all three agents share the same set of weights θ, and role identity is induced solely by role tokens and visible context. Visual Para-Thinker++ is therefore a single-policy multi-agent system: a unified optimization framework rather than a collection of agents glued together by prompts.

3.2 Single-Policy Multi-Agent Formulation

We model Visual Para-Thinker++ as a single autoregressive policy π_θ unrolled under a role-aware state machine. Given image v and question q, a multi-agent trajectory τ is decomposed into $K{+}2$ role segments:

$$\tau = \underbrace{m}_{\text{Main}} \Big\| \underbrace{s_1}_{\text{Worker}_1} \Big\| \cdots \Big\| \underbrace{s_K}_{\text{Worker}_K} \Big\| \underbrace{u}_{\text{Summary}}, \qquad (1)$$

where the Main segment m is conditioned on (v, q, r_{main}), each Worker segment s_i on $(v, q, m, r_{\text{worker}_i})$, and the Summary segment u on $(v, q, m, s_{1:K}, r_{\text{sum}})$. Worker Agents never see each other's tokens during their own reasoning; the Summary Agent retains full visibility. Role tokens $r_{(\cdot)}$ are learned embeddings in one shared vocabulary; all three agents share the transformer weights θ. The protocol order Main $\rightarrow$ Worker $\rightarrow$ Summary, the four-path layout, and the visibility relations are fixed at training and inference.

3.3 Main-Agent Task Allocation with Fixed Patterns

The Main Agent converts (v, q) into K visual sub-problems for the Worker Agents through one of two fixed allocation patterns (Fig. 2): **Block-based allocation** partitions the image into K disjoint regions and dispatches each Worker Agent to collect local evidence, fitting grounding, fine-grained perception, and attribute verification; **Scan-order allocation** assigns each Worker Agent a distinct global scanning trajectory while retaining a full receptive field, fitting counting and global verification where errors arise from missing or double-counting under a single direction. By default the Main Agent emits Block-based allocation and switches to Scan-order when the task is dominated by counting or global verification; the chosen pattern is part of segment m and is visible to all Workers and to the Summary Agent.

3.4 Multi-Agent Capability Injection

The first training stage teaches the shared policy π_θ to instantiate Main, Worker, and Summary roles via SFT. Given a sample $(v, q, m^\star, s^\star_{1:K}, u^\star)$ synthesised by a stronger MLLM teacher (Appendix B),

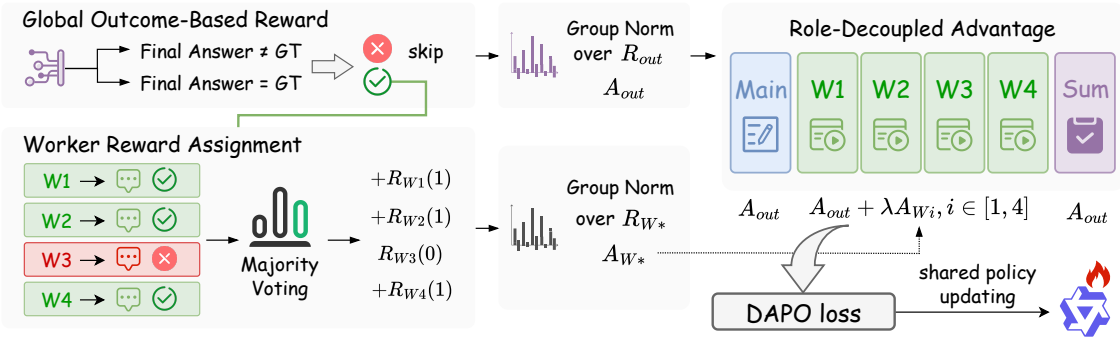

Figure 3: Training pipeline of Role-Decoupled Multi-Agent Optimization. **Top branch (Global Outcome-Based Reward).** A trajectory contributes the outcome reward $R_{\text{out}} \in \{0, 1\}$ depending on whether the Summary Agent's final answer matches ground truth; group normalisation over the on-policy group then yields the outcome advantage A_{out}. **Bottom branch (Worker Reward Assignment).** Each Worker Agent's intermediate sub-answer is graded by a lightweight cross-Worker majority-vote heuristic (e.g., W3 disagrees with the majority and is penalised, R_{W3}=0); group normalisation over all Worker rewards yields the per-Worker advantage stream, written compactly as A_{W*} in the figure and instantiated as $A_w^{(i)}$ for Worker i. **Composition.** The two advantage streams are composed token-wise (Eq. 4): Main and Summary segments receive A_t=A_{out}, while Worker i's segment receives A_t=$A_{\text{out}} + \lambda A_w^{(i)}$. The composed advantages are plugged into a clipped DAPO loss to update the shared policy.

we minimise

$$\begin{aligned}\mathcal{L}_{\text{MAC}}(\theta) = &-\sum_{t \in m} \log \pi_\theta(m_t \mid c_m, m_{<t}) \\ &-\sum_{i=1}^{K} \sum_{t \in s_i} \log \pi_\theta(s_{i,t} \mid c_i, s_{i,<t}) \quad (2) \\ &-\sum_{t \in u} \log \pi_\theta(u_t \mid c_u, u_{<t}),\end{aligned}$$

with c_m=(v, q, r_{main}), c_i=$(v, q, m, r_{\text{worker}_i})$, and c_u=$(v, q, m, s_{1:K}, r_{\text{sum}})$. The role tokens $r_{(\cdot)}$ are the only inputs distinguishing the three agents under shared weights, and the Worker index i in r_{worker_i} forces $\pi_\theta(\cdot \mid \cdot, r_{\text{worker}_i})$ to specialise across i rather than collapse. We additionally apply a context-isolation mask on the packed trajectory so that each Worker Agent attends only to (v, q, m) plus its own suffix while the Summary Agent retains full visibility of all Worker traces; without this mask the parallel paths copy each other rather than explore complementary evidence, and the Summary Agent's full visibility is what later enables trace-level reconciliation in §3.5.

3.5 Role-Decoupled Multi-Agent Optimization

The second stage refines the shared policy with group-relative RL over the multi-agent trajectory of Eq. 1. Worker Agents and the Summary Agent pursue partially different objectives, so a single trajectory-level reward broadcast to every token couples their gradients: a wrong final answer poisons every Worker, and a noisy Worker reward leaks into the Summary segment. We introduce role-specific rewards together with role-specific advantages applied only to the corresponding token segments; the overall pipeline is shown in Fig. 3.

Reward signals. Each Worker Agent i receives a worker-level reward $R_{\text{worker}}^{(i)} \in [0, 1]$ that grades its local reasoning trace, instantiated by a lightweight cross-Worker majority-vote heuristic and used only as a training signal. The team-level outcome reward $R_{\text{out}}(\tau) \in \{0, 1\}$ grades the Summary Agent's final answer and supervises the Main and Summary segments.

Role-decoupled advantages. We group-normalise each reward source over the on-policy DAPO group separately:

$$\begin{aligned}A_{\text{out}} &= \frac{R_{\text{out}} - \mu_{\text{out}}}{\sigma_{\text{out}}}, \\ A_{\text{worker}}^{(i)} &= \frac{R_{\text{worker}}^{(i)} - \mu_{\text{worker}}}{\sigma_{\text{worker}}},\end{aligned} \quad (3)$$

and compose them token-wise:

$$A_t = \begin{cases} A_{\text{out}}, & t \in m \cup u, \\ A_{\text{out}} + \lambda\, A_{\text{worker}}^{(i)}, & t \in s_i, \end{cases} \quad (4)$$

with Worker weight λ=0.5 (Appendix C). The outcome advantage flows to every role as the team-level signal, while the Worker advantage updates only its own segment, equivalent to a segment-wise mask on the gradient: a mistake in Worker j

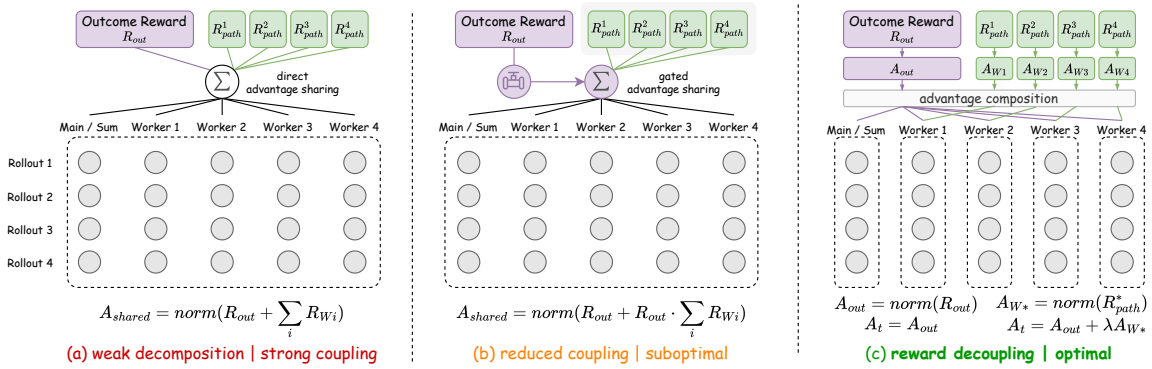

Figure 4: Three representative ways to compose a Worker reward with the outcome reward across a multi-agent rollout. The figure denotes the per-Worker reward as R_{Wi} (also written as R^i_{path}) and its normalised advantage as A_{W*}; in the main text we equivalently write these as $R^{(i)}_w$ and $A^{(i)}_w$. **(a) Naive Sum** (*weak decomposition, strong coupling*): all Worker rewards are summed with R_{out} into a single $A_{\text{shared}} = \text{norm}(R_{\text{out}} + \sum_i R_{Wi})$ that is broadcast to every token segment, coupling Worker and outcome gradients. **(b) Conditional Sum** (*reduced coupling, suboptimal*): R_{out} gates the Worker rewards via $A_{\text{shared}} = \text{norm}(R_{\text{out}} + R_{\text{out}} \cdot \sum_i R_{Wi})$, but the advantage is still shared across roles. **(c) Role-decoupled Advantages (ours)** (*reward decoupling, optimal*): R_{out} and each R_{Wi} are normalised *within their own on-policy groups* into A_{out} and A_{W*}, then composed token-wise via Eq. 4—Main and Summary tokens receive $A_t = A_{\text{out}}$, while Worker i's tokens receive $A_t = A_{\text{out}} + \lambda A_{W*}$.

changes the advantage only of tokens in s_j. The advantages are plugged into a clipped DAPO (Yu et al., 2025) surrogate loss.

How to compose R_w and R_o. A natural design question is *at which level* the Worker reward R_w should be combined with the outcome reward R_o. We identify three representative schemes (Fig. 4). **(a) Naive Sum** forms $R_o + \sum_i R^{(i)}_w$ and broadcasts a single normalised advantage to every token. **(b) Conditional Sum** gates the Worker rewards by the outcome via $R_o + R_o \cdot \sum_i R^{(i)}_w$ but still uses a shared advantage. **(c) Role-decoupled Advantages (ours)**, in the spirit of per-reward group normalisation for multi-reward RL (Liu et al., 2026), normalises R_o and each $R^{(i)}_w$ within their own groups (Eq. 3) and composes them at the advantage level (Eq. 4). Schemes (a) and (b) couple the two signals through a shared advantage; only (c) routes each reward source to the role tokens it is meant to supervise. Table 4 shows that (c) outperforms both summation variants under the same R_w, and Table 5 further shows that R_w and the role-decoupled advantage are complementary, with the largest gains obtained by combining them.

3.6 Native Multi-Agent Inference Engine

Efficient rollout is necessary for making single-policy multi-agent reasoning practical. A naive Main/Worker/Summary rollout requires $1+K+1$ decoding calls with redundant visual prefills.

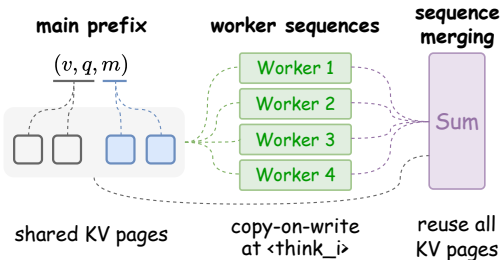

Figure 5: Native multi-agent inference engine.

Our vLLM-based engine instead pays the image-question-Main prefill once, stores the shared KV pages in a parent `SequenceGroup`, forks K Worker sequences at role-trigger tokens via PagedAttention Copy-on-Write, decodes Workers in parallel, and merges their KV pages before the Summary segment (Fig. 5). Thus, Worker Agents share the visual prefix without copying memory at fork time, and the Summary Agent reuses both parent and Worker KV pages rather than refilling the complete multi-agent context.

3.7 Implementation Details

We instantiate Visual Para-Thinker++ on a Qwen2.5-VL-3B (Bai et al., 2025) backbone (with a complementary 7B run in Appendix A). Training proceeds in two stages: Stage 1 performs Multi-Agent Capability Injection under the context-isolation mask of Eq. 2, and Stage 2 performs Role-Decoupled Multi-Agent Optimization with on-policy DAPO (Yu et al., 2025) using the role-

Table 1: Comprehensive evaluation of models on vision-centric tasks, including counting, fine-grained perception, and hallucination tasks. The first **Avg.** column averages the three counting columns (Pixmo val, Pixmo test, CountBench); the second **Avg.** column averages V*, MMVP, and HallusionBench. **Bold** marks our Visual Para-Thinker++ results in the last two rows; "–" denotes results not reported by the source. "Majority voting@4" generates four trajectories per query and aggregates by majority vote. Visual Para-Thinker++ extends Visual Para-Thinker with Role-Decoupled Multi-Agent Optimization, and improves over every Peer-Model baseline (the open-source 3B/7B group) on every benchmark and at both scales.

Model	Size	Counting			Avg.	Visual Search	Fine-grained	Hallucination	Avg.
		Pixmo		CountBench		V*	MMVP	HallusionBench	
		val	test	test					
Frontier Models									
GPT5-mini	–	–	54.7	84.4	69.5	78.6	65.3	63.2	64.2
GPT-4o (OpenAI et al., 2024)	–	–	54.4	85.7	70.5	73.9	63.8	61.4	62.6
Gemini2.5-Pro	–	–	59.8	89.7	74.7	76.7	69.8	63.7	66.7
Claude-4-Sonnet	–	–	53.5	90.2	71.9	45.0	63.9	59.2	61.5
Qwen2.5-VL (Bai et al., 2025)	72B	74.0	70.4	86.7	77.0	76.9	72.1	70.0	71.1
Peer Models									
Qwen2.5-VL	3B	50.1	51.6	69.2	57.0	63.3	59.2	56.1	57.7
Qwen2.5-VL	7B	63.0	67.7	83.1	71.3	75.9	68.3	66.0	67.2
Sequential (SFT)	3B	51.5	52.1	70.4	58.0	65.0	60.0	58.3	59.2
Sequential (SFT)	7B	65.5	68.0	83.7	72.4	76.1	68.7	66.7	67.7
GRPO	3B	53.0	52.9	71.2	59.0	70.4	62.1	60.4	64.3
GRPO	7B	64.4	69.2	84.2	72.6	78.4	70.9	68.8	72.7
Majority voting@4	3B	52.1	52.8	69.8	58.2	64.7	60.1	58.1	59.1
Majority voting@4	7B	63.2	67.6	83.2	71.3	75.6	68.9	66.8	67.8
Visual Para-Thinker	3B	55.0	54.4	72.6	60.7	75.9	63.6	62.2	67.2
Visual Para-Thinker	7B	65.7	70.8	85.4	74.0	82.2	71.3	71.0	74.8
Visual Para-Thinker++	3B	**60.7**	**69.5**	**74.8**	**68.3**	**80.0**	**69.5**	**64.0**	**71.2**
Visual Para-Thinker++	7B	**68.2**	**74.0**	**88.3**	**76.8**	**85.7**	**74.9**	**73.8**	**78.1**

specific advantages of Eq. 4.

4 Experiments

4.1 Setup

Benchmarks. We evaluate on six visual benchmarks that stress different reasoning dimensions: **V*** (Wu and Xie, 2023) for high-resolution visual search, **CountBench** (Paiss et al., 2023) and **Pixmo** (Deitke et al., 2024) for precise object counting, **MMVP** (Tong et al., 2024) for fine-grained perception, **RefCOCO / RefCOCO+ / RefCOCOg** (Yu et al., 2016) for referring-expression grounding, and **HallusionBench** (Guan et al., 2024) for visual hallucination.

Baselines. We compare against (i) the base MLLM (Qwen2.5-VL-3B (Bai et al., 2025)) with greedy decoding, (ii) the same base model with long CoT, (iii) self-consistency (Wang et al., 2023) with K samples at a comparable token budget, (iv) Para-Thinker (Wen et al., 2025) (parallel thinking without role separation), (v) a multi-agent debate (Du et al., 2023) variant of the same MLLM, and (vi) a scaled-up Qwen2.5-VL-7B variant reported for reference in Appendix A.

Metrics. We report task-specific accuracy/AP, total output tokens per example (Tok.), and wall-clock rollout latency.

4.2 Main Results

Table 1 reports results across counting, fine-grained perception, and hallucination tasks. At the 3B scale, Visual Para-Thinker++ lifts the perception-and-hallucination average from 57.7 on the Qwen2.5-VL-3B backbone to 71.2 (+13.5), and outperforms every peer-scale baseline on every benchmark. Long-CoT-style sequential SFT and Majority Voting@4 leave the counting average essentially unchanged ($57.0 \rightarrow 58.0/58.2$), and on-policy single-advantage RL (GRPO) closes the perception gap but barely moves counting, indicating that the bottleneck is the underlying single-trajectory perceptual commitment rather than inference compute. Para-Thinker, the closest parallel-thinking baseline without role separation, reaches 60.7 and 67.2 on the counting and hallucination averages; Visual Para-Thinker++ adds another +7.6 and +4.0 on top of it under the same multi-path inference

Table 2: Referring-expression grounding accuracy (Acc@0.5, %) on the RefCOCO family. Numbers in the first three columns are reproduced from Wen et al. (2025); Visual Para-Thinker++ is trained on the same backbone (Qwen2.5-VL-3B) with the same training data. **Bold** = best in the row.

Split	Qwen2.5-VL-3B	Sequential	Visual Para-Thinker (3B)	Visual Para-Thinker++ (3B)
RefCOCO val	85.4	85.7	88.1	**90.5**
RefCOCO testA	86.1	87.3	89.8	**91.6**
RefCOCO testB	83.0	84.2	87.7	**90.4**
RefCOCO Avg	84.8	85.7	88.5	**90.8**
RefCOCO+ val	76.5	79.2	83.5	**87.1**
RefCOCO+ testA	82.4	84.0	87.2	**89.4**
RefCOCO+ testB	71.4	75.9	77.0	**80.9**
RefCOCO+ Avg	76.7	79.7	82.6	**85.8**
RefCOCOg val	84.7	86.1	87.3	**89.8**
RefCOCOg test	83.1	84.0	86.9	**89.8**
RefCOCOg Avg	83.9	85.0	87.1	**89.8**

Table 3: Efficiency comparison on the V* benchmark. "Majority Voting" generates four trajectories per query. "Ours (w/o reuse)" and "Ours (w/ reuse)" refer to our multi-agent rollout engine (§3.6) without and with KV-cache reuse across role segments.

Method	Time (s) ↓	Throughput (token/s) ↑
Qwen2.5-VL-3B	266	58
Sequential	481	53
Majority Voting	1197	49
Ours (w/o reuse)	364	104
Ours (w/ reuse)	312	123

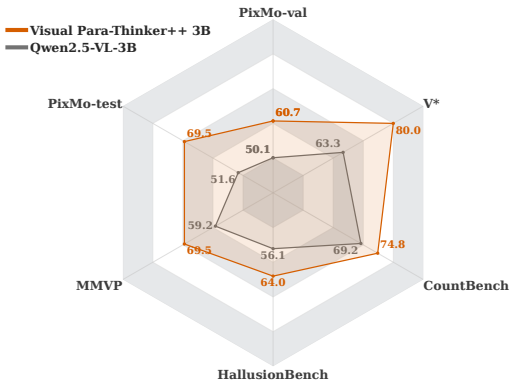

Figure 6: Per-benchmark comparison of Visual Para-Thinker++ (3B) against the Qwen2.5-VL-3B backbone on six vision-centric axes drawn directly from Table 1.

template, isolating the contribution of role-specific optimisation.

The largest gains are on **Pixmo-test** (+17.9), **V*** (+16.7), and **HallusionBench** (+7.9): exactly the benchmarks where the dominant error is early commitment to a wrong region or a hallucinated object. Counting (CountBench) and fine-grained verification (MMVP) show smaller but consistent gains (+5.6 and +10.3), consistent with the Scan-order pattern reducing single-direction missing-instance errors. Figure 6 visualises this margin on six vision-centric axes. The qualitative conclusions transfer to the 7B backbone (Appendix A), where Visual Para-Thinker++-7B remains competitive with frontier closed-source models on V*; the consistent margins across two scales suggest that the design is orthogonal to backbone size.

Grounding (RefCOCO family). We further evaluate referring-expression grounding on the RefCOCO/+/g splits (Acc@0.5; Table 2). Visual Para-Thinker++ outperforms both the Qwen2.5-VL-3B backbone and the SFT-aligned Visual Para-Thinker on every split. The improvement over Visual Para-Thinker is consistent across all nine entries (between +1.8 and +3.9 points), with the largest residual gain on the harder RefCOCO+ splits (val +3.6, testB +3.9), where attribute-and-relation cues—rather than pure spatial cues—play a larger role. This is consistent with the picture from §4.2: when the dominant error mode is early commitment to an incorrect region or attribute, role-decoupled optimisation over four parallel Worker reasoning paths reconciled by the Summary Agent yields gains that single-trajectory CoT and shared-advantage RL cannot reach.

4.3 Ablation Studies

How should R_w and R_o be composed? Table 4 compares the three reward-composition paradigms of Fig. 4 under an identical Worker-reward signal R_w. Naive Sum is the worst of the three and falls

Table 4: Cross-paradigm comparison of reward composition on the Qwen2.5-VL-3B backbone using an identical Worker-reward signal R_w and the same Stage-1 checkpoint. **Bold** = best.

Composition	Reward → advantage	CountBench	V*	MMVP	HallusionBench
Naive Sum	$R_o + \sum_i R_w^{(i)} \rightarrow A_{\text{shared}}$	58.7	66.4	61.3	50.9
Conditional Sum	$R_o + R_o \cdot \sum_i R_w^{(i)} \rightarrow A_{\text{shared}}$	62.9	69.8	65.1	55.2
Role-decoupled A (**Ours**)	$R_o \rightarrow A_{\text{out}};\ R_w^{(i)} \rightarrow A_w^{(i)}$	**66.2**	**72.8**	**68.7**	**59.0**

Table 5: Factorial ablation of the two design switches in Role-Decoupled Multi-Agent Optimization (§3.5). R_w **(Worker reward)**: ✓ enables the per-Worker local-quality signal $R_{\text{worker}}^{(i)}$ on top of the outcome reward; ✗ uses outcome reward only. **Role-decoupled** A **(advantage)**: ✓ assigns role-specific advantages following Eq. 4 (A_{out} for Main/Summary tokens; $A_{\text{out}} + \lambda A_{\text{worker}}^{(i)}$ for Worker i's tokens); ✗ uses a single shared advantage on every token.

Worker reward R_w	Role-decoupled advantage A	CountBench	V*	MMVP	HallusionBench
✗	✗	60.4	68.1	62.7	52.7
✓	✗	63.5	70.6	65.9	55.8
✗	✓	64.7	71.2	66.8	56.9
✓	✓	**66.2**	**72.8**	**68.7**	**59.0**

below the outcome-only baseline (R_w off, see the (✗, ✗) row of Table 5), confirming the gradient-coupling failure mode of §3.5. Conditional Sum recovers most of this loss but is still bounded by the shared-advantage bottleneck. Composing the two rewards at the *advantage* level, as in our role-decoupled scheme, yields the largest and most consistent gains across all four benchmarks.

Are R_w and role-decoupled A each necessary? Table 5 factorises Role-Decoupled Multi-Agent Optimization into two orthogonal switches: Worker reward R_w on/off and role-decoupled advantage on/off. Role-decoupled advantage alone (✗, ✓) already lifts CountBench by $+4.3$ and HallusionBench by $+4.2$ over the (✗, ✗) baseline, indicating that gradient routing across role segments is the dominant lever; adding R_w on top contributes a further $+1.5$ to $+2.1$, with the largest residual gain on MMVP. Adding R_w on a shared advantage (✓, ✗) yields a smaller and less consistent gain, reproducing the gradient-coupling pattern of Naive Sum. The two design choices are therefore complementary: R_w supplies the local signal, and role-decoupled A keeps it from polluting the Main and Summary gradients.

Sensitivity to the Worker advantage weight λ. Setting λ=0 in Eq. 4 recovers an outcome-only RL run equivalent to GRPO over the multi-agent trajectory, trailing the full method by 3.1–4.4 points across V*, CountBench, and HallusionBench. Increasing λ from 0 to 0.5 monotonically improves all three; pushing λ to 1.0 or 2.0 over-weights the noisy Worker reward and partially erases the gain. We use λ=0.5 throughout (full sweep in Appendix C, Table 7).

Effect of the number of Worker Agents K. Reducing K from 4 to 2 drops V* by 2.7 and CountBench by 3.1, while increasing K to 8 yields negligible gains ($+0.3$ on average) at nearly doubled rollout cost, confirming K=4 as the accuracy–efficiency sweet spot under our fixed allocation protocol.

4.4 Efficiency Analysis

Table 3 reports the inference efficiency on the V* benchmark following the Visual Para-Thinker evaluation setting. With KV-cache reuse across role segments, Visual Para-Thinker++ keeps total inference time close to the base Qwen2.5-VL-3B model while substantially improving throughput over sequential and majority-voting parallel baselines.

5 Conclusion

We presented Visual Para-Thinker++, a single-policy multi-agent framework that instantiates one shared MLLM as Main, Worker, and Summary Agents under a fixed protocol, trained with Multi-Agent Capability Injection and Role-Decoupled Multi-Agent Optimization to reduce reward conflict across roles. Across six benchmarks, Visual Para-Thinker++ delivers consistent gains over single-trajectory and parallel baselines.

Limitations

Our study has several limitations. First, our primary experiments are conducted on a Qwen2.5-VL-3B backbone, with a complementary 7B run reported for reference in Appendix A; the scaling behaviour on substantially larger backbones, such as models above 30B parameters, and on alternative role organizations remains to be characterized. Second, the empirical study covers six benchmarks that emphasize visual search, counting, fine-grained perception, grounding, and hallucination detection, but does not yet examine broader multimodal settings such as document understanding, chart reasoning, or video-based decision making. Third, our current Main Agent uses a fixed task-allocation protocol with two task-dependent patterns: Block-based allocation and Scan-order allocation. We deliberately do not model the Main Agent as a free-form open-ended planner in this paper. Adaptive task allocation, including when to switch patterns, when to vary the number of parallel Worker reasoning paths, and how to compose more granular Worker roles, remains an important direction for future work. Fourth, although the native multi-agent inference engine reduces redundant prefill and KV cache overhead, horizontal scaling still increases overall decoding cost as additional Worker reasoning paths are introduced; in our 3B experiments the four-path configuration provides the best accuracy-cost trade-off, and this trade-off may shift for larger backbones. Fifth, the Worker reward is currently instantiated by an agreement-driven majority-vote heuristic, which can be brittle when most Worker reasoning paths share the same hallucination; the Summary reward is currently instantiated by the final-answer outcome reward, and we do not yet explicitly supervise the faithfulness of the Summary Agent's reconciliation. Stronger process supervision for both Worker and Summary roles is a natural extension. Finally, we do not yet isolate the Summary Agent's decision policy with a dedicated ablation; future work should separate the contribution of trace reading from that of evidence reconciliation, for instance by replacing the trained Summary segment with a non-learned majority vote at inference time. Addressing adaptive Main-Agent task allocation, stronger process supervision for Worker and Summary rewards, Summary-Agent decision-policy ablation, and broader evaluation across multimodal domains remain future work.

Ethics Statement

Our work aims to improve the reliability of multimodal reasoning systems by encouraging explicit exploration of multiple visual hypotheses through role-conditioned Worker Agents before producing a final answer. In principle, this can have positive downstream effects in settings where premature commitment to a single interpretation is costly, such as assistive interfaces, educational tools, and decision-support systems that rely on visual evidence. The native multi-agent inference engine is also designed to reduce computational cost relative to naive role-separated rollout through KV cache reuse, which may modestly reduce the marginal cost of multi-agent reasoning.

At the same time, improved visual reasoning can also amplify misuse risks. More capable multimodal models may be used to generate more persuasive but incorrect analyses of images, documents, or screenshots, including misleading explanations that appear internally coherent. Moreover, our method does not eliminate hallucinations; it only reduces them empirically on the evaluated benchmarks. In high-stakes applications, a more confident multi-agent system could still produce incorrect outputs, potentially with a stronger veneer of justification. For this reason, we do not view Visual Para-Thinker++ as a substitute for domain-specific verification, human oversight, or calibrated uncertainty estimation.

The training pipeline also raises practical and methodological considerations. First, the Worker reward used in Stage 2 relies on agreement among intermediate Worker traces, which may reinforce majority errors when all Worker reasoning paths share the same blind spot. Second, teacher-generated SFT traces may encode biases or stylistic artifacts from the upstream teacher system. Third, multi-agent reasoning increases total generated tokens even when inference is optimized, which has compute and environmental costs. Throughout this paper we therefore report both accuracy and inference cost, evaluate explicitly on hallucination-sensitive benchmarks, and caution against deploying Visual Para-Thinker++ in safety-critical scenarios without additional safeguards and external verification mechanisms.

References

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. Qwen2.5-vl technical report. *Preprint*, arXiv:2502.13923.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, YenSung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, and 31 others. 2024. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *Preprint*, arXiv:2305.14325.

Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, Dinesh Manocha, and Tianyi Zhou. 2024. Hallusionbench: An advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Yejin Choi, Jan Kautz, and Pavlo Molchanov. 2026. Gdpo: Group reward-decoupled normalization policy optimization for multi-reward rl optimization.

Chancharik Mitra, Brandon Huang, Trevor Darrell, and Roei Herzig. 2024. Compositional chain-of-thought prompting for large multimodal models.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, and 401 others. 2024. Gpt-4o system card.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 246 others. 2026. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Roni Paiss, Ariel Ephrat, Omer Tov, Shiran Zada, Inbar Mosseri, Michal Irani, and Tali Dekel. 2023. Teaching clip to count to ten.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model.

Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. 2024. Visual cot: Advancing multi-modal language models with a comprehensive dataset and benchmark for chain-of-thought reasoning.

Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. 2024. Eyes wide shut? exploring the visual shortcomings of multimodal llms. *Preprint*, arXiv:2401.06209.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. 2025. Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute. *Preprint*, arXiv:2509.04475.

Penghao Wu and Saining Xie. 2023. V*: Guided visual search as a core mechanism in multimodal llms.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. 2025. Kvcomm: Online cross-context kv-cache communication for efficient llm-based multi-agent systems.

Licheng Yu, Patrick Poirson, Shan Yang, Alexander C. Berg, and Tamara L. Berg. 2016. Modeling context in referring expressions.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale.

Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. 2024. Multimodal chain-of-thought reasoning in language models.

A Results on Qwen2.5-VL-7B

To verify that our conclusions are not specific to the 3B backbone, we repeat the main evaluation on Qwen2.5-VL-7B with the same training recipe (Table 6). Absolute accuracy rises across the board; the qualitative conclusions from the 3B experiments (consistent gains over long-CoT, self-consistency, Para-Thinker, and multi-agent-debate baselines, with the largest improvement on HallusionBench) carry over to 7B, and HallusionBench remains the benchmark on which Visual Para-Thinker++ most clearly separates from the strongest baseline.

B Training data and hyper-parameters

Backbone, training stages, and protocol. We instantiate Visual Para-Thinker++ on a Qwen2.5-VL-3B (Bai et al., 2025) backbone, with a complementary 7B run reported in Appendix A. Training proceeds in two stages: Stage 1 performs Multi-Agent Capability Injection under the context-isolation mask of Eq. 2; Stage 2 continues from the Stage-1 checkpoint and performs Role-Decoupled Multi-Agent Optimization with on-policy DAPO (Yu et al., 2025) using the role-specific advantages of Eq. 4. All main experiments instantiate the framework with a fixed four-path collaboration protocol, chosen to align with natural visual partitions and complementary scan-order reasoning. The Worker advantage weight λ is selected on validation as in Table 7. Train and validation splits are disjoint, with validation used exclusively for model selection and final evaluation.

Evaluation protocol details. Unless explicitly stated otherwise, all reported numbers are computed on the official validation or test splits provided by the corresponding benchmarks, using the benchmark-standard metrics and post-processing rules. For open-ended generation tasks, the final answer is extracted from the Summary Agent's output using the same normalization rule across methods. For grounding tasks, we follow the benchmark convention for box or region matching. Each timing measurement in the efficiency table is averaged over repeated runs on the same hardware. We interpret small absolute differences conservatively and do not claim significance beyond the resolution supported by these repeated measurements.

SFT data construction. Stage-1 traces are synthesised by a stronger MLLM teacher. The exact teacher system is withheld for double-blind submission and will be named in the camera-ready release together with the data-generation prompts. For each (v, q) pair, the teacher is prompted to produce $K+2$ aligned segments: a Main Agent task allocation chosen by task type from Block-based and Scan-order patterns, K Worker Agent traces each targeting a distinct visual sub-problem, and a Summary Agent trace that reconciles the Worker outputs into a final answer conditioned on the visual context. For the running V*-style example, the Worker sub-problems correspond to layout, counting, spatial relation, and verification. The segments are stored as one packed multi-agent trajectory annotated with role tokens, which the Stage-1 context-isolation mask of Eq. 2 then enforces. We filter the synthesised pool with three conservative rules: (i) reject malformed role sequences, including missing or duplicated role tokens and fewer than K Worker Agent traces; (ii) reject traces with no extractable final answer in the Summary segment; (iii) when a benchmark gold answer is available, reject traces whose Summary answer disagrees with the gold. We additionally stratify the retained pool by task family, covering visual search, counting, grounding, and hallucination, and by visual-token bin so that no single regime dominates. The total number of Stage-1 trajectories used in this paper is approximately $163{,}000$; training and validation splits are disjoint and validation is used exclusively for model selection.

RL hyper-parameters. We summarize the concrete training configuration used in the Stage-2 (Role-Decoupled Multi-Agent Optimization) script. The backbone is initialized from a Qwen2.5-VL-3B Stage-1 checkpoint. The RL trainer uses a fully on-policy DAPO variant in which `train_batch`=`ppo_mini_batch`=16 and the number of PPO epochs per on-policy batch is 1; the per-GPU micro-batch size is 1 for both policy optimization and log-probability computation. The Worker advantage weight (the parameter named `dtpo_lambda` in the training config; corresponds to λ in Eq. 4) is set to 0.5 in this configuration. The optimizer learning rate is 5×10^{-7}. KL is applied as a loss with coefficient 0.01 and low-variance KL type, while the reward-side KL penalty is disabled. Entropy coefficient is 0.02, clipping thresholds are 0.20 and 0.28, the auxiliary clip parameter is 3.0, and the advantage clip is 5.0.

The data loader uses 4 workers. The maximum

Table 6: Results on the Qwen2.5-VL-7B backbone with the same training recipe. The qualitative conclusions from the 3B experiments carry over to 7B. "MAC" denotes Multi-Agent Capability Injection (Stage 1) and "RDMAO" denotes Role-Decoupled Multi-Agent Optimization (Stage 2); underline marks the Stage-1-only intermediate checkpoint and **bold** marks the final two-stage Visual Para-Thinker++.

Method	V*	CountB.	RefCOCO	RefCOCO+/g	Hallus.	Tok.
Base MLLM (greedy, 7B)	64.5	56.4	79.7	73.1 / 74.6	47.8	134
+ Long CoT	68.7	59.6	80.8	74.5 / 75.9	50.6	605
+ Self-Consistency (K=4)	70.4	62.0	81.7	75.1 / 76.4	52.3	2410
+ Para-Thinker (K=4)	72.5	63.5	82.6	76.0 / 77.2	53.9	2456
+ Multi-Agent Debate	73.6	64.5	83.0	76.4 / 77.6	55.4	2854
Visual Para-Thinker++ (MAC only, 7B)	74.9	66.7	84.4	77.6 / 78.6	58.0	2503
Visual Para-Thinker++ (MAC+RDMAO, 7B)	**80.2**	**71.5**	**88.1**	**82.0 / 82.9**	**64.7**	2540

prompt length is 18,000 tokens and the maximum response length is 2,048 tokens. Rollout uses vLLM with temperature 1.0, $n = 8$ generated samples per group, block size 16, maximum batched tokens 64,000, tensor-parallel size 1, disabled chunked prefill, eager execution enabled, and GPU memory utilization target 0.4. The native multi-agent inference engine is enabled with four role-trigger tokens and `parthink_size=4` (one trigger per Worker Agent). Dynamic group filtering is enabled using the outcome reward as the filtering metric, with up to 100 generated batches considered during filtering; concretely, to keep the Summary Agent from degenerating into a naive majority voter we retain only Stage-2 samples with at most two correct Worker Agents out of K=4, and we stratify by task family and by visual-token bin for data balance. The training run uses 8 GPUs on a single node and saves checkpoints every 50 steps.

Hardware and software. The experiments are run on a single machine equipped with 8× NVIDIA H20 GPUs. The rollout engine uses float16 generation, and the training system enables padding removal for the actor model, parameter offloading for the reference model, and a custom vLLM-based scheduler to support sequence fork/merge for the multi-agent collaboration protocol. These implementation choices are important for reproducing the inference time and throughput reported in Table 3.

C Extended ablations

The sensitivity of the decoupling weight λ is reported in Table 7.

Table 7: Sensitivity to the Worker advantage weight λ in the role-decoupled advantage objective (Eq. 4). λ=0 recovers an outcome-only RL run (only A_{out} is propagated to Worker tokens); large λ over-weights the Worker reward and amplifies its noise. We use λ=0.5 for all main-text experiments.

λ	V*	CountBench	HallusionBench
0.0	69.7	62.8	54.6
0.25	71.9	65.1	57.3
0.5	**72.8**	**66.2**	**59.0**
1.0	72.4	65.7	58.4
2.0	70.8	64.0	56.9

D Task Allocation Patterns: Details and Data Examples

This appendix expands on the two fixed allocation patterns introduced in §3.3 and shows two concrete Stage-1 (MAC) data examples used to teach the shared policy how each pattern is realised.

D.1 Block-based Allocation

Pattern definition. Given an image v and the four-path collaboration protocol used in this paper, the Main Agent partitions v into K=4 disjoint visual blocks. By default the blocks correspond to the four image quadrants (top-left, top-right, bottom-left, bottom-right). For tasks where the natural visual structure is not aligned with quadrants (e.g., elongated horizontal scenes or images with a strong centre bias), the Main Agent emits a task-specific partition—for instance, a 1×4 horizontal strip layout, a 2×2 centre-vs-periphery layout, or a foreground/background separation—chosen from a small fixed pool. Worker Agent i is then conditioned on v, q, the Main Agent's segmentation m, and its role token r_{worker_i}, and is asked to collect local visual evidence within its assigned block while ignoring tokens from sibling Workers (Eq. 2).

When this pattern fits. Block-based allocation is well suited to tasks dominated by local visual evidence collection: high-resolution visual search where the answer is contained in one region (e.g., V*), referring-expression grounding (RefCOCO/+/g), fine-grained perception (MMVP), and attribute or hallucination verification (HallusionBench). The disjointness of the blocks provides a hard inductive bias that complementary regions are scanned by complementary Workers, which the Stage-1 context-isolation mask then consolidates by preventing one Worker from copying another's reasoning.

Data example. Stage-1 training data follow the packed multi-agent trajectory format introduced in §3.2. The teacher first generates per-block captions, the four blocks are then linked into separate Worker reasoning paths via $\langle \text{think}_i \rangle / \langle / \text{think}_i \rangle$ tags, and the Summary Agent's reconciled answer is wrapped in $\langle \text{summary} \rangle / \langle / \text{summary} \rangle$. We then apply the conservative filtering rules in Appendix B (malformed sequences, missing answers, gold-answer disagreement) before training. A representative MMVP-style block-based example is shown below.

Data Example (Block-based allocation)

<|User|><image> *What is the color of the container holding the broccoli and meatball? A. yellow B. pink C. blue D. green. This is a multiple-choice question, please select one option. You FIRST think about the reasoning process as an internal monologue and then summarize the reasoning process to get the final answer. The summary process MUST BE enclosed within* <summary></summary> *tags. The final answer MUST BE put in* \boxed{}.<|Assistant|><vllm_pad>

<think1>
Top-Left: This quadrant shows a pink container holding a slice of white bread with a dollop of white spread, several whole almonds, and two dried fruit slices, one of which is round and brown. The container has a red paper liner.
</think1>

<think2>
Top-Right: This quadrant shows a pink container filled with yellow fruit pieces, including what appear to be pineapple chunks and mandarin orange segments. The container is set against a blue background.
</think2>

<think3>
Bottom-Left: This quadrant shows a yellow container holding a round, brown meatball covered in red sauce and a large floret of green broccoli. The container has a yellow rim.
</think3>

<think4>
Bottom-Right: This quadrant shows a dark brown, round cookie with a textured surface. Part of a blue container edge is visible, and another similar cookie is partially visible behind the first one.
</think4>

<summary>
By analyzing the four parallel reasoning paths above, only Worker 3 reports a yellow container that simultaneously holds a meatball and broccoli. The final answer is $\boxed{\text{A. yellow}}$.
</summary>

D.2 Scan-order Allocation

Pattern definition. Under Scan-order allocation, the Main Agent does not partition the image; instead it assigns each Worker Agent a distinct global scanning trajectory. The four trajectories used in this paper are left-to-right, right-to-left, top-to-bottom, and bottom-to-top. Each Worker Agent retains a global receptive field over the full image and traverses it in its assigned order, enumerating or verifying objects along the way. The disagreements between scan orders, rather than spatial coverage, become the source of complementary evidence.

When this pattern fits. Scan-order allocation is well suited to tasks where the dominant error mode is missing or double-counting an instance under a single scanning direction: precise object counting (Pixmo, CountBench), global verification (e.g., *is every object in the image still present after a perturbation?*), and spatial-consistency checking (left-of, above-of relations). Permuting the traversal order across Workers is a cheap way to expose order-dependent miscounting at training time, which the

Summary Agent then learns to reconcile rather than majority-vote on.

Data example. Stage-1 traces under Scan-order allocation use the same packed trajectory format and the same filtering rules as Block-based traces; the only difference is that each $\langle \text{think}_i \rangle$ block enumerates the same scene under a distinct scanning order. A representative counting example is shown below.

Data Example (Scan-order allocation)

<|User|><image> *How many people are in the image? You FIRST think about the reasoning process as an internal monologue. In your thoughts, try to locate and count the objects one by one or group by group to ensure accuracy. Then, summarize the counting process to get the final number. The summary process MUST BE enclosed within* <summary></summary> *tags. The final answer MUST BE a single integer put in* \boxed{}.<|Assistant|><vllm_pad>

<think1>
Scanning Left-to-Right: 1. Found a young man in a white shirt looking down on the far left. That makes 1. 2. Next, a young man in a white shirt pouring water from a bottle. That makes 2. 3. Behind him, a man in a dark grey "Zoo York" t-shirt and a cap. That makes 3. 4. In the centre foreground, a young man in a white shirt with his hand on his chin. That makes 4. 5. To his right, a young man in a white shirt with a "NUC" logo. That makes 5. 6. Behind him, a young man in a grey t-shirt looking to the right. That makes 6. 7. On the far right, a partially visible person in a dark shirt. That makes 7. Total count: 7.
</think1>

<think2>
Scanning Top-to-Bottom: Starting from the top, I first encounter the man in the "Zoo York" t-shirt and cap. That makes 1. To his right, the young man in the grey t-shirt. That makes 2. To the far right, the partially visible person in a dark shirt. That makes 3 . . . Total count: 7.
</think2>

<think3>
Scanning Right-to-Left: Starting from the far right, I see a partially visible person in a dark shirt. That makes 1. Moving left, I see a young man in a grey t-shirt. That makes 2. Next, a young man in a white "NUC" shirt. That makes 3. In front of him, a young man in a white shirt with his hand on his chin. That makes 4. Behind him, a man in a "Zoo York" t-shirt and cap. That makes 5 . . . Total count: 7.
</think3>

<think4>
Scanning Bottom-to-Top: Starting from the bottom, I first find the young man looking down on the far left. That makes 1. Moving up, I see the young man with his hand on his chin. That makes 2. To his right, the young man in the "NUC" shirt. That makes 3. Above him, the young man in the grey t-shirt . . . Total count: 7.
</think4>

<summary>
All four scans converge on the same count: every individual is enumerated exactly once across the four orderings, which mutually verify against double-counting and omission. The final answer is $\boxed{7}$.
</summary>

D.3 Pattern selection at inference time

By default the Main Agent emits Block-based allocation. It switches to Scan-order allocation when the task is dominated by counting or by global verification. The selection signal is a short tag inside the Main segment m, learned during Stage-1 from the teacher-annotated allocation choice, so all four Worker Agents and the Summary Agent see the chosen pattern as part of their context. We deliberately keep the pattern set small (two patterns, four canonical scan orders, four fixed quadrant blocks) so that the role-decoupled optimisation in §3.5 operates on a small number of stable inductive biases rather than on a free-form planner output.