ReGRPO: Reflection-Augmented Policy Optimization for Tool-Using Agents

Binjie Zhang and Mike Zheng Shou*

Show Lab, National University of Singapore
binjie97@u.nus.edu

Abstract. Tool-augmented vision–language models (VLMs) can solve multimodal, multi-step tasks by calling external tools, yet they remain fragile in practice. Existing works have two common gaps. Supervised fine-tuning (SFT) is built mostly on successful trajectories and offers little signal for recovery after tool failures, while sparse trajectory-level RL rewards provide limited guidance on which step failed and how to repair it. We introduce ReGRPO (Reflection-augmented Group Relative Policy Optimization), a framework that learns reflection-guided correction in tool-using agents. ReGRPO starts with a structured reflective data engine: we execute near-miss actions to collect grounded failure observations, then build Reflection-of-Thought triplets (ErrorType, Evidence, FixPlan) paired with corrected actions for warm-start SFT. We then optimize reflection tokens and corrective actions jointly within local trajectories using group-relative advantages, and include a reflection-cost term to reduce unnecessary reflection. Experiments on GTA and GAIA show that, under the same backbone and tool suite, ReGRPO consistently outperforms strong open-source baselines and achieves the best results among the compared open-source controllers. Code and RoT data are available at https://github.com/showlab/ReGRPO.

Keywords: Multimodal agents · Tool-use learning · Reflection · Reinforcement learning

1 Introduction

External tools such as web search, OCR, table readers, PDF parsers, code execution, and visual operators expand the capability of vision-language models (VLMs) beyond in-context prompting [3,6,14,23]. In these settings, success depends not only on the final answer, but also on planning and executing grounded intermediate steps, including which tool to call, when to call it, and how to set its arguments from multimodal evidence.

One common approach is to train trajectory-tuned controllers on synthetic tasks and verified traces. MAT-AGENT [5] follows this approach by generating multimodal tool-use tasks, executing tools to collect successful trajectories, and applying supervised fine-tuning (SFT) to learn tool invocation. This strategy is effective, but it also ties the controller closely to supervised traces. When the policy deviates, for example under a new PDF layout or receipt style, the training signal offers limited guidance for recovery.

* Corresponding author.

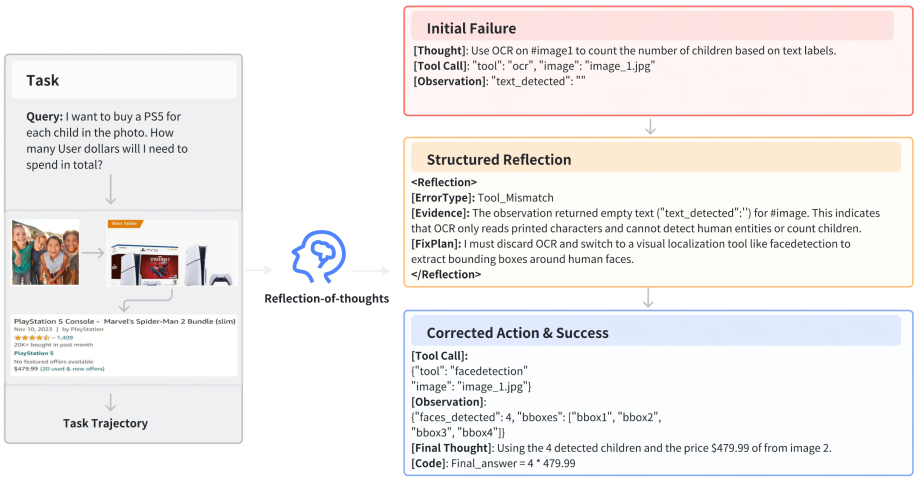

Fig. 1: Pipeline of the Structured Reflective Data Engine. Given a task trajectory, we first induce a tool failure (for example, OCR on a face image returns empty text). A teacher model (GPT-4o by default) then generates a structured Reflection-of-Thought with explicit ErrorType, Evidence, and FixPlan, which explains the failure and proposes the next action (for example, switching to face detection). The agent executes the corrected action to recover and finish the task. This Error, Reflection, and Correction loop converts raw failures into grounded supervision for recovery.

A complementary direction uses self-exploration. SPORT [9] alternates between sampling tool-use steps and verifying them, then converts rollouts into step-wise preferences without human labels. This introduces process-level feedback, but the supervision remains relatively unstructured. For example, SPORT does not explicitly encode multimodal chain-of-thought, grounding tags that link language to regions or cells, or verifier rationales. As a result, the learned preferences are often hard to interpret and can fail under distribution shift.

These approaches share two limitations. SFT-only tuning on expert trajectories tends to saturate quickly because the loss depends heavily on one teacher trace and provides weak signals for local repair. Standard reinforcement learning also provides limited recovery-oriented supervision in long-horizon tasks, because a scalar failure reward does not identify which decision should be revised.

We therefore train tool-using VLM agents with reflection-augmented reinforcement learning, where diagnostic reflections are optimized as explicit recovery signals. Instead of treating reflection as a test-time prompting heuristic, we model it as a learnable variable that links a failed action to its correction.

We introduce ReGRPO (Reflection-augmented Group Relative Policy Optimization) to realize this idea. On the data side, ReGRPO builds a Structured Reflective Data Engine that converts tool-execution failures into grounded Reflection-of-Thought (RoT) triplets. The engine perturbs MAT-AGENT trajectories [5], executes the perturbed actions to obtain real failure observations, and then generates structured reflections with ErrorType,

Evidence, and FixPlan. On the optimization side, ReGRPO jointly optimizes reflection and correction tokens within the same local trajectory objective, which gives the policy step-level supervision beyond final success or failure. ReGRPO keeps the standard GRPO optimizer and adds structured reflection parameterization to enable end-to-end policy learning.

At inference time, ReGRPO follows the same principle that failures should trigger explicit local repair rather than blind continuation. We use a lightweight single-path, zero-verifier setup with a deterministic trigger, which opens a reflection-correction block only when needed. This design allows recovery from tool errors while keeping deployment efficient.

On GTA [25] and GAIA [16], under the same backbone and tool suite as MAT-AGENT and SPORT, ReGRPO consistently improves overall accuracy.

In summary, this paper makes three contributions:

- We build a *Structured Reflective Data Engine* that converts tool-execution failures into grounded Reflection-of-Thought supervision with ErrorType, Evidence, and FixPlan, paired with corrected actions.
- We present *ReGRPO*, which adopts the standard GRPO optimizer while adding structured reflection trajectory parameterization and a zero-verifier trigger, enabling joint optimization of reflection and correction tokens under reproducible single-path deployment.
- Under the same backbone and tool settings, ReGRPO achieves the *strongest results* among the compared open-source controllers on GTA and GAIA.

2 Related Work

2.1 Multi-Modal Agents

Multimodal agents [8, 11, 24, 29, 30] can solve complex problems via external tools and APIs. For example, CLOVA [4] uses LLMs as controllers to compose off-the-shelf visual tools. ViperGPT [23] uses code-generation models to compose vision-and-language models into subroutines to produce results for arbitrary queries. VideoAgent [3] adopts multi-step reasoning, where the agent selects tools according to intermediate observations.

Though LLM-driven methods can achieve strong results, VLM-driven agents [18, 26, 31] are often more efficient for visual tasks because the controller directly consumes images or videos and tool outputs. For example, GenArtist [26] proposes a unified image generation and editing system coordinated by a multimodal large language model (MLLM) agent.

In addition, other works [10,27,28] use models to generate AI feedback that improves performance. For example, LLaVA-Critic [28] presents a high-quality dataset tailored to follow instructions in complex evaluation settings, providing quantitative judgments and accompanying reasoning.

2.2 Datasets for Tool-Using Agents

The reasoning ability of VLM-driven agents is often weaker than that of large text-only LLMs. To bridge this gap, recent works synthesize tool-usage data to tune open-source VLMs [12, 13, 24]. For example, DEDER [2] uses in-context learning to generate trajectories and distills chain-of-thought reasoning from LLMs to smaller models. Lumos [29] converts ground-truth reasoning steps from existing benchmarks into tool-usage trajectories. TASKBENCH [21] samples trajectories from pre-defined graphs. MAT-AGENT [5] scales up trajectory tuning for open VLM controllers with a diverse tool suite and synthetic tasks.

2.3 Reflective and Self-Correcting Agents

The concept of reflection—prompting a model to critique its own outputs—has been widely explored in LLMs. Reflexion [22] and similar frameworks [1, 15] use verbal feedback and episodic memory to improve performance over multiple trials at inference time. However, these methods typically treat reflection as a frozen prompting strategy or rely on external scalar feedback, without optimizing the reflection generation process itself. In contrast, ReGRPO explicitly learns how to generate a diagnostic reflection and how that reflection guides the next corrective tool call under an explicit trigger mechanism. Rather than relying on inference-time trial-and-error alone, ReGRPO trains the model to produce grounded diagnostics that enable successful recovery, internalizing the correction loop into the policy. We evaluate this approach on GTA and GAIA to assess effectiveness and sample efficiency in tool-using settings.

3 Method

Existing training strategies for tool-using agents have two key issues. First, supervised trajectories rarely include recovery steps [5]. As a result, the model learns only successful traces. A small mistake can then cascade without guidance on how to fix it. For example, in receipt QA, SFT teaches the correct `object_loc` followed by OCR. At test time, a slight layout shift may lead to a nearby crop and empty OCR, but the training data provides no corrective signal (e.g., expand the box or re-localize). Second, standard RL provides weak localization. A multi-step document QA run can fail after a wrong crop, then an empty OCR, then an incorrect answer. A final reward of 0 does not reveal which step was wrong or how to correct it. This ambiguity makes learning slow and unstable.

We introduce **ReGRPO** (Reflection-Augmented Group Relative Policy Optimization), which explicitly learns *how* to generate diagnostic reflection and *how* reflection guides the next corrective tool call in local trajectories. As shown in Figure 2, ReGRPO follows three principles. First, it uses a Structured Reflective Data Engine that converts execution failures into grounded diagnostic triplets. Second, it applies ReGRPO to optimize the generation of reflection steps that lead to successful corrections. Third, it uses a Zero-Verifier Inference Stage with single-path execution and no external verifier calls. We evaluate the resulting agent on GTA and GAIA benchmarks to assess overall tool-use performance.

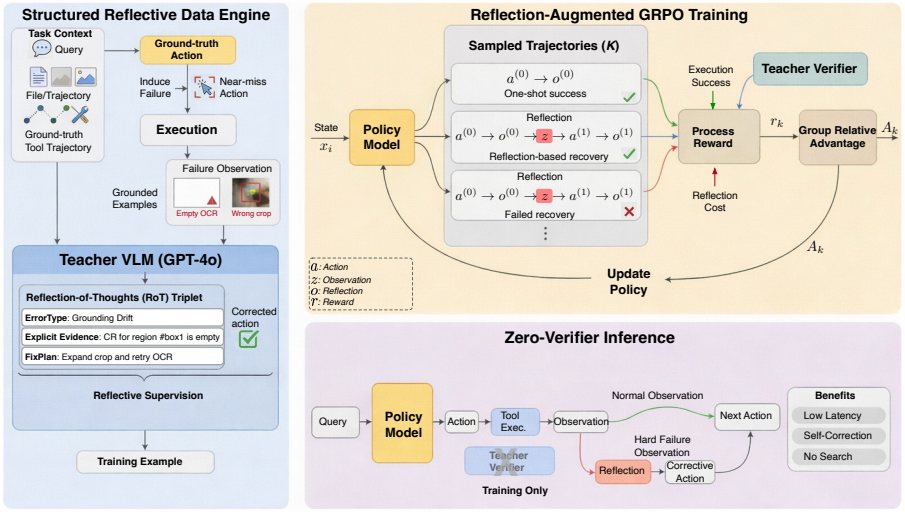

Fig. 2: Overview of ReGRPO. (1) Structured Reflective Data Engine: from multimodal inputs and a successful action, synthesize a near-miss failure (wrong crop/tool/argument), execute it to obtain grounded failure observations (e.g., empty OCR or tool error), then use a teacher VLM (e.g., GPT-4o) to generate a structured Reflection-of-Thought triplet (ErrorType, Evidence, FixPlan). Pair the reflection with the corrected action to form reflective supervision (failure action, failure observation, reflection, corrected action), and warm-start SFT on these trajectories. (2) ReGRPO training: form groups of candidate local trajectories, including one-shot successes and reflection-based recoveries $a^{(0)} \to o^{(0)} \to z \to a^{(1)} \to o^{(1)}$; combine execution success, an optional teacher-derived verifier score (training only, computed deterministically from the teacher's RoT metadata with no in-loop LLM call), and reflection cost into a reward; compute group-relative advantages to update both reflection and correction tokens. (3) Zero-Verifier Inference Stage: single-path execution with a deterministic trigger that opens a local reflection-correction block only when failure evidence appears, enabling efficient recovery without external verifier calls.

The remainder of this section presents the problem setup (Sec. 3.1), the Structured Reflective Data Engine (Sec. 3.2), ReGRPO training (Sec. 3.3), and the inference strategy (Sec. 3.5).

3.1 Problem Setup

A task is given by (Q, F), where Q is a user query and F is a set of files or images. At step i, the agent observes history $h_i = \{(t_j, c_j, o_j)\}_{j=1}^{i-1}$. Standard agents predict an action $a_i = (t_i, c_i)$, where t_i is a thought and c_i is a tool call.

In ReGRPO, the action space includes an additional reflection step z_i. When an action $a_i^{(0)}$ fails and produces observation $o_i^{(0)}$, the agent may generate reflection z_i

before attempting correction $a_i^{(1)}$. The local trajectory segment is

$$\tau_i = \begin{cases} (a_i^{(0)}, o_i^{(0)}) & \text{if success} \\ (a_i^{(0)}, o_i^{(0)}, z_i, a_i^{(1)}, o_i^{(1)}) & \text{if reflection triggered} \end{cases} \tag{1}$$

The goal is to learn a policy π_θ that both acts and reflects to self-correct when necessary.

3.2 Structured Reflective Data Engine

Existing trajectory cloning methods and outcome-based reward models provide only coarse supervision in multimodal environments. When a trajectory fails, a scalar penalty does not reveal *which* step or parameter caused the error, so the model cannot learn how to repair it. To address this, we introduce a Structured Reflective Data Engine that converts failures into explicit causal evidence. Instead of treating failed actions as generic negatives—which can worsen off-policy shifts—we reformulate them as instructional "Error-Reflection-Correction" trajectories, giving the agent concrete recovery supervision from the outset.

Failure Induction. To teach recovery, the model must observe concrete failure states rather than only successful traces. We therefore construct *initial failures* that are realistic yet recoverable, so the agent can learn how errors arise and how to fix them. Specifically, starting from a ground-truth step a_i^* (from MM-Traj [5]), we synthesize plausible "near-miss" actions a_i^{fail} by perturbing tool choices or arguments (e.g., shifting a bounding box, selecting an adjacent table column, or calling a mismatched tool). We then *execute* a_i^{fail} in the sandbox to obtain real failure observations o_i^{fail} (e.g., API exceptions, empty OCR outputs, or irrelevant crops). These executed failures provide grounded error signals that the Reflective Data Engine links to diagnostic reflections and corrections. We organize the resulting annotations using a Reflection-of-Thought (RoT) reflection z_i. Each annotation is a RoT triplet $(a_i^{fail}, o_i^{fail}, z_i)$ paired with the corrected action a_i^*, which together explicitly encode the failure, structured reflection, and recovery target.

Reflection Annotation. To bridge the causal gap between the faulty action a_i^{fail} and the failure observation o_i^{fail}, we use a teacher vision-language model (GPT-4o by default) to generate a structured RoT reflection z_i, conditioned on the failure context $(h_i, a_i^{fail}, o_i^{fail})$. To prevent the agent from generating free-form, hallucinated excuses, z_i is strictly constrained to a triplet schema:

- **ErrorType**: A categorical diagnosis of the failure (e.g., `ToolMismatch`, `Arg Invalid`, `GroundingDrift`, `InfoInsufficient`).
- **Evidence**: A mandatory reference to the visual or textual observation that triggered the error. For instance, rather than a generic "the tool failed", the model must explicitly ground its reasoning: "The OCR output for the specific region `#box1` returned empty text, indicating an incorrect crop."
- **FixPlan**: A concrete, actionable natural language strategy to correct the error and reach the target state (e.g., "Expand the bounding box slightly to cover the text" or "Switch to a visual localization tool").

Correction Pairing and Internalization. Finally, the original ground-truth action a_i^* is appended as the corrected action, closing the causal loop. This pipeline converts the MM-Traj dataset [5] into a supervised corpus of augmented trajectories: $\tau_i^{reflective} = (x_i, a_i^{fail}, o_i^{fail}, z_i, a_i^*)$. During supervised fine-tuning (SFT), the model is forced to maximize the conditional likelihood $P(z_i, a_i^* \mid x_i, a_i^{fail}, o_i^{fail})$. Consequently, the agent internalizes the multimodal error diagnostic capability at the parameter level, enabling subsequent training-time exploration over reflective recoveries in ReGRPO.

3.3 Reflection-Augmented Group Relative Policy Optimization

Dynamic Local Trajectory Formulation. For a given state context x_i, rather than scoring atomic actions, we form a group of K candidate local trajectories $\{\tau_i^{(k)}\}_{k=1}^K$ and score each under the current policy π_θ. The structure of each candidate trajectory depends on the intermediate environmental feedback: a candidate begins with an initial action $a_i^{(0)}$ and its observation $o_i^{(0)}$. If the execution succeeds, the trajectory terminates early: $\tau_i = (a_i^{(0)}, o_i^{(0)})$. Otherwise, if $o_i^{(0)}$ indicates a hard failure (e.g., execution error, empty visual crop) or low task–observation consistency, the policy generates a diagnostic reflection z_i and a corrective action $a_i^{(1)}$, forming the recovery segment in (1).

Reflection-Aware Process Reward. To encourage meaningful exploration while penalizing infinite loops or verbosity, we define a composite reward function $R(\tau)$ that balances task success with execution efficiency:

$$R(\tau) = \lambda_{\text{exec}}\, \mathbf{1}\{\text{success}\} - \eta\, C(\tau) + \lambda_{\text{val}}\, V(x_i, \tau). \tag{2}$$

Here, $\mathbf{1}\{\text{success}\}$ is deterministic environment feedback indicating task completion, and $C(\tau)$ penalizes unnecessary reflection length. The first two terms already form a complete *verifier-free* objective. $V(\cdot)$ is an *optional* training-only verifier value, computed *deterministically* from the active record's RoT metadata (no model is queried inside the RL loop): it can be enabled with $\lambda_{\text{val}} > 0$ for extra stabilization, or disabled with $\lambda_{\text{val}} = 0$ without changing the deployment algorithm. Default coefficients in our reported setting are $\lambda_{\text{exec}} = 1.0$, $\lambda_{\text{val}} = 0.3$, and $\eta = 0.1$.

The term $C(\tau)$ introduces a reflection cost penalty. If the trajectory invokes the reflection step z_i, $C(\tau)$ is proportional to the token length of z_i; otherwise, for a one-shot success, $C(\tau) = 0$. The penalty coefficient η forces the agent to reflect *only* when strictly necessary, ensuring that the expected reward gain from a successful recovery strictly outweighs the penalty of generating additional reasoning tokens.

Optional training-time verifier design. When enabled, the verifier value $V(x_i, \tau)$ is computed *deterministically* from each candidate and the active record's RoT metadata; **no GPT-4o (or any LLM) is queried inside the RL loop**. We derive three subscores in $[0, 1]$ by signature matching and a grounded-reflection check rather than by model scoring:

- Plan validity s_p: $s_p = 1$ iff the candidate's normalized primary tool and first argument match the stored corrected-action signature, else 0.
- Answer consistency s_a: $s_a = 1$ iff the candidate's replay succeeds (its terminal action agrees with the group's correct answer), else 0.

– Grounding s_g: $s_g = 1$ iff the candidate carries a reflection whose evidence is text-grounded in the stored failure observation *and* $s_p > 0$, else 0. There is intentionally no fallback to s_p for reflection-less candidates, so V rewards grounded reflection above a bare plan repair.

The overall score is the weighted sum $V = w_a s_a + w_g s_g + w_p s_p$, with each subscore clamped to $[0, 1]$ and weights set to emphasize grounding ($w_g \geq w_a, w_p$); we use $(w_a, w_g, w_p) = (0.25, 0.50, 0.25)$ by default. Because V is a function of the candidate and the metadata, it is used only as a training-time reward-shaping signal and never as a deployment-time call. A GPT-4o teacher is used *only offline*, to synthesize the RoT reflections in the data engine (Sec. 3.2); it is never queried during RL or at inference. Full coefficient ranges, the deterministic subscore definitions, and the selection protocol are provided in Appendix A3.

3.4 Training Objective

Structured Reflective Data in Supervised Fine-Tuning (warm start). We first teach the model to diagnose and correct failures with explicit supervision, so Reinforcement Learning (RL) can focus on *how* reflection should be written under explicit trigger gating rather than learning recovery from scratch. Given the failure context $(x_i, a_i^{fail}, o_i^{fail})$, we maximize the likelihood of the structured reflection and corrected action:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(x_i, a_i^{fail}, o_i^{fail}, z_i, a_i^*)} \left[\log P_\theta(z_i, a_i^* \mid x_i, a_i^{fail}, o_i^{fail}) \right]. \tag{3}$$

This stage teaches the model *what* to diagnose and *how* to fix errors, providing a strong initialization for subsequent RL.

Structured Reflective Data in Policy Optimization. SFT on RoT data teaches the model to fix common tool-call errors (e.g., crop shift, wrong region, mismatched tool arguments), but real-world failures are more diverse. Therefore, to generalize beyond curated patterns, the agent must self-explore to discover new failure modes and recovery strategies. This motivates an RL stage to expand coverage and improve robustness.

However, PPO [19] and DPO [17] optimize actions or preference pairs without modeling reflection as a decision variable. In long-horizon tool use, a scalar success/failure signal does not indicate *which* step failed or what correction would fix the trajectory. Moreover, DPO relies on preference data that is difficult to collect for multi-step tool executions, and PPO can be sample-inefficient under sparse rewards. In contrast, GRPO [20] optimizes relative rewards within a group of sampled trajectories, producing lower-variance advantages that are well suited to sparse, delayed rewards. By comparing recoveries against nearby failures, GRPO yields more informative gradients than absolute-reward optimization.

As illustrated in Figure 3, ReGRPO extends GRPO by making reflection z_i part of the optimized trajectory. This ties the advantage signal directly to diagnostic tokens, so the policy learns *which* reflections are useful and *how much* reflection is worth generating. Consequently, reflection quality and brevity become explicitly optimized rather than a fixed prompting heuristic.

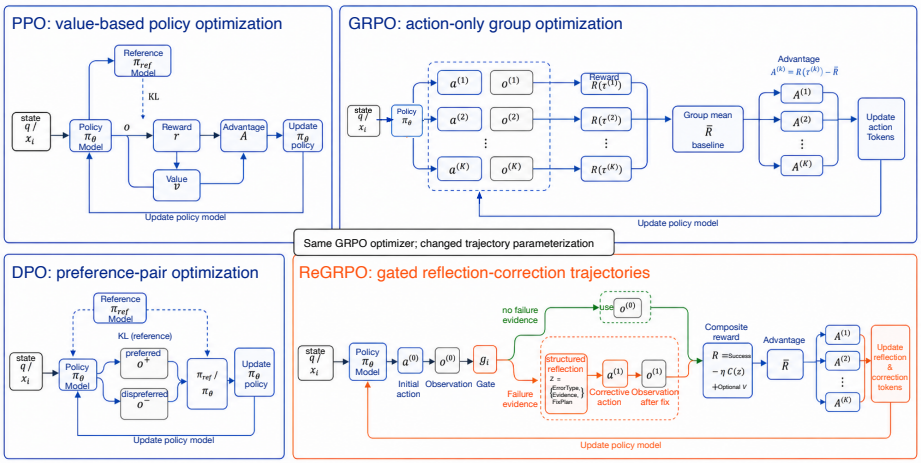

Fig. 3: Comparison of PPO [19], DPO [17], GRPO [20], and ReGRPO (ours). PPO and DPO optimize actions or preferences without treating reflection as a decision variable; GRPO reduces variance via group-relative rewards; ReGRPO further includes reflection in the optimized trajectory to provide stronger recovery-oriented supervision for failed steps.

Reflection-Driven Advantage and Optimization. Within the sampled group, we compute the baseline as the mean reward $\bar{R}_i = \frac{1}{K}\sum_{k=1}^{K} R(\tau_i^{(k)})$, and extract the advantage for each trajectory: $A_i^{(k)} = R(\tau_i^{(k)}) - \bar{R}_i$.

For two trajectories from the same state context, an unrecovered failure τ^- and a recovered trajectory τ^+, the reward gap is $\Delta R = R(\tau^+) - R(\tau^-)$. When recovery succeeds and reflection remains concise, ΔR is typically positive, so $A(\tau^+) > A(\tau^-)$ within the same sampled group. Importantly, this remains true in the verifier-free setting ($\lambda_{\text{val}} = 0$), where improvements come entirely from reflection-conditioned recovery and reflection-cost control.

The ReGRPO objective is optimized as:

$$\mathcal{L}_{\text{ReGRPO}} = -\mathbb{E}_{x_i,\tau\sim\pi_\theta}\left[\frac{1}{K}\sum_{k=1}^{K} A_i^{(k)} \log \pi_\theta(\tau_i^{(k)} \mid x_i)\right] + \beta\, \mathbb{D}_{\text{KL}}\big(\pi_\theta \| \pi_{\text{ref}}\big)$$

Since the generation probability of structured reflection $\log \pi_\theta(z_i \mid x_i, a_i^{(0)}, o_i^{(0)})$ is factorized within the trajectory likelihood $\log \pi_\theta(\tau_i^{(k)} \mid x_i)$, relative advantages directly scale gradients on reflection tokens. ReGRPO therefore uses the standard GRPO optimizer and advantage estimator; the contribution is the structured reflection trajectory design and training/deployment protocol, not a new value estimator. Detailed quantitative results are presented in Sec. 4, and extended diagnostics are reported in the appendix.

3.5 Zero-Verifier Inference Stage

At deployment, we use single-path inference without external verifier calls. A deterministic gate decides whether the policy should open one local reflection-correction block. Tool outputs are first normalized into a canonical schema

$$\hat{o}_i = \{\texttt{status}, \texttt{payload}, \texttt{meta}\}, \tag{4}$$

where `status` stores backend error codes and `payload` stores normalized content. We then use a minimal trigger

$$g_i = \mathbf{1}\{\texttt{ToolError}(\hat{o}_i) \vee \texttt{EmptyObs}(\hat{o}_i) \vee u_i < \kappa_i\}, \tag{5}$$

with policy confidence

$$u_i = \exp\left(\frac{1}{|a_i^{(0)}|}\sum_j \log \pi_\theta(a_{i,j}^{(0)} \mid x_i, h_i)\right). \tag{6}$$

The first two terms capture explicit runtime failures. The confidence term is a lightweight uncertainty proxy: it measures token-level confidence for the predicted action sequence. We therefore use it only as a trigger heuristic for potential silent failures. To avoid per-tool offline calibration, we use an online adaptive threshold computed from the current trajectory:

$$\kappa_i = \frac{1}{\max(1, i-1)}\sum_{j=1}^{i-1} u_j. \tag{7}$$

For $i = 1$, we disable the confidence trigger and rely on hard-failure checks only. If $g_i = 0$, inference continues with the next standard action. If $g_i = 1$, the policy executes one local block $a_i^{(0)} \to o_i^{(0)} \to z_i \to a_i^{(1)} \to o_i^{(1)}$, with at most one reflection-correction block per step. This design keeps inference deterministic and lightweight, while reducing manual feature engineering and per-tool tuning.

Pseudo-code for this trigger is provided in the Appendix.

4 Experiments

4.1 Benchmarks and Metrics

We evaluate ReGRPO on two multimodal tool-use benchmarks in the MAT-AGENT setting [5], where agents must reason over images and documents with real tools.

GTA Dataset. GTA [25] contains 229 tasks paired with 252 images. Each task requires 2–8 tool-use steps (typically 2–4) and tests visual perception, local operations (e.g., reading receipts or charts), and short reasoning chains over screenshots and UI-like images.

GAIA Dataset. GAIA [16] is a document-centric benchmark with 446 tasks over 109 files in PPTX, PDF, and XLSX formats. Tasks are grouped into three difficulty levels and

Table 1: Main comparison on GTA and GAIA under a unified single-path, zero-verifier inference protocol. The default verifier-free ReGRPO ($\lambda_{\text{val}} = 0$) achieves the best performance among the evaluated open-source controllers on both benchmarks.

Method	**Controller**	**GTA**			**GAIA**			
		ToolAcc	*CodeExec*	*AnsAcc*	*Level 1*	*Level 2*	*Level 3*	*AnsAcc*
Closed-source Controller								
Lego Agent	GPT-4	-	-	46.59	-	-	-	-
Lego Agent	GPT-4o	-	-	41.52	-	-	-	-
Warm-up Agent	GPT-4-turbo	-	-	-	30.20	15.10	0.00	17.60
HF Agent	GPT-4o	63.41	95.12	57.05	47.17	31.40	11.54	33.40
HF Agent	GPT-4o-mini	56.10	100.00	57.69	33.96	27.91	3.84	26.06
Open-Source Controller								
HF Agent	LLaVA-NeXT-8B	14.97	25.08	14.10	9.43	1.16	0.00	3.64
HF Agent	InternVL2-8B	36.75	52.18	32.05	7.55	4.65	0.00	4.85
HF Agent	MiniCPM-V-8.5B	36.59	56.10	33.97	13.21	5.81	0.00	7.27
HF Agent	Qwen2-VL-7B	44.85	65.19	42.31	16.98	8.14	0.00	9.70
T3-Agent	MAT-MiniCPM-V-8.5B	65.85	80.49	52.56	26.42	11.63	3.84	15.15
T3-Agent	MAT-Qwen2-VL-7B	64.63	84.32	53.85	26.42	15.12	3.84	16.97
SPORT Agent	Tuned-Qwen2-VL-7B	72.41	91.87	60.26	35.85	16.28	3.84	20.61
Ours								
ReGRPO (default, $\lambda_{\text{val}} = 0$)	MAT-Qwen2-VL-7B	**76.35**	**93.77**	**67.66**	**39.02**	**18.71**	**4.89**	**23.35**

often require multiple tool calls for document understanding, web navigation, logical reasoning, and summarization.

Metrics. We report standard accuracy metrics: *AnsAcc* (answer accuracy), *ToolAcc* (tool-call validity), and *CodeExec* (execution success rate). Unless noted otherwise, all main-text comparisons use zero-verifier deployment, with no external verifier calls at test time.

4.2 Baselines

We compare ReGRPO with both closed-source and open-source controllers.

Closed-source agents. We report GPT-4/4o-based results from prior work, including Lego Agent and Warm-up Agent [16, 25], and HF Agents powered by GPT-4o and GPT-4o-mini. These models provide strong proprietary references.

Open-source agents. We include HF Agents based on LLaVA-NeXT-8B, InternVL2-8B, MiniCPM-V-8.5B, and Qwen2-VL-7B. We also evaluate MAT-AGENT (T3-Agent) with MAT-MiniCPM-V-8.5B and MAT-Qwen2-VL-7B, and SPORT Agent with a tuned Qwen2-VL-7B controller. For fair internal comparisons, all ablations use the same Qwen2-VL-7B controller and tool suite.

Our model. We use the same Qwen2-VL-7B backbone and toolset as SPORT [9] to isolate the effect of our method. We compare controlled variants under matched training settings (same total updates, rollout budget, and token budget): (1) MAT-AGENT: Fine-tuned on MM-Traj without RL; (2) +RoT SFT: SFT with structured reflective data only; (3) +Optional Verifier Distill: adds verifier-aware distillation but no RL; (4) GRPO-only: GRPO on actions only (no reflection tokens); (5) GRPO + Free-form Reflection: allows unstructured reflection text without schema constraints; (6) ReGRPO core (default, verifier-free RL): structured reflection with $\lambda_{\text{val}} = 0$ during RL; (7) ReGRPO + optional verifier reward: additive shaping with the deterministic, metadata-derived verifier value ($\lambda_{\text{val}} > 0$). All variants use Single-Path, Zero-Verifier Inference at test time.

Table 2: Ablation of ReGRPO under the same backbone and tool settings. The verifier-free core pipeline (RoT + ReGRPO, $\lambda_{\text{val}} = 0$) reaches $67.66/23.35$ GTA/GAIA AnsAcc, and adding the optional teacher-derived verifier reward further improves to $68.49/24.01$. Most gains come from the RoT SFT + structured RL combination, while verifier signals act as optional additive improvements.

Method	Module			GTA			GAIA			
	SFT Data	RL Alg.	Reflect.	*ToolAcc*	*CodeExec*	*AnsAcc*	*Level 1*	*Level 2*	*Level 3*	*AnsAcc*
MAT-AGENT [5]	MM-Traj.	-	-	64.63	84.32	53.85	26.42	15.12	3.84	16.97
+RoT (SFT)	RoT	-	✓	68.73	87.92	58.59	30.58	15.61	4.17	19.03
+Optional Verifier Distill (no RL)	RoT	-	✓	69.84	88.41	59.72	31.04	15.94	4.17	19.84
GRPO-only (w/o RoT)	MM-Traj.	GRPO	-	71.34	90.23	64.51	36.79	18.12	3.84	18.92
GRPO + Free-form Reflection	RoT	GRPO	✓	73.05	91.02	65.34	37.26	18.02	4.17	21.38
ReGRPO core (default, $\lambda_{\text{val}} = 0$)	RoT	ReGRPO	✓	76.35	93.77	67.66	39.02	18.71	4.89	23.35
+ optional verifier reward (deterministic)	RoT	ReGRPO	✓	77.26	94.91	68.49	40.09	19.67	5.32	24.01

4.3 Implementation Details

Model and optimization. We use Qwen2-VL-7B as the controller. The vision encoder and visual token compressor are frozen. We fine-tune the language model with LoRA [7] (rank 32), applied to query, key, and value projections in all self-attention layers. We optimize with AdamW and cosine learning-rate decay, using a base learning rate of 1.0×10^{-6} and batch size 2 per device.

Training stages. Our default pipeline has two mandatory stages: (1) *SFT warm start* on our Structured Reflective Data (triplets); (2) *ReGRPO process RL* on a mixture of offline groups and online self-exploration groups. Optionally, we insert verifier-aware distillation between (1) and (2), where the controller predicts the deterministic, metadata-derived verifier subscores for better calibration and more stable initialization. ReGRPO uses the reflection-aware reward in Eq. 2, with a reflection-cost penalty $\eta = 0.1$. By default, we set $\lambda_{\text{val}} = 0$ (verifier-free RL); runs with $\lambda_{\text{val}} > 0$ using the deterministic, metadata-derived verifier value are reported as additive variants. For controlled comparisons, variants with and without distillation use identical optimizer settings and matched update steps to avoid gains from extra training budget.

4.4 Quantitative Results

Table 1 summarizes results on GTA and GAIA [16, 25]. Under the same single-path, zero-verifier setup, default ReGRPO ($\lambda_{\text{val}} = 0$) gives the strongest results among the compared open-source controllers. On GTA, it reaches 76.35 ToolAcc and 67.66 AnsAcc, improving over SPORT [9] by $+3.94$ ToolAcc and $+7.40$ AnsAcc. The larger AnsAcc gain suggests that reflection helps end-to-end reasoning beyond tool-call validity alone. On GAIA, default ReGRPO raises overall AnsAcc to 23.35 ($+2.74$ over SPORT), indicating stronger results on document-centric multi-step tasks even without verifier reward. Adding the optional teacher-derived verifier reward further improves performance to $68.49/24.01$ GTA/GAIA AnsAcc in Table 2, while the main gains are already achieved in verifier-free training and deployment. Across both datasets, the results support our hypothesis that structured reflections improve tool-grounded reasoning. Multi-seed statistics and extended analyses are provided in the appendix.

(a) MAT SFT vs. RoT training example (verbatim record `0FLZe2lb_rot_s0_GroundingDrift`)

MAT SFT has top-level fields `id`, `image`, `answer`, `conversations`. **RoT** keeps `id`/`image`/`conversations` and adds a `reflection` triplet (`error_type`, `evidence`, `fix_plan`) plus a `corrected_action`.

MAT instance (abbreviated)

```
{"id":"0FLZe2lb", "image":"coco/.../000000043093.jpg",
"answer":"...smoothie calorie estimate...",
"conversations":[{"role":"user",...},{"role":"assistant",...}]}
```

RoT instance (abbreviated) (blue = fields RoT adds over MAT)

```
{"id":"0FLZe2lb_rot_s0_GroundingDrift",
"image":"coco/.../000000043093.jpg",
  "failed_action":"visualizer(q='fruits/utensils on the table')",
  "reflection":{"error_type":"GroundingDrift",
      "evidence":"cutting board, banana, knife",
      "fix_plan":"re-ground to the smoothie glass"},
  "corrected_action":"visualizer(q='ingredients in this smoothie')"}
```

(b) Mechanism-aligned reflection (real `(ErrorType, Evidence, FixPlan)` triplet)

Task. "How many calories are in the smoothie shown in this image?" The image shows a smoothie glass beside a cutting board with a sliced banana, a knife, and granola.

Silent failure (image grounding). The perturbed action asks `visualizer` about the *fruits and utensils on the table*; the tool returns a valid, non-empty description ("a wooden cutting board with a sliced banana, a knife, ...; the smoothie glass is not described"), so no explicit tool error is raised. Low policy confidence ($u_i < \kappa_i$) opens one local repair block.

Reflection (ReGRPO schema, verbatim). `ErrorType=GroundingDrift; Evidence="wooden cutting board with a sliced banana, a knife"; FixPlan="refocus from the table/cutting board to the smoothie glass and ask about its contents".` The corrected action re-grounds the query to the smoothie itself.

Takeaway. This real record exposes the full Error→Reflection→Correction chain that ReGRPO optimizes, under the same single-path zero-verifier deployment.

(c) Inference Path Comparison

Sample content

```
tools: visualizer
files: coco/train2017/000000043093.jpg
dialogs[0]: {"role":"user","content":"How many calories are in the
smoothie shown in this image?"}
```

Note: abbreviated from the verbatim RoT record; tool outputs are shortened for readability.

Baseline path (no local repair block)

1. **Step 1**: `visualizer(image, q="fruits/utensils on the table")` → "cutting board, sliced banana, knife, granola; smoothie glass not described".
2. **Step 2**: estimate calories from the off-target items on the board → ungrounded ingredient list (wrong basis).

ReGRPO path (single-path, zero-verifier)

1. **Step 1**: same first call; the description omits the target smoothie ($g_i = 0$ so far).
2. **Step 1 (gate)**: the target object is missing from the observation, so confidence is low ($u_i < \kappa_i$); set $g_i = 1$.
3. **Local block**: $a_i^{(0)} \to o_i^{(0)} \to z_i \to a_i^{(1)} \to o_i^{(1)}$. Reflection emits `ErrorType=GroundingDrift`, `Evidence=cutting board / banana / knife`, and `FixPlan=re-ground to the smoothie glass`.
4. **Step 2**: corrected action `visualizer(image, q="ingredients in this smoothie")` → grounded smoothie contents, then a grounded calorie estimate.

Fig. 4: Figure-level evidence for ReGRPO, instantiated on a verbatim synthesized RoT record (`0FLZe2lb_rot_s0_GroundingDrift`). (a) RoT augments the MAT SFT format with explicit reflective fields (a `reflection` triplet and a `corrected_action`). (b) The real `(ErrorType, Evidence, FixPlan)` reflection diagnoses a silent grounding failure—the tool answers about the cutting board rather than the smoothie—and prescribes a re-grounding fix. (c) Inference-path comparison contrasts a brittle baseline route that estimates calories from off-target items with a ReGRPO route where one confidence trigger opens a single local reflection–correction block that re-grounds to the smoothie.

4.5 Ablation Studies

Table 2 reports a controlled ablation in which we progressively add reflective supervision and reflection-aware optimization.

Contribution coverage. Our experimental design covers these method components: (1) the *Structured Reflective Data Engine* is evaluated by MAT-AGENT $\rightarrow$ +RoT SFT and the data-format comparison in Figure 4; (2) the *GRPO-based reflection protocol* is demonstrated by comparing GRPO-only and free-form reflection against structured ReGRPO under the same settings; (3) *deterministic zero-verifier trigger gating* is assessed under single-path, zero-verifier deployment, with extended trigger diagnostics in the appendix.

Baseline and reflective SFT gains. Starting from MAT-AGENT (MM-Traj, no RL), performance is 53.85 GTA AnsAcc and 16.97 GAIA AnsAcc. Replacing SFT data with RoT (+RoT SFT) raises GTA AnsAcc to 58.59 and GAIA AnsAcc to 19.03 ($+4.74$ GTA / $+2.06$ GAIA), showing that explicit Error-Reflection-Correction supervision improves trajectory quality before RL.

Optional verifier distillation contributes modestly. Adding optional verifier distillation on top of RoT SFT further improves performance to 59.72 GTA AnsAcc and 19.84 GAIA AnsAcc (an additional $+1.13$ GTA / $+0.81$ GAIA over +RoT SFT). This suggests the deterministic verifier subscores help intermediate decisions, but alone they do not close the gap to RL-based methods.

Action-only RL vs. reflection-aware variants. GRPO-only (without RoT reflection tokens in policy optimization) reaches 64.51 GTA AnsAcc and 18.92 GAIA AnsAcc, indicating that RL exploration is effective on GTA but more limited on GAIA. Allowing free-form reflection improves results to 65.34 GTA AnsAcc and 21.38 GAIA AnsAcc ($+0.83$ GTA / $+2.46$ GAIA over GRPO-only), suggesting reflection text helps, while unstructured reflection remains less efficient and less consistent.

Effect of structured ReGRPO and optional verifier reward. Switching from free-form reflection to structured ReGRPO with $\lambda_{\text{val}} = 0$ yields 67.66 GTA AnsAcc and 23.35 GAIA AnsAcc ($+2.32$ GTA / $+1.97$ GAIA), showing that schema-constrained reflection and reflection-correction coupling add gains even without verifier reward in RL. Adding the optional teacher-derived verifier reward further improves to 68.49 GTA AnsAcc and 24.01 GAIA AnsAcc ($+0.83$ GTA / $+0.66$ GAIA over ReGRPO core), indicating that verifier reward is useful but not essential.

Key insight. The ablation shows complementary effects: (1) RoT data provides a strong starting point, (2) structured reflection-aware RL adds consistent gains on top of that initialization, and (3) verifier signals provide modest additive shaping rather than core capability. ReGRPO therefore remains effective in a verifier-free default setting during both training ($\lambda_{\text{val}} = 0$) and inference (zero-verifier single-path execution). We further verify that ReGRPO preserves the base model's VQA ability (Appendix Sec. A6).

4.6 Qualitative Analysis

Figure 4 presents three views of the mechanism, instantiated on a verbatim synthesized RoT record (a `GroundingDrift` case). Panel (a) shows that RoT extends the MAT SFT format with explicit reflective fields (a `reflection` triplet and a `corrected_action`), so supervision covers failure diagnosis and correction rather than tool calls alone. Panel (b) gives case-level evidence aligned with Sec. 3.5: the tool returns a valid but off-target description (it answers about the cutting board, not the smoothie), so no explicit error is raised, and low confidence ($u_i < \kappa_i$) triggers a structured reflection whose schema (ErrorType, Evidence, FixPlan) re-grounds the query rather than adding generic free-form text. Panel (c) shows the same mechanism as paths: the baseline route estimates from off-target items, while ReGRPO opens exactly one local block at the uncertain step and rewrites the action to re-ground on the target object. The full record is given in Appendix Table 12.

5 Conclusion

We presented **ReGRPO**, a reflection-augmented framework for training tool-using vision–language agents. ReGRPO builds a Structured Reflective Data Engine and applies a GRPO-based reflection protocol to optimize reflection and action generation jointly in tool-using trajectories. Our method keeps the standard GRPO optimizer, with the main design changes in structured reflection representation and the training/deployment protocol.

On the GTA and GAIA benchmarks, our approach consistently outperforms action-level and SFT-only baselines under the same backbone/tool setting. The ablation results show complementary gains: RoT SFT provides a strong foundation, structured RL contributes further improvements, and optional verifier distillation/reward adds further but smaller gains. Overall, ReGRPO improves answer accuracy and tool correctness over MAT-AGENT and SPORT while using the same VLM backbone and tool suite.

Acknowledgements. This project is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award No: MOE-T2EP20124-0012).

References

1. Chen, X., Lin, M., Schärli, N., Zhou, D.: Teaching large language models to self-debug (2023), https://arxiv.org/abs/2304.05128
2. Choi, W., Kim, W.K., Yoo, M., Woo, H.: Embodied cot distillation from llm to off-the-shelf agents. In: ICML. pp. 8702–8721 (2024)
3. Fan, Y., Ma, X., Wu, R., Du, Y., Li, J., Gao, Z., Li, Q.: Videoagent: A memory-augmented multimodal agent for video understanding. In: ECCV (2024)
4. Gao, Z., Du, Y., Zhang, X., Ma, X., Han, W., Zhu, S.C., Li, Q.: Clova: A closed-loop visual assistant with tool usage and update. In: CVPR. pp. 13258–13268 (2024)
5. Gao, Z., Zhang, B., Li, P., Ma, X., Yuan, T., Fan, Y., Wu, Y., Jia, Y., Zhu, S.C., Li, Q.: Multi-modal agent tuning: Building a vlm-driven agent for efficient tool usage. arXiv preprint arXiv:2412.15606 (2024)
6. Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al.: Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948 (2025)
7. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: Lora: Low-rank adaptation of large language models. In: ICLR (2022)
8. Li, F., Zhang, R., Zhang, H., Zhang, Y., Li, B., Li, W., Ma, Z., Li, C.: Llava-next-interleave: Tackling multi-image, video, and 3d in large multimodal models. arXiv preprint arXiv:2407.07895 (2024)
9. Li, P., Gao, Z., Zhang, B., Mi, Y., Ma, X., Shi, C., Yuan, T., Wu, Y., Jia, Y., Zhu, S.C., et al.: Iterative tool usage exploration for multimodal agents via step-wise preference tuning. arXiv preprint arXiv:2504.21561 (2025)
10. Liao, Y.H., Mahmood, R., Fidler, S., Acuna, D.: Can feedback enhance semantic grounding in large vision-language models. arXiv preprint arXiv:2404.06510 **3** (2024)
11. Liu, H., Li, C., Li, Y., Li, B., Zhang, Y., Shen, S., Lee, Y.J.: Llava-next: Improved reasoning, ocr, and world knowledge (January 2024), https://llava-vl.github.io/blog/2024-01-30-llava-next/
12. Liu, S., Cheng, H., Liu, H., Zhang, H., Li, F., Ren, T., Zou, X., Yang, J., Su, H., Zhu, J., et al.: Llava-plus: Learning to use tools for creating multimodal agents. arXiv preprint arXiv:2311.05437 (2023)
13. Liu, X., Zhang, T., Gu, Y., Iong, I.L., Xu, Y., Song, X., Zhang, S., Lai, H., Liu, X., Zhao, H., et al.: Visualagentbench: Towards large multimodal models as visual foundation agents. arXiv preprint arXiv:2408.06327 (2024)
14. Lu, H., Liu, W., Zhang, B., Wang, B., Dong, K., Liu, B., Sun, J., Ren, T., Li, Z., Yang, H., et al.: Deepseek-vl: towards real-world vision-language understanding. arXiv preprint arXiv:2403.05525 (2024)
15. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B.P., Hermann, K., Welleck, S., Yazdanbakhsh, A., Clark, P.: Self-refine: Iterative refinement with self-feedback (2023), https://arxiv.org/abs/2303.17651
16. Mialon, G., Fourrier, C., Swift, C., Wolf, T., LeCun, Y., Scialom, T.: Gaia: a benchmark for general ai assistants. arXiv preprint arXiv:2311.12983 (2023)
17. Rafailov, R., Sharma, A., Mitchell, E., Manning, C.D., Ermon, S., Finn, C.: Direct preference optimization: Your language model is secretly a reward model. Advances in neural information processing systems **36**, 53728–53741 (2023)
18. Sasazawa, Y., Sogawa, Y.: Layout generation agents with large language models. arXiv preprint arXiv:2405.08037 (2024)

19. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
20. Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al.: Deepseekmath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300 (2024)
21. Shen, Y., Song, K., Tan, X., Zhang, W., Ren, K., Yuan, S., Lu, W., Li, D., Zhuang, Y.: Taskbench: Benchmarking large language models for task automation. arXiv preprint arXiv:2311.18760 (2023)
22. Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., Yao, S.: Reflexion: Language agents with verbal reinforcement learning (2023), https://arxiv.org/abs/2303.11366
23. Surís, D., Menon, S., Vondrick, C.: Vipergpt: Visual inference via python execution for reasoning. In: ICCV. pp. 11888–11898 (2023)
24. Wang, C., Luo, W., Chen, Q., Mai, H., Guo, J., Dong, S., Xuan, X., Li, Z., Ma, L., Gao, S.: Mllm-tool: A multimodal large language model for tool agent learning. arXiv preprint arXiv:2401.10727 **4** (2024)
25. Wang, J., Ma, Z., Li, Y., Zhang, S., Chen, C., Chen, K., Le, X.: Gta: A benchmark for general tool agents. In: NeurIPS (2024)
26. Wang, Z., Li, A., Li, Z., Liu, X.: Genartist: Multimodal llm as an agent for unified image generation and editing. arXiv preprint arXiv:2407.05600 (2024)
27. Wang, Z., Xie, E., Li, A., Wang, Z., Liu, X., Li, Z.: Divide and conquer: Language models can plan and self-correct for compositional text-to-image generation. arXiv preprint arXiv:2401.15688 (2024)
28. Xiong, T., Wang, X., Guo, D., Ye, Q., Fan, H., Gu, Q., Huang, H., Li, C.: Llava-critic: Learning to evaluate multimodal models. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 13618–13628 (2025)
29. Yin, D., Brahman, F., Ravichander, A., Chandu, K., Chang, K.W., Choi, Y., Lin, B.Y.: Agent lumos: Unified and modular training for open-source language agents. In: ICML. pp. 12380–12403 (2024)
30. Zhang, J., Lan, T., Murthy, R., Liu, Z., Yao, W., Tan, J., Hoang, T., Yang, L., Feng, Y., Liu, Z., et al.: Agentohana: Design unified data and training pipeline for effective agent learning. arXiv preprint arXiv:2402.15506 (2024)
31. Zheng, B., Gou, B., Kil, J., Sun, H., Su, Y.: Gpt-4v (ision) is a generalist web agent, if grounded. In: ICML. pp. 61349–61385 (2024)