MATRIX: Multimodal Agent Tuning for Robust Tool-Use Reasoning

Anonymous authors
Paper under double-blind review

Figure 1: Comparison of baseline Qwen2-VL (Yang et al., 2024), MAT (Gao et al., 2025b), and proposed MATRIX agent on a visual reasoning task. MATRIX shows superior tool use, fewer hallucinations, and more consistent reasoning, while Qwen2-VL and MAT often struggle with tool coordination and fallback strategies.

Abstract

Vision language models (`VLMs`) are increasingly deployed as controllers with access to external tools for complex reasoning and decision-making, yet their effectiveness remains limited by the scarcity of high-quality multimodal trajectories and the cost of manual annotation. We address this challenge with a vision-centric agent tuning framework that automatically synthesizes multimodal trajectories, generates step-wise preference pairs, and trains a `VLM` controller for robust tool-use reasoning. Our pipeline first constructs `M-TRACE`, a large-scale dataset of 28.5K multimodal tasks with 177K verified trajectories, enabling imitation-based trajectory tuning. Building on this, we develop `MATRIX` Agent, a controller finetuned on `M-TRACE` for step-wise tool reasoning. To achieve finer alignment, we further introduce `Pref-X`, a set of 11K automatically generated preference pairs, and optimize `MATRIX` on it via step-wise preference learning. Across three benchmarks, Agent-X, GTA, and GAIA, `MATRIX` consistently surpasses both open- and closed-source `VLMs`, demonstrating scalable and effective multimodal tool use. Our datasets and models will be open-sourced to support future research.

1 Introduction

Vision language models (`VLMs`) augmented with external tools are increasingly used as controllers for complex reasoning and decision-making tasks (Gao et al., 2024; Surís et al., 2023; Gupta & Kembhavi, 2023; Yuan et al., 2024). Acting as central planners, they invoke diverse tools through structured prompts, enabling applications in visual editing (Wu et al., 2023), embodied control (ichter et al., 2023), question answering (Shen et al., 2023), video reasoning (Fan et al., 2024), and desktop automation (Trivedi et al., 2024). Existing

approaches improve tool use by fine-tuning controllers on trajectories collected via manual annotation or closed-source APIs. However, such data are costly to obtain and often biased toward narrow environments, which limits their generalization to broader multimodal tasks.

Limitations of existing approaches. Most existing agents are trained with supervised fine-tuning (`SFT`) on curated tool-use demonstrations (Peng et al., 2023; Wang et al., 2024d; Sun et al., 2024). Such datasets are expensive to collect, difficult to scale, and biased toward narrow domains or tool distributions. Reliance on static demonstrations further constrains generalization: if the examples emphasize only certain tools (e.g., `video_parser`, `image_qa`), agents often fail when confronted with tasks requiring different capabilities, such as live web search or object grounding. As a result, models overfit to specific usage patterns and struggle in unseen multimodal contexts. Recent works (Gao et al., 2025b; Li et al., 2025) explore synthetic generation to reduce annotation cost, but key challenges remain. Generated trajectories are inconsistent in quality, exploration is limited, and evaluations are restricted to narrow domains. Moreover, imitation learning alone cannot teach agents to refine tool usage or recover from partially correct rollouts, since it lacks reinforcement-based feedback.

Our insight: robust agents need both traces and preferences. The central challenge in training tool-using agents lies in bridging two gaps: (i) acquiring fundamental tool-usage skills from scarce, high-quality trajectories, and (ii) refining these skills to handle the ambiguities, errors, and partial successes that naturally occur in open-ended reasoning. Existing methods either focus solely on imitation, which cannot teach recovery or refinement, or rely on trajectory-level labels, which are too coarse to capture step-wise decision quality. Our key insight is that these challenges require a staged approach. *First,* large-scale supervised traces are essential to ground a `VLM` controller in multimodal reasoning and tool invocation. *Second,* once the model can follow trajectories, its limitations are best addressed through step-level preference optimization, where alternative actions are explored, compared, and refined. This pairing of supervised grounding with preference-based alignment enables agents not only to execute valid tool calls but also to select the most effective ones in complex reasoning chains.

Our framework. We introduce `MATRIX`, a two-stage framework that first equips a controller with supervised tool-use skills and then refines its decision-making through preference optimization. (1) *Trajectory-driven SFT:* We construct `M-TRACE`, a dataset of 28.5K multimodal tasks with 177K verified step-level tool-use trajectories, providing broad coverage of tool reasoning skills. (2) *Preference optimization:* We build `Pref-X`, 11K automatically generated preference pairs from step-level exploration and verification, and apply Direct Preference Optimization (DPO) (Kong et al., 2025) to align the controller with fine-grained tool-use preferences. This staged design grounds the agent in verified traces while enabling progressive improvement through self-exploration and automatic verification. We evaluate `MATRIX` on three challenging benchmarks, Agent-X (Ashraf et al., 2025), GTA (Wang et al., 2024b), and GAIA (Mialon et al., 2023), where it improves answer accuracy by 14%, 23%, and 11%, respectively. As shown in Fig. 1, `MATRIX` achieves consistent reasoning and more adaptive tool selection compared to prior agents. Our main contributions are as follows:

1. `M-TRACE`: a large-scale corpus of 28.5K multimodal tasks and 177K verified tool-use trajectories built via automated synthesis and verification.
2. `Pref-X`: 11K preference-labeled step pairs that enable fine-grained alignment of tool-usage decisions beyond imitation learning.
3. `MATRIX`: a robust, vision-centric agent that leverages trajectory supervision with preference optimization for efficient tool-use reasoning.
4. We show consistent improvements over strong baselines on Agent-X, GTA, and GAIA. Together, these contributions establish `MATRIX` as a scalable and effective agent for training robust multimodal agents.

2 Related Work

Multimodal Agents. The rapid progress of large multimodal models (`LMMs`) (Achiam et al., 2023; Grattafiori et al., 2024; Team et al., 2023; Bi et al., 2024; Bai et al., 2023) has enabled agents that integrate perception, reasoning, and external tools. Moving beyond

text generation, modern agents act as central planners by invoking APIs (Zhang et al., 2025), operating systems (Mei et al., 2024), document analyzers (Musumeci et al., 2024), or web environments (Song et al., 2024), supporting broad interaction with digital ecosystems. This has fueled the rise of orchestration frameworks such as `Avatar` (Wu et al., 2024), `LangChain` (Chase, 2022), and `AutoGPT` (Gravitas, 2023), which couple reasoning with tool execution. Specialized systems extend these capabilities to web browsing (Yao et al., 2022; Nakano et al., 2021; Qin et al., 2023), REST APIs (Song et al., 2023), or multi-model collaboration (Shen et al., 2024; Li et al., 2023). Multimodal variants such as `MLLMTool` (Wang et al., 2025) combine vision and language for perception-driven reasoning. Despite these advances, most frameworks lack systematic training and evaluation protocols for sequential tool reasoning, limiting their robustness in open-ended multimodal environments.

Tool Usage Datasets. Datasets for tool-using agents have primarily targeted text-based settings (Tang et al., 2023; Qin et al., 2024; Du et al., 2024; Liu et al., 2024b). Multimodal benchmarks including Agent-X (Ashraf et al., 2025), `OSWorld` (Xie et al., 2024), `MMInA` (Zhang et al., 2024b), `GAIA` (Mialon et al., 2023), and `GTA` (Wang et al., 2024b) broaden coverage but still rely heavily on curated or repurposed trajectories. Existing tuning methods fall into two paradigms: supervised fine-tuning (`SFT`) on annotated tool-use traces (Shen et al., 2023; Liu et al., 2024b), which is costly and brittle, or reinforcement learning (RL) with synthetic rewards or preferences (Lee et al., 2024; Fu et al., 2024; Yu et al., 2024), which assume reliable reward signals. Recent work explores step-wise preferences (Lai et al., 2024; Wang et al., 2024a; Chen et al., 2024a; Kong et al., 2025), but applications remain narrow (e.g., code or math) where ground-truth labels exist. In contrast, `MATRIX` introduces scalable *step-wise preference optimization with AI-based verification*, where the agent generates, evaluates, and improves its own trajectories. This reduces reliance on manual annotation and enables robust multimodal tool-use reasoning in diverse environments.

3 MATRIX AGENT

`MATRIX` is a vision-centric multimodal agent built to perform reliable step-wise reasoning and tool use. The key challenge for such agents lies in the scarcity of high-quality trajectories and the cost of manual annotations, which limit scalability and generalization. To overcome this, we design a two-stage training framework that leverages trajectory supervision with preference optimization. In the first stage, supervised fine-tuning (`SFT`) on automatically synthesized multimodal trajectories (`M-TRACE`) equips the controller with structured tool-use skills. In the second stage, preference optimization via Direct Preference Optimization (`DPO`) (Kong et al., 2025) on step-level exploration data (`Pref-X`) refines decision-making beyond imitation, encouraging the agent to favor accurate, consistent, and goal-directed actions. The overall framework is illustrated in Fig. 2.

3.1 `M-TRACE` FORMULATION

Pipeline Overview. Our `M-TRACE` synthesis pipeline (Fig. 2) consists of four stages: (1) *query generation*, (2) *artifact construction*, (3) *trajectory collection*, and (4) *parallel verification*. To ensure reliability, we include two parallel verifiers: a *query–artifact verifier* that checks task feasibility and input alignment, and a *trajectory verifier* that validates tool-use consistency.

Data Format. Each multimodal tool-usage instance is represented as:

$$\mathcal{D}_{\texttt{M-TRACE}} = \{\mathcal{F}^*, \mathcal{Q}, \{t_i\}_{i=1}^n, \{c_i\}_{i=1}^n, \{o_i\}_{i=1}^n, \mathcal{A}\}, \tag{1}$$

where $\mathcal{F}^*$ denotes the optional multimodal files (e.g., images, videos, PDFs, PPTX), $\mathcal{Q}$ is the query, $\{t_i\}_{i=1}^n$ are the reasoning thoughts (step-level plans), $\{c_i\}_{i=1}^n$ are the generated code snippets (tool calls), $\{o_i\}_{i=1}^n$ are the corresponding observations (tool outputs), n is number of steps, and $\mathcal{A}$ is the final answer. Following prior works (Ashraf et al., 2025; Wang et al., 2024b), we support two categories of queries: (1) *question answering*, where $\mathcal{A}$ is textual, and (2) *image generation*, where $\mathcal{A}$ is a generated image. Each task may involve multiple steps, forming a trajectory $\tau = \{t_1, c_1, o_1, \ldots, t_n, c_n, o_n\}$, which integrates reasoning, tool execution, and observations across n steps to solve the query.

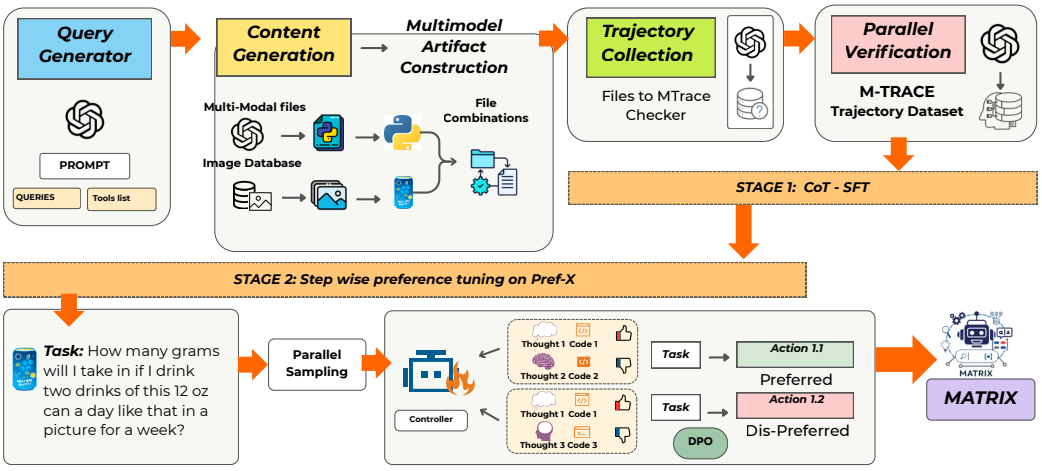


Figure 2: **Overall pipeline.** `M-TRACE` construction, where multimodal queries are paired with verified step-by-step trajectories to create high-quality training traces; and `Pref-X` generation, which produces preference pairs from step-level exploration and verification for preference optimization. `MATRIX` is trained first with supervised fine-tuning on `M-TRACE` and then refined through preference optimization with `DPO` on `Pref-X`.

Data Generation. We followed a four-stage process for generating `M-TRACE`.

1. *Query Generation.* We aim to construct a large pool of diverse, practical, and executable queries. A small set of manually designed seed queries serves as the starting point. Starting from seed queries, we iteratively prompt GPT-4o-mini (Hurst et al., 2024) with tool descriptions and structured templates to generate diverse and executable queries, using hyperparameters (e.g., temperature) to promote diversity.
2. *Artifact Construction.* Unlike prior works that sample files first, we adopt a query-first strategy, ensuring tighter alignment between queries and resources. This is crucial as real tasks often require heterogeneous inputs (e.g., DOCX, PPTX, XLSX, PDF) and multiple resources. For each query, GPT-4o-mini specifies the file type and draft content; images are retrieved via BGE (Chen et al., 2024b) embeddings with similarity search, while non-image files are programmatically generated.
3. *Trajectory Collection.* A zero-shot ReAct-style agent (Yao et al., 2023a) powered by GPT-4o-mini generates multi-step trajectories. Given a query and artifacts, the agent produces step-wise *thoughts*, executable *tool calls*, and corresponding *observations*. Only valid executions are retained, ensuring high-quality reasoning traces.
4. *Parallel Verification.* Two verifiers ensure robustness: (i) the *query–artifact verifier* checks task feasibility and input relevance, and (ii) the *trajectory verifier* validates tool usage, arguments, and outputs. Following prior verification protocols (Liu et al., 2024b; Wang et al., 2023; Gao et al., 2025b), GPT-4o-mini filters noisy or inconsistent samples, discarding ill-posed queries and trajectories.

Data Sources. To diversify visual context, we collect ∼100K image–caption pairs from eight datasets: COCO (Lin et al., 2014), ChartQA (Masry et al., 2022), LLaVA (Liu et al., 2024a), SAM (Kirillov et al., 2023), TextVQA (Singh et al., 2019), WebCelebrity (Liu et al., 2015), Web-Landmark (Weyand et al., 2020), and WikiArt (Saleh & Elgammal, 2015). We further enrich coverage with ShareGPT4V (Zhang et al., 2024a) captions, spanning charts, documents, science QA, visual reasoning, and art.

`M-TRACE` Analysis. After verification, `M-TRACE` yields 28.5K multimodal tasks with 27.5K associated artifacts, distilled from 43.5K initial candidates. These tasks produce 177K verified trajectories, ensuring both scale and quality. The dataset is diverse across several dimensions: (i) *File types:* `M-TRACE` spans over 10 formats—including images, documents, spreadsheets,

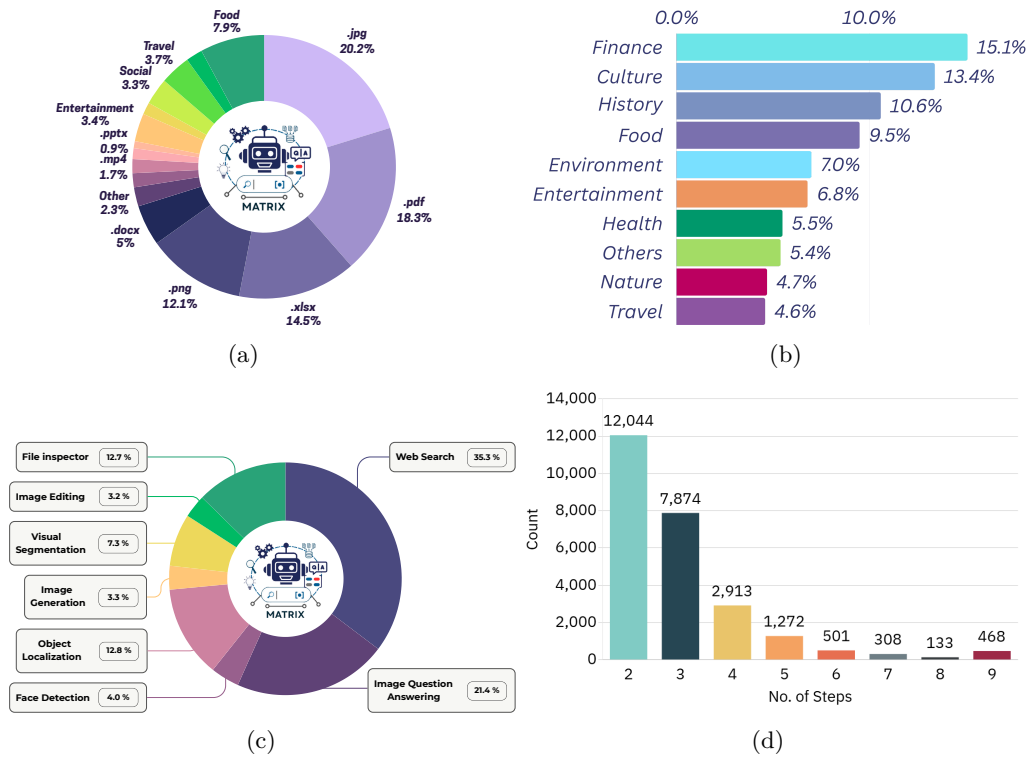


Figure 3: Statistics of M-TRACE. (a) File-type distribution, (b) Domain coverage, (c) Tool usage, (d) Step complexity.

audio, video, and slides—capturing realistic multimodal contexts, with additional coverage of formats like HTML and JSON (Fig. 3a). (ii) *Knowledge domains:* Tasks cover 16 categories such as finance, health, culture, environment, and history, ensuring broad topical coverage (Fig. 3b). (iii) *Tool usage:* Trajectories invoke a wide range of tools, with web search most common, followed by image QA, file inspection, visualization, and Python execution, mirroring real-world problem solving (Fig. 3c). (iv) *Step complexity:* Tasks vary in reasoning depth, with most requiring 2–5 steps and some up to 9, reflecting both practical and complex reasoning cases (Fig. 3d).

3.2 Stage 1: Supervised Fine-Tuning (SFT) with Tool-Use Reasoning

Step-wise reasoning with ReAct. We use Qwen2-VL-7B (Yang et al., 2024) as the controller, an open-source VLM with integrated vision-language grounding. To equip the controller with structured tool-usage skills, we adopt the ReAct paradigm (Yao et al., 2023b), where reasoning unfolds step by step. At each step i, the controller first generates a *thought* t_i (a natural language plan) and then produces a corresponding *code snippet* c_i to invoke a tool. Compared with fixed formats (e.g., JSON), Python-style code provides greater flexibility for diverse input–output types and seamless integration with real tools.

Formally, given a query Q, optional external resources $\mathcal{F}^*$, and history $\mathcal{H}_i = \{t_1, c_1, o_1, \ldots, t_{i-1}, c_{i-1}, o_{i-1}\}$, the controller chooses a tool and arguments by maximizing:

$$t_i^\star, c_i^\star = \arg\max_{t_i, c_i} P_\theta(t_i, c_i \mid Q, \mathcal{F}^*, \mathcal{H}_i), \tag{2}$$

where o_i is the observed outcome of executing c_i.

Tool integration. Unlike symbolic simulations, the agent executes *real tools* spanning categories such as web search, visual perception, image generation/editing, file inspection, multimodal reasoning, and a broad set of Python libraries (see Tab. 5). This setup grounds

reasoning in executable actions, ensuring realistic trajectories and better generalization to practical tasks. Further implementation details are provided in Appendix §A.1.

Training objective. Given a trajectory $\tau = \{t_1, c_1, o_1, \ldots, t_n, c_n, o_n\}$ paired with query Q and resources $\mathcal{F}^*$, the controller is optimized with a step-level cross-entropy objective:

$$\mathcal{L}_{\text{SFT}} = \mathbb{E}_{(Q,\mathcal{F}^*,\tau,A)\sim\mathcal{D}_{\texttt{M-TRACE}}}\Big[-\sum_{i=1}^{n}\log P_\theta(t_i, c_i \mid Q, \mathcal{F}^*, \mathcal{H}_i)\Big]. \tag{3}$$

Crucially, the final answer $\mathcal{A}$ is *not supervised*, forcing the controller to rely on tool interactions rather than memorized knowledge. This design grounds the model in executable tool use and sets the stage for further refinement with preference optimization.

3.3 STAGE 2: Preference Tuning

While `SFT` equips the controller with high-quality demonstrations, it remains restricted to imitation, limiting the ability to refine tool usage, recover from partially correct rollouts, or adapt beyond static trajectories. To overcome this, we leverage step-wise preference optimization on 11K preference pairs (`Pref-X`), enabling the agent to compare candidate actions and learn to favor accurate, consistent, and semantically useful behaviors. This reinforcement-style refinement improves robustness and adaptability across multimodal tasks (see Fig. 4).

Formulation. As in Stage 1, we adopt the ReAct framework (Yao et al., 2023b), where at step i, the agent generates an action $a_i = (t_i, c_i)$ consisting of a natural-language *thought* t_i and executable *code* c_i. Given query $\mathcal{Q}$, optional artifacts $\mathcal{F}^*$, history $h_i = \{t_1, c_1, o_1, \ldots, t_{i-1}, c_{i-1}, o_{i-1}\}$, and tool set $\mathcal{T}$, the controller selects:

$$t_i^\star, c_i^\star = \arg\max_{t_i, c_i} \pi_\theta(t_i, c_i \mid \mathcal{Q}, \mathcal{F}^*, h_i, \mathcal{T}), \tag{4}$$

where π_θ is the Stage 1 SFT-initialized controller.

Preference data synthesis. We construct `Pref-X`, a dataset of 11K preference pairs. Starting from seed queries in `M-TRACE`, an `LLM` (e.g., Qwen2.5-7B) expands queries and specifies artifact types. Relevant images are retrieved via embedding search, while documents (DOCX, PPTX, XLSX, PDF) are synthesized programmatically. Each task thus consists of $(\mathcal{Q}, \mathcal{F}^*)$, enriched with realistic multimodal context (see Appendix §A).

Step exploration and verification. At step i, the controller proposes multiple candidates $\{a_i^1, \ldots, a_i^n\}$, each executed to yield outcomes $\{o_i^1, \ldots, o_i^n\}$. An `LLM`-based verifier, conditioned on $(\mathcal{Q}, h_i)$, compares these outcomes and selects the most reliable action $a_i^{\text{pre}} = (t_i^\star, c_i^\star)$. The remaining candidates form the dispreferred set D_i^{dis}. A task with m steps yields $m(n-1)$ preference pairs:

$$\mathcal{D} = \{(x_i, a_i^{\text{pre}}, a_i^{\text{dis}}) \mid i \in [1, m]\}. \tag{5}$$

`Pref-X` pipeline. Unlike traditional RLHF approaches based on PPO (Schulman et al., 2017), which require reward modeling and costly reinforcement learning updates, Direct Preference Optimization (DPO) directly optimizes over preference pairs (Rafailov et al., 2023). It leverages a fixed reference policy to stabilize training, avoids the need for explicit reward models, and is significantly more computationally efficient. This makes DPO particularly well-suited for step-wise preference tuning, where fine-grained comparisons are abundant but full reinforcement learning would be prohibitively expensive. To create preference-labeled trajectories, we combine two complementary components: *step sampling* and *step verification*. Instead of relying on static demonstrations, we employ an online exploration scheme (Fig. 2) where the agent iteratively samples actions and verifies their quality within each task.

At step i, the controller proposes n candidate actions $\{a_i^1, a_i^2, \ldots, a_i^n\}$, each decomposed into (t_i^k, c_i^k), which are executed to yield observations $\{o_i^1, \ldots, o_i^n\}$. We then prompt an `LLM`-based verifier with the query Q, history h_i, candidate actions, and corresponding observations, and select the most reliable action $(t_i^\star, c_i^\star, o_i^\star)$. This process expands the trajectory step by step until the task is solved.

Table 1: **Overall results on Agent-X**. Best values in each column (within open/closed-source) are in **bold**, and second-best are underlined. Metrics are detailed in Appendix

Model	Step-by-Step			Deep Reasoning				Outcome		
	G_s	T_p	T_{acc}	F_{acc}	C_s	F_p	S_{acc}	G_{acc}	G_a^*	T_{acc}^s
Closed-source										
Gemini-2.5-Pro	0.40	0.36	0.81	0.72	0.48	0.64	0.73	0.40	0.56	0.62
GPT-4o	**0.60**	**0.47**	0.72	**0.81**	**0.57**	**0.79**	0.59	0.37	**0.70**	**0.68**
OpenAI-o4-mini	0.42	0.32	**0.89**	0.71	0.51	0.60	**0.80**	**0.45**	0.67	0.63
Open-source										
Phi-4-VL-Instruct	0.13	0.21	0.24	0.61	0.19	0.47	0.40	0.11	0.26	0.42
InternVL2.5-8B	0.45	0.31	0.47	0.68	0.47	0.52	0.60	0.28	0.55	0.58
Gemma-3-4B	0.26	0.30	0.78	0.61	0.54	0.38	0.54	0.27	0.67	0.60
InternVL3-8B	0.46	0.34	0.54	0.68	0.45	0.70	0.40	0.20	0.59	0.62
VideoLLaMA3-7B	0.45	0.28	0.46	0.65	0.46	0.62	0.54	0.28	0.54	0.54
Qwen2-VL-7B	0.51	0.39	0.54	0.62	0.41	0.34	0.38	0.25	0.55	0.57
Ours										
MATRIX (Ours)	**0.59**	**0.44**	**0.91**	**0.71**	0.48	**0.88**	**0.71**	**0.39**	**0.76**	**0.77**
Baseline Improvement (Qwen2-VL-7B)	+8%	+5%	+37%	+9%	+7%	+54%	+33%	+14%	+21%	+20%

Closed-source results shown for reference; best/second-best highlighting applies only to Open-source models.

The preference data is constructed in a pairwise manner: for each input x_i, the selected best action $a_i^{\text{pre}} = (t_i^\star, c_i^\star)$ serves as the *preferred* label, while the remaining candidates $\{a_i^j\}_{j \neq \star}$ form the *dispreferred* set D_i^{dis}. A single task with m steps thus yields $m(n-1)$ preference pairs, summarized as

$$\mathcal{D} = \{(x_i, a_i^{\text{pre}}, a_i^{\text{dis}}) \mid i \in [1, m]\}. \tag{6}$$

Objective. Given the constructed dataset $\mathcal{D}$, we optimize the controller using the Direct Preference Optimization (DPO) objective (Kong et al., 2025):

$$\mathcal{L}(\theta) = -\mathbb{E}_{(x_i, a_i^{\text{pre}}, a_i^{\text{dis}}) \sim \mathcal{D}} \left[\log \sigma \Big(\beta \big(\log \frac{\pi_\theta(a_i^{\text{pre}} | x_i)}{\pi_{\text{ref}}(a_i^{\text{pre}} | x_i)} - \log \frac{\pi_\theta(a_i^{\text{dis}} | x_i)}{\pi_{\text{ref}}(a_i^{\text{dis}} | x_i)} \big) \Big) \right], \tag{7}$$

where π_{ref} is the reference controller (obtained after supervised fine-tuning), β controls deviation from the reference, and $\sigma(\cdot)$ is the logistic function.

Training scheme. The final MATRIX controller is trained in two phases: (i) *Trajectory-driven SFT* on 177K verified traces (`M-TRACE`), grounding step-wise tool reasoning. (ii) *Preference tuning* on 11K preference pairs (`Pref-X`), where the agent self-explores, generates candidate actions, and updates via the DPO objective. This staged design allows the agent to benefit from verified traces while progressively improving decision-making through exploration and preference alignment. A summary of the training loop is given in Algorithm 1.

4 Results

We evaluate `MATRIX` across three challenging multimodal agent benchmarks. **Agent-X** (Ashraf et al., 2025) comprises 828 tasks spanning six environments (e.g., web browsing, driving, sports), requiring fine-grained step-wise reasoning. **GTA** (Wang et al., 2024b) consists of 229 real-world queries paired with authentic tools and multimodal inputs, emphasizing long-horizon tool usage. **GAIA** (Mialon et al., 2023) provides 106 open-ended multimodal questions covering diverse reasoning levels and task domains.

We benchmark against: (i) *closed-source controllers* (GPT-4, GPT-4o), (ii) *open-source controllers* (LLaVA-NeXT-8B (Liu et al., 2024a), InternVL2-8B (Chen et al., 2024c), Qwen2-VL-7B (Yang et al., 2024), MiniCPM-V-8.5B (Yao et al., 2024)), and (iii) *agent baselines* (Lego (Team, 2023), Sibyl (Wang et al., 2024c), Warm-up Act (Mialon et al., 2023), HF Agent (HuggingFace Contributors, 2024)). Beyond head-to-head comparisons, we conduct ablations on data generation and preference tuning, and provide qualitative case studies illustrating how `MATRIX` solves complex multimodal tasks through adaptive tool reasoning.

Table 2: **Results on GTA and GAIA benchmarks.** Bold numbers indicate the best performance among open-source models, underline denotes the second best.

Method	Controller	GTA	GAIA			
		AnsAcc	Level 1	Level 2	Level 3	AnsAcc
Closed-source Controllers						
Lego Agent	GPT-4	46.59	-	-	-	-
Lego Agent	GPT-4o	41.52	-	-	-	-
Sibyl Agent	GPT-4-turbo	-	43.40	27.90	7.70	29.70
Warm-up Agent	GPT-4-turbo	-	30.20	15.10	0.00	17.60
HF Agent	GPT-4o	57.05	47.17	31.40	11.54	33.40
HF Agent	GPT-4o-mini	57.69	33.96	27.91	3.84	26.06
Open-source Controllers						
HF Agent	InternVL2-8B	32.05	7.55	4.65	0.00	4.85
HF Agent	MiniCPM-V-8.5B	33.97	13.21	5.81	0.00	7.27
HF Agent	Qwen2-VL-7B	42.31	16.98	8.14	0.00	9.70
T3-Agent	MAT-MiniCPM-V-8.5B	<u>52.56</u>	<u>26.42</u>	11.63	3.84	<u>15.15</u>
T3-Agent	MAT-Qwen2-VL-7B	53.85	26.42	<u>15.12</u>	<u>3.84</u>	16.97
Ours						
MATRIX Agent	Tuned Qwen2-VL-7B	**65.38 ± 4%**	**29.15 ± 4%**	**19.28 ± 2%**	**6.84 ± 3%**	**21.47 ± 3%**
Improvement over Qwen2-VL-7B		+23.07%	+12.17%	+11.14%	+6.84	+11.77%

The variance and error study are given in Appendix§ C.1

4.1 Experimental Setup

Implementation. We adopt Qwen2-VL-7B (Yang et al., 2024) as the controller and fine-tune the language backbone with LoRA (Hu et al., 2022). Training runs for five epochs on `M-TRACE` using LoRA rank 32 applied to query, key, and value projections in all attention layers. We optimize with AdamW (lr=1×10^{-6}), cosine annealing, batch size 2 per device, and a 10,240-token context window. Experiments use 4×H200 GPUs, and inference is performed without sampling or verification for fair comparison.

Evaluation Metrics. Following prior works (Ashraf et al., 2025; Wang et al., 2024b; Gao et al., 2025a), we evaluate performance of Agent-X using three modes: `Step-by-Step` (correctness of individual tool-use steps), `Deep Reasoning` (coherence and factual accuracy of multi-step reasoning), and `Outcome` (overall task-solving success via final answers and tool execution). For GTA and GAIA, we report `AnsAcc`, with GAIA results further broken down by difficulty levels (Level 1, Level 2, and Level 3).

4.2 State-of-the-art Comparisons

Agent-X: Tab. 1 shows that while open-source models like `Qwen2-VL-7B`, `InternVL3-8B`, and `VideoLLaMA3-7B` improve grounding and factual precision on Agent-X, they remain behind closed-source controllers (e.g., GPT-4o, Gemini). Key metrics for Agent-X include **`Tool Accuracy`** (correct execution), **`Faithfulness Accuracy`** (evidence alignment), and **`Semantic Accuracy`** (contextual fit). `MATRIX` achieves the highest scores, 0.91, 0.71, and 0.71, respectively, yielding relative gains of +8% grounding, +5% precision, +37% tool accuracy, and +50% factual precision over `Qwen2-VL-7B`. These results confirm that step-wise preference optimization with AI feedback substantially enhances grounding with tool-use reasoning and offers a scalable open-source alternative.

GTA and GAIA: Tab. 2 reports results on GTA and GAIA. On GTA, `MATRIX` outperforms both closed-source (GPT-4/4o) and open-source (InternVL2-8B, Qwen2-VL-7B) controllers, with a +23.07% `AnsAcc` gain over Qwen2-VL-7B. Compared to SFT-based methods like T3-Agent, it shows clear advantages from self-exploration and preference refinement, relying less on costly annotations. On GAIA, Matrix-Agent achieves best performance among open-source models, surpassing Qwen2-VL-7B by +11.77% in `AnsAcc`. While a small gap remains to closed-source models, we attribute this to scale and proprietary data. The results validate the effectiveness of our step-wise preference optimization for multimodal tool-use.

Table 3: **Ablation studies for MATRIX on GTA** *Left*: Effect of steps (d). *Middle*: BLEU scores for verifier discrimination (lower is better). *Right*: Effect of two-verifier design.

Iteration Steps (d)			
d	200	500	1000
AnsAcc	55.17	**65.38**	60.50

Verifier BLEU (↓)				
Verifier	B1	B2	B3	B4
Random	0.53	0.41	0.36	0.34
Ours	**0.21**	**0.22**	**0.19**	**0.17**

Two-Verifier Ablation		
Method	GTA	GAIA
w/o verifiers	50.00	13.33
Ours	**65.38**	**19.28**

Table 4: **Ablation studies for MATRIX on Agent-X.** *Left*: Effect of computation budget. *Middle*: Effect of dataset size. *Right*: Comparison of different RL methods.

GPU Compute			
Data	8K	17.5K	28.5K
Memory (GB)	221	270	318

Dataset Size (samples)			
Size	8K	17.5K	28.5K
Goal_Acc	0.29	0.35	**0.39**

RL Method Comparison			
Method	SFT	ORPO	DPO
Goal_Acc	0.31	0.37	**0.39**

4.3 Ablation and Analysis

Effect of Iteration Step Size. The iteration step size d controls the trade-off between update frequency (how often the policy is updated) and data diversity (breadth of sampled trajectories). A very small d (e.g., 200) yields frequent updates but limited diversity, while a large d (e.g., 1000) increases diversity at the cost of slower adaptation. As shown in Tab. 3(*left*), $d = 500$ achieves the best balance, giving the highest AnsAcc of 65.38%.

Verifier Discrimination Ability. We measure how well our verifier distinguishes candidate steps by comparing it with random selection using BLEU scores (lower is better, since lower overlap means more diverse actions). Tab. 3(*middle*) shows that our verifier achieves consistently lower BLEU (e.g., BLEU-1 = 0.21 vs. 0.53 for random), indicating it selects more distinct and informative steps, which translates to improved AnsAcc.

Impact of Dual-Verifier Framework. We further ablate the two-verifier design by removing one verifier. As reported in Tab. 3(*right*), performance drops substantially (GTA: 65.38% → 50.00%, GAIA: 19.28% → 13.33%), confirming that combining both verifiers is critical for filtering inconsistent or low-quality samples.

Ablation on Dataset Scale, Memory, and Optimization Methods. Tab. 4 summarizes the effect of training data size and tuning strategies. Increasing the dataset from 8K to 28.5K samples raises memory usage (from 221 GB to 318 GB across 4×H200 GPUs) but yields steady gains in Goal_Acc (0.29 → **0.39**). On the optimization side, ORPO improves over pure SFT (0.37 vs. 0.31), while DPO achieves the highest score (**0.39**), underscoring the effectiveness of preference-based tuning for step-level tool reasoning.

Additional ablations on variance analysis, tool preference, and modality contributions are in Appendix §C. Qualitative/failure case analysis is shown in Appendix §D.

5 Conclusion

We introduced MATRIX, a vision-centric framework for multimodal agent tuning that advances tool-use reasoning through staged training. MATRIX combines large-scale trajectory supervision (M-TRACE) with step-wise preference optimization (Pref-X), enabling agents to both acquire fundamental tool-use skills and refine their decision-making beyond imitation. This unified design achieves consistent gains across Agent-X, GTA, and GAIA, surpassing existing baselines. Our results highlight the scalability and effectiveness of integrating synthetic data generation with iterative self-exploration for building robust multimodal agents. =

Limitations and Future Directions. While effective, MATRIX has some limitations. Currently, it only grounds multimodal signals at the query/task level, relies on prompt-based verifiers that may falter under distribution shifts, and optimizes step-level preferences without trajectory-level credit assignment. Future work will address these by exploring adaptive verifiers, continuous multimodal grounding, and hierarchical preference modeling.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 2

Tajamul Ashraf, Amal Saqib, Hanan Ghani, Muhra AlMahri, Yuhao Li, Noor Ahsan, Umair Nawaz, Jean Lahoud, Hisham Cholakkal, Mubarak Shah, Philip Torr, Fahad Shahbaz Khan, Rao Muhammad Anwer, and Salman Khan. Agent-x: Evaluating deep multimodal reasoning in vision-centric agentic tasks. 2025. URL https://arxiv.org/abs/2505.24876. 2, 3, 7, 8

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023. 2

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024. 2

Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18392–18402, 2023. 20

Harrison Chase. Langchain, October 2022. URL https://github.com/langchain-ai/langchain. 3

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Step-level value preference optimization for mathematical reasoning. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7889–7903, 2024a. 3

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. 2024b. 4

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 24185–24198, 2024c. 7

Yu Du, Fangyun Wei, and Hongyang Zhang. Anytool: Self-reflective, hierarchical agents for large-scale api calls. In *International Conference on Machine Learning (ICML)*, pp. 11812–11829, 2024. 3

Yue Fan, Xiaojian Ma, Rujie Wu, Yuntao Du, Jiaqi Li, Zhi Gao, and Qing Li. Videoagent: A memory-augmented multimodal agent for video understanding. In *European Conference on Computer Vision (ECCV)*, 2024. 1

Yuwei Fu, Haichao Zhang, Di Wu, Wei Xu, and Benoit Boulet. Furl: visual-language models as fuzzy rewards for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 14256–14274, 2024. 3

Zhi Gao, Yuntao Du, Xintong Zhang, Xiaojian Ma, Wenjuan Han, Song-Chun Zhu, and Qing Li. Clova: A closed-loop visual assistant with tool usage and update. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13258–13268, 2024. 1

Zhi Gao, Bofei Zhang, Pengxiang Li, Xiaojian Ma, Tao Yuan, Yue Fan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, and Qing Li. Multi-modal agent tuning: Building a vlm-driven agent for efficient tool usage. In *International Conference on Learning Representations (ICLR)*, 2025a. 8

Zhi Gao, Bofei Zhang, Pengxiang Li, Xiaojian Ma, Tao Yuan, Yue Fan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, and Qing Li. Multi-modal agent tuning: Building a vlm-driven agent for efficient tool usage. In *The Thirteenth International Conference on Learning Representations*, 2025b. 1, 2, 4

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 2

Significant Gravitas. Autogpt, 2023. URL https://github.com/Significant-Gravitas/AutoGPT. 3

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14953–14962, 2023. 1

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. 8

HuggingFace Contributors. Agents and tools, 2024. URL https://huggingface.co/docs/transformers/agents. 7

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. 4

brian ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, pp. 287–318, 2023. 1

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *International Conference on Computer Vision (ICCV)*, pp. 4015–4026, 2023. 4

Aobo Kong, Wentao Ma, Shiwan Zhao, Yongbin Li, Yuchuan Wu, Ke Wang, Xiaoqian Liu, Qicheng Li, Yong Qin, and Fei Huang. Sdpo: Segment-level direct preference optimization for social agents. *arXiv preprint arXiv:2501.01821*, 2025. 2, 3, 7

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms. *arXiv preprint arXiv:2406.18629*, 2024. 3

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Ren Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, et al. Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback. In *International Conference on Machine Learning (ICML)*, pp. 26874–26901. PMLR, 2024. 3

Chenliang Li, He Chen, Ming Yan, Weizhou Shen, Haiyang Xu, Zhikai Wu, Zhicheng Zhang, Wenmeng Zhou, Yingda Chen, Chen Cheng, et al. Modelscope-agent: Building your customizable agent system with open-source large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 566–578, 2023. 3

Jian Li, Yabiao Wang, Changan Wang, Ying Tai, Jianjun Qian, Jian Yang, Chengjie Wang, Jilin Li, and Feiyue Huang. Dsfd: dual shot face detector. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5060–5069, 2019. 20

Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, Hongxuan Lu, Tianrui Qin, Chenghao Zhu, Yi Yao, Shuying Fan, Xiaowan Li, Tiannan Wang, Pai Liu, King Zhu, He Zhu, Dingfeng Shi, Piaohong Wang, Yeyi Guan, Xiangru Tang, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Jiaheng Liu, Ge Zhang, and Wangchunshu Zhou. Chain-of-agents: End-to-end agent foundation models via multi-agent distillation and agentic rl, 2025. URL https://arxiv.org/abs/2508.13167. 2

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*, pp. 740–755. Springer, 2014. 4, 26

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in Neural Information Processing Systems*, 36, 2024a. 4, 7

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *International Conference on Computer Vision (ICCV)*, 2015. 4

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh RN, et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:54463–54482, 2024b. 3, 4

Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 2263–2279, 2022. 4

Kai Mei, Xi Zhu, Wujiang Xu, Wenyue Hua, Mingyu Jin, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. Aios: Llm agent operating system. *arXiv preprint arXiv:2403.16971*, 2024. 3

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023. 2, 3, 7

Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen, et al. Simple open-vocabulary object detection. In *European Conference on Computer Vision (ECCV)*, pp. 728–755. Springer, 2022. 20

Emanuele Musumeci, Michele Brienza, Vincenzo Suriani, Daniele Nardi, and Domenico Daniele Bloisi. Llm based multi-agent generation of semi-structured documents from semantic templates in the public administration domain. In *International Conference on Human-Computer Interaction*, pp. 98–117. Springer, 2024. 3

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021. 3

Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu Wei. Kosmos-2: Grounding multimodal large language models to the world. *arXiv preprint arXiv:2306.14824*, 2023. 2

Yujia Qin, Zihan Cai, Dian Jin, Lan Yan, Shihao Liang, Kunlun Zhu, Yankai Lin, Xu Han, Ning Ding, Huadong Wang, et al. Webcpm: Interactive web search for chinese long-form question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8968–8988, 2023. 3

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. 3

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023. 6

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, 2022. 20

Babak Saleh and Ahmed Elgammal. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *arXiv preprint arXiv:1505.00855*, 2015. 4

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 6

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. *arXiv preprint arXiv:2311.18760*, 2023. 1, 3

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024. 3

Amanpreet Singh, Vivek Natarjan, Meet Shah, Yu Jiang, Xinlei Chen, Devi Parikh, and Marcus Rohrbach. Towards vqa models that can read. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8317–8326, 2019. 4

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. Restgpt: Connecting large language models with real-world applications via restful apis. *arXiv preprint arXiv:2306.06624*, 2023. 3

Yueqi Song, Frank Xu, Shuyan Zhou, and Graham Neubig. Beyond browsing: Api-based web agents. *arXiv preprint arXiv:2410.16464*, 2024. 3

Xiaowen Sun, Xufeng Zhao, Jae Hee Lee, Wenhao Lu, Matthias Kerzel, and Stefan Wermter. Details make a difference: Object state-sensitive neurorobotic task planning. *arXiv preprint arXiv:2406.09988*, 2024. 2

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *International Conference on Computer Vision (ICCV)*, pp. 11888–11898, 2023. 1

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023. 3

AgentLego Developer Team. Enhance llm agents with versatile tool apis. https://github.com/InternLM/agentlego, 2023. 7

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023. 2

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 16022–16076, 2024. 1

Chenyu Wang, Weixin Luo, Sixun Dong, Xiaohua Xuan, Zhengxin Li, Lin Ma, and Shenghua Gao. Mllm-tool: A multimodal large language model for tool agent learning. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 6678–6687. IEEE, 2025. 3

Huaijie Wang, Shibo Hao, Hanze Dong, Shenao Zhang, Yilin Bao, Ziran Yang, and Yi Wu. Offline reinforcement learning for llm multi-step reasoning. *arXiv preprint arXiv:2412.16145*, 2024a. 3

Jize Wang, Ma Zerun, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. Gta: a benchmark for general tool agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024b. 2, 3, 7, 8

Junke Wang, Lingchen Meng, Zejia Weng, Bo He, Zuxuan Wu, and Yu-Gang Jiang. To see is to believe: Prompting gpt-4v for better visual instruction tuning. *arXiv preprint arXiv:2311.07574*, 2023. 4

Yulong Wang, Tianhao Shen, Lifeng Liu, and Jian Xie. Sibyl: Simple yet effective agent framework for complex real-world reasoning. *Arxiv*, 2024c. URL https://arxiv.org/abs/2407.10718. 7

Zhenyu Wang, Aoxue Li, Zhenguo Li, and Xihui Liu. Genartist: Multimodal llm as an agent for unified image generation and editing. *arXiv preprint arXiv:2407.05600*, 2024d. 2

T. Weyand, A. Araujo, B. Cao, and J. Sim. Google Landmarks Dataset v2 - A Large-Scale Benchmark for Instance-Level Recognition and Retrieval. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 4

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023. 1

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis Ioannidis, Karthik Subbian, Jure Leskovec, and James Y Zou. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *Advances in Neural Information Processing Systems*, 37:25981–26010, 2024. 3

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024. 3

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024. 1, 5, 7, 8

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022. 3

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023a. 4

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b. 5, 6

Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, et al. Minicpm-v: A gpt-4v level mllm on your phone. *arXiv preprint arXiv:2408.01800*, 2024. 7

Tianyu Yu, Haoye Zhang, Yuan Yao, Yunkai Dang, Da Chen, Xiaoman Lu, Ganqu Cui, Taiwen He, Zhiyuan Liu, Tat-Seng Chua, et al. Rlaif-v: Aligning mllms through open-source ai feedback for super gpt-4v trustworthiness. *arXiv preprint arXiv:2405.17220*, 2024. 3

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R Fung, Hao Peng, and Heng Ji. Craft: Customizing llms by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations (ICLR)*, 2024. 1

Chaoyun Zhang, Shilin He, Liqun Li, Si Qin, Yu Kang, Qingwei Lin, and Dongmei Zhang. Api agents vs. gui agents: Divergence and convergence. *arXiv preprint arXiv:2503.11069*, 2025. 3

Ruohong Zhang, Bowen Zhang, Yanghao Li, Haotian Zhang, Zhiqing Sun, Zhe Gan, Yinfei Yang, Ruoming Pang, and Yiming Yang. Improve vision language model chain-of-thought reasoning. *arXiv preprint arXiv:2410.16198*, 2024a. 4

Ziniu Zhang, Shulin Tian, Liangyu Chen, and Ziwei Liu. Mmina: Benchmarking multihop multimodal internet agents. *arXiv preprint arXiv:2404.09992*, 2024b. 3

Appendix for Matrix

Appendix Contents

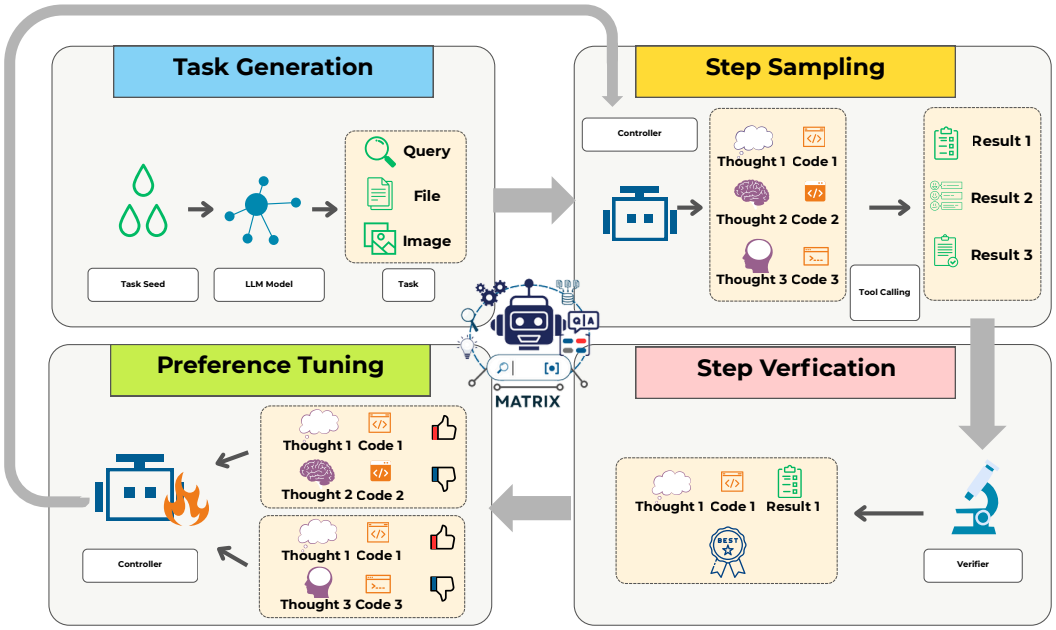


Figure 4: Overview of the `Pref-X` construction pipeline. Tasks are sampled from `M-TRACE`, then expanded through step sampling, step verification, and preference collection. Verified preference pairs are used in Direct Preference Optimization (DPO) to update the controller.

A Additional Details on Pref-X Pipeline

In the main paper (§3.2), we described how `Pref-X` is constructed to enable step-wise preference optimization. Here, we provide further details of the data generation and verification pipeline, illustrated in Fig. 4.

Task Generation. We begin by sampling a set of seed tasks $\mathcal{S}$ from the Stage 1 corpus (`M-TRACE`, see §3.2). Each task is defined as a query–file pair (Q, F), where Q denotes the user query and F contains the associated multimodal evidence (e.g., text files, images). Candidate step-wise trajectories are produced by the current controller π_θ through iterative interaction with the toolset.

Step Sampling. For each task (Q, F), the controller generates multiple candidate steps at each reasoning turn. These steps include tool calls, arguments, and intermediate reasoning traces. From this pool, diverse samples are retained to avoid mode collapse and to ensure broad coverage of possible reasoning paths. To contextualize our contributions, we position MATRIX against representative RL based sampling frameworks. As illustrated in Fig. 5, the comparison spans three axes: task domain, collection granularity, and annotation format. Unlike prior methods that primarily operate in narrow domains with trajectory-level rewards, MATRIX emphasizes diverse multimodal tasks, collects preferences at the step level, and leverages executable tool feedback for scalable and precise supervision.

Step Verification. Each sampled step is then automatically verified. Verification checks whether (i) the tool call matches the schema, (ii) the arguments are valid and executable, and (iii) the intermediate output remains consistent with the task context. Invalid or incomplete steps are filtered out. This process corresponds to the loop over history states h_i in Algorithm 1.

Preference Collection. For the verified steps, pairwise preferences are collected using a mixture of automated heuristics and model-based evaluators. Preference signals capture

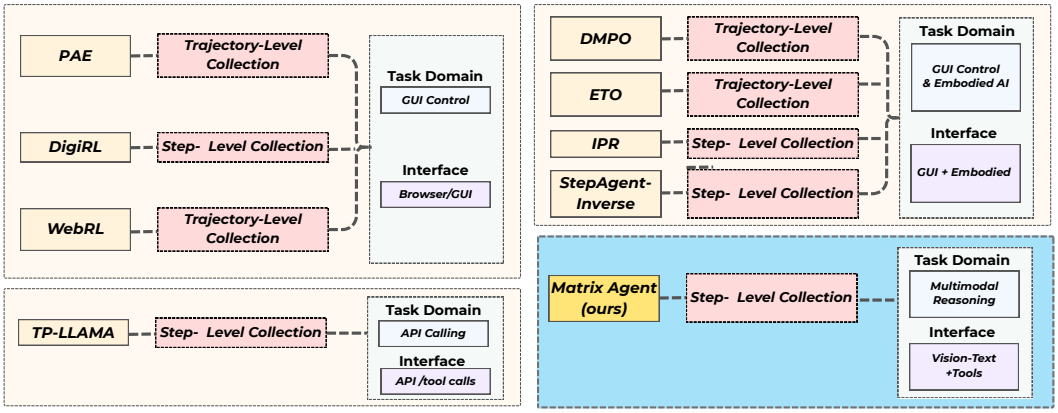


Figure 5: **Comparison of preference data construction frameworks.** We contrast MATRIX with reinforcement-learning–based sampling pipelines across three key dimensions: (1) *Task domain*, i.e., the scope and modality of tasks considered; (2) *Collection granularity*, i.e., whether data is gathered at the full-trajectory or step level; and (3) *Annotation format*, i.e., the type of supervision used for preference signals. MATRIX uniquely focuses on multimodal tasks, constructs preferences at the step level, and employs executable tool feedback for scalable, fine-grained supervision.

relative correctness, faithfulness to the query, and progression towards the goal. The resulting preference pairs form the core supervision signal for preference optimization.

Four Stages of DPO Process. The overall `Pref-X` construction pipeline aligns with the four-step DPO process outlined in Fig. 4:

1. *Trajectory sampling:* Generate candidate step-wise rollouts from π_θ for each seed task.
2. *Step verification:* Discard malformed or invalid steps using automatic schema and execution checks.
3. *Preference generation:* Construct preference pairs by comparing valid candidate steps.
4. *Policy update:* Apply Direct Preference Optimization (DPO) to update π_θ against the reference policy π_{ref} using the verified preference dataset $\mathcal{D}$.

This iterative pipeline produces the `Pref-X` corpus, which directly supervises step-wise improvements of the agent while preventing reliance on memorized final answers.

A.1 Tools Used

To enable flexible and realistic multimodal task solving, our framework integrates a diverse set of tools spanning vision, web, file understanding, and generative capabilities. Below, we provide details of each tool and justify its inclusion. Together, these tools allow agents to ground reasoning in real data, perform cross-modal analysis, and produce actionable outputs.

Web Search. This tool is implemented as a meta-agent consisting of three sub-modules: (i) *SearchInformation*, which retrieves candidate webpages given a query, (ii) *Visit*, which extracts textual content from webpages, and (iii) *WebQA*, which performs targeted question answering over retrieved text. This design ensures the agent can dynamically access and reason over up-to-date web knowledge rather than relying solely on static training data.

Image Question Answering. We integrate GPT-4o-mini as an image QA module, which accepts an image and a natural language question to output a textual answer. This capability allows the agent to perform grounded reasoning over visual inputs such as charts, natural images, or scanned documents.

Table 5: Overview of tools used in **Matrix-Agent**. Each tool specifies the model or library backbone and its primary functionality.

Tool	Model / Package	Functionality
Web Search	Google search + GPT	Sub-tools: *SearchInformation* (title/abstract/URL), *Visit* (HTML → text), *WebQA* (Q&A on text).
Image QA	GPT	Answers questions given an image input.
File Inspector	MarkdownConverter + GPT-4o-mini	Converts multi-modal files into markdown and performs Q&A.
Object Localization	OWL-ViT (Minderer et al., 2022)	Detects objects in images and outputs bounding boxes.
Image Generation	Stable Diffusion (Rombach et al., 2022)	Generates an image from a text query.
Image Editing	InstructPix2Pix (Brooks et al., 2023)	Edits an image according to an instruction.
Face Detection	DSFD (Li et al., 2019)	Detects and outputs bounding boxes of faces in an image.
Python Package	Standard libraries + packages	Enables code execution with: `requests`, `pandas`, `numpy`, `scipy`, `torch`, `cv2`, etc.

File Inspector. For structured documents (PDFs, Word, Excel, PowerPoint), we use the Python package `MarkdownConverter` to parse raw files into text. The resulting content is combined with a query and passed to GPT-4o-mini for reasoning. This tool extends the agent's ability to understand heterogeneous non-image resources, which are common in real-world multimodal tasks.

Object Localization. We employ OWL-ViT (Minderer et al., 2022) for object grounding. Given an image and a query (e.g., "localize all cups"), the tool outputs bounding boxes for relevant objects. This allows the agent to handle spatial reasoning and locate specific entities in visual contexts.

Image Generation. Stable Diffusion (Rombach et al., 2022) is used for text-to-image generation, enabling agents to produce synthetic images from natural language prompts. This supports tasks such as visualization, illustration, or generating multimodal artifacts required by a query.

Image Editing. We incorporate InstructPix2Pix (Brooks et al., 2023), which takes an instruction and an input image to output a modified version. This capability is essential for tasks requiring visual manipulation, such as highlighting regions, altering attributes, or iterative refinement of generated content.

Face Detection. We use DSFD (Li et al., 2019) as a robust face detection backbone. It identifies bounding boxes of all visible faces in an image. Face-level grounding is a critical capability for tasks involving identity verification, demographic analysis, or interaction reasoning.

Python Package Execution. Finally, we allow the agent to call a curated set of Python packages (`pandas`, `numpy`, `matplotlib`, `torch`, etc.). This provides a flexible computational backend for data analysis, symbolic reasoning, and numerical tasks. By combining tool execution with code-level reasoning, the agent can go beyond natural language planning and solve complex multimodal problems.

In summary, these tools collectively enable `MATRIX` to handle tasks requiring perception, reasoning, retrieval, and generation across diverse modalities. The broad coverage of tool categories (search, vision, file understanding, generation, and computation) ensures the agent is capable of solving realistic and complex multimodal tasks.

B MATRIX Training Algorithm

To complement the description in the main paper, we provide a detailed summary of the training pipeline and its algorithmic formulation (Algorithm 1).

Overview. After Stage 1 supervised fine-tuning (SFT), the agent is refined with **step-wise preference optimization**. Unlike static imitation learning, this stage enables the controller

to actively *explore* multiple candidate actions per reasoning step and receive structured feedback from AI-based verifiers. This design addresses three limitations of pure imitation: (i) it improves adaptability by allowing recovery from suboptimal or partially correct rollouts, (ii) it leverages exploration rather than discarding incomplete or noisy demonstrations, and (iii) it scales preference data construction without requiring expensive manual annotations.

Process. The algorithm starts with a seed task pool $\mathcal{S}$ and a controller π_θ initialized from Stage 1. For each task, the agent interacts step-by-step: it generates candidate actions, executes them through real tool calls, and submits outcomes to a verifier. The verifier compares the candidates and ranks them, producing preference pairs that distinguish consistent, accurate behaviors from weaker alternatives. These pairs are accumulated into a dataset $\mathcal{D}$, which is then used to update the controller via the Direct Preference Optimization (DPO) objective, with π_{ref} (the Stage 1 model) serving as the reference.

Algorithm 1 `MATRIX`: Iterative Step-Wise Preference Optimization

Input: Seed tasks $\mathcal{S}$, controller π_θ, reference $\pi_{\text{ref}} = \pi_\theta$
Output: Updated controller $\pi_\theta^\star$ from Stage 1.
while not converged **do**
  $\mathcal{D} \leftarrow \emptyset$
  for task $(Q, F) \in \mathcal{S}$ **do**
    $h_1 \leftarrow \emptyset$
    for $i = 1 \dots m$ **do**
      Sample candidates from π_θ
      Execute, verify, and add preferences
      Update history h_{i+1}
    end for
  end for
  Update $\pi_\theta \leftarrow \text{DPO}(\pi_\theta, \pi_{\text{ref}}, \mathcal{D})$
end while

Iteration. This loop is repeated iteratively until convergence. Over time, the agent becomes aligned with behaviors that are not only *correct* but also *robust*, *consistent*, and *semantically useful* across diverse multimodal tasks. The procedure is formalized in Algorithm 1, which illustrates the alternating phases of step-level exploration, preference pair construction, and parameter updates.

C Additional Experiments

C.1 Error Bars and Variance Analysis.

We observe small but non-negligible fluctuations across repeated runs (Tab. 6), even though the tuning pipeline itself is deterministic. The primary sources of variance stem from external API dependencies: (i) the Google Search API occasionally fails or returns unstable rankings of web results, leading to variation in retrieved evidence; (ii) the OpenAI API (used for GPT-4o-mini based verification and artifact generation) can occasionally time out or produce slightly different responses under identical prompts. These inconsistencies propagate into tool execution and trajectory verification, ultimately affecting downstream accuracy metrics by a few percentage points. Importantly, despite this natural variance, our improvements over the baseline remain statistically significant, confirming the robustness of our framework.

Table 6: Performance with variance on the GTA benchmark. Results are reported as mean ± standard deviation over 5 runs.

Method	**AnsAcc**
Baseline (Qwen2-VL-7B)	43.21
MATRIX Agent with Qwen2-VL-7B)	63.26 ± 4.78

C.2 Ablation on Modalities

To analyze the contribution of different modalities, we perform ablation experiments on the GTA benchmark. As shown in Tab. 7, removing the image modality drastically reduces

performance, with `AnsAcc` dropping by nearly 40%. This highlights the critical role of visual inputs for accurate tool-use reasoning.

Table 7: Ablation on GTA benchmark. Only `AnsAcc` is reported.

Method	AnsAcc
MATRIX Agent w/o image	8.67
MATRIX Agent w/ image	63.56

C.3 Tool Preference.

We further analyze the distribution of tools across selected and rejected steps (Fig. 6). In MATRIX, frequently adopted tools such as `visualizer` (2101 uses) and `objectloc` (1051 uses) dominate the chosen steps, while the rejected steps show heavier reliance on `objectloc` (1442 uses), `visualizer` (1524 uses), and less effective utilities such as `ocr` and `seg`. This mismatch results in a 45.62% divergence between the two distributions, suggesting that MATRIX's verifier favors tool combinations that are more semantically aligned and practically useful, while systematically filtering out noisy or redundant tool usage.

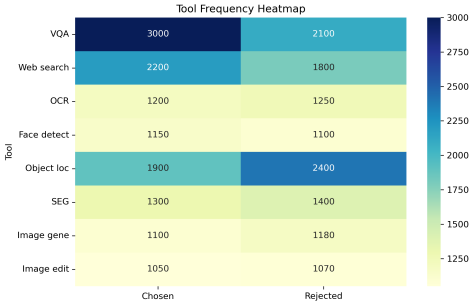


Figure 6: Tool distribution for the chosen and rejected steps.

D Qualitative and Failure Analysis

Goal. We analyze how agentic vision–language pipelines fail on image-grounded arithmetic and counting tasks, and why our MATRIX agent is more robust than the baseline (MAT). We focus on two representative cases: (i) computing the number of *boxes* of eggs required for 12 servings (discrete reasoning), and (ii) summing calories from a table (continuous arithmetic).

D.1 Example 1

Observed behavior. On visually grounded arithmetic (e.g., the "eggs/servings" task) as shown in Fig. 7, the baseline `MAT` frequently entered a *tool-use loop* where it produced the same action multiple times, received similar observations, and then repeated the action again without incorporating the feedback. This repetition culminated in a confident but incorrect answer. In contrast, `MATRIX` exhibited an initially brittle code synthesis (a parsing error and missing `print` statement) but subsequently self-corrected and produced the correct discrete count.

Failure modes in MAT. We categorize the baseline errors into three coupled modes:

1. **Mis-interpretation of the task.** The agent failed to decompose the instruction into sub-goals (extract numbers → compute total → round up to boxes), so its actions did not target missing information.
2. **Planning deficit.** Absent an explicit "plan–act–observe–revise" scaffold, the agent treated unchanged observations as if they were new evidence, never triggering a branch to an alternative tool or a reformulated query.
3. **Looping/hallucination.** Repeating identical (or template-like) tool calls despite identical observations indicates policy collapse toward a habitual trajectory, rather than belief update from evidence.

Why MATRIX eventually succeeds (but starts loose). `MATRIX's` first attempt produced (i) an invalid parse (`NoneType.group`) and (ii) no printed value, hence no usable observation. However, its reflective step modified the code to (a) explicitly compute `ceil(total_eggs/eggs_per_box)` and (b) `print` the result, restoring the tool–feedback loop and yielding the correct answer. This success originates from a minimal but effective revise-and-retry behavior whereas the brittleness stems from unconstrained code generation and weak I/O contracts.

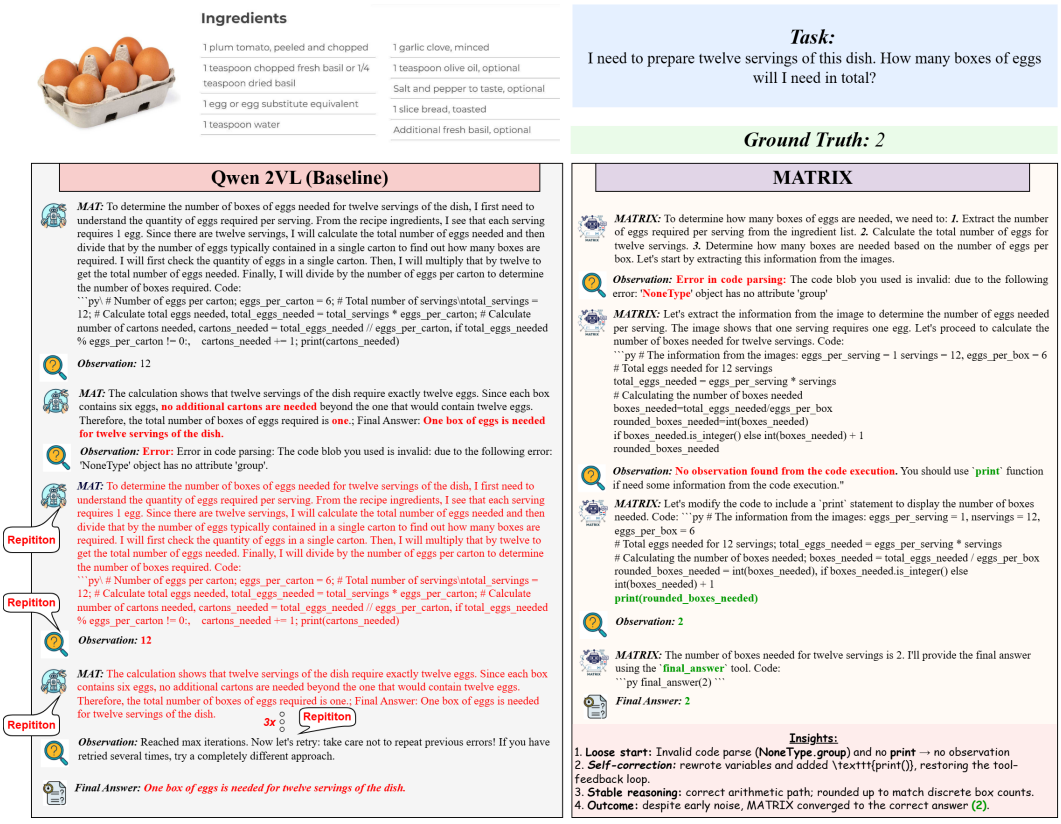


Figure 7: **Baseline vs. MATRIX.** For the given task, the baseline repeats and answers *1* (wrong), while MATRIX self-corrects after early code/IO hiccups and outputs the correct discrete count *2*.

D.2 Example 2

Observed Behavior. In Fig. 8, the baseline (MAT/Qwen-2VL) tries multiple extraction routes: *(i)* it first fails on a missing image file, then *(ii)* fails on a visualizer call, and *(iii)* finally falls back to parsing the provided image as a table text. It correctly recovers the per-item values like Egg = 157 kcal/100 g, Tomato = 19 kcal/100 g, and even logs them, but it returns them directly as the final answer rather than performing the requested aggregation, which was the main logic for the provided task. By contrast, `MATRIX` reads the same two numbers and, given the task's simplicity, directly performs the scalar sum to be 176 because of the simple nature of the task, but it finalizes without emitting any intermediate logs (no 'print'), i.e., a "no-tool path" with an empty observation buffer.

Failure Modes in MAT. The core error is task misinterpretation / incomplete aggregation as MAT stops at entity-level extraction and treats the tuple $\{157, 19\}$ as the answer, ignoring the query's composition verb ("in total"). This reflects a weak finalization contract (it permits non-scalar answers) and missing schema constraints (no requirement that the output type match a single numeric total with units). Although MAT shows robustness to I/O

issues by switching tools, its reasoning halts before the final operation (sum), so correctness is never realized.

Why `MATRIX` eventually succeeds. `MATRIX` executes the correct computation chain where it identifies two calibrated numerals and performs a deterministic addition directly, so it outputs the right scalar 176. The trade-off is observability, as it did not log inputs or intermediates, and that is why the success is unverifiable and would conceal OCR/unit mistakes if they occurred. A minimal tightening can be helpful for such scenarios, which require one logged tool step that prints the parsed inputs and the computed sum, plus unit/assert checks (per-100 g) before calling 'final_answer', would preserve the current correctness while eliminating the "answers without evidence" risk.

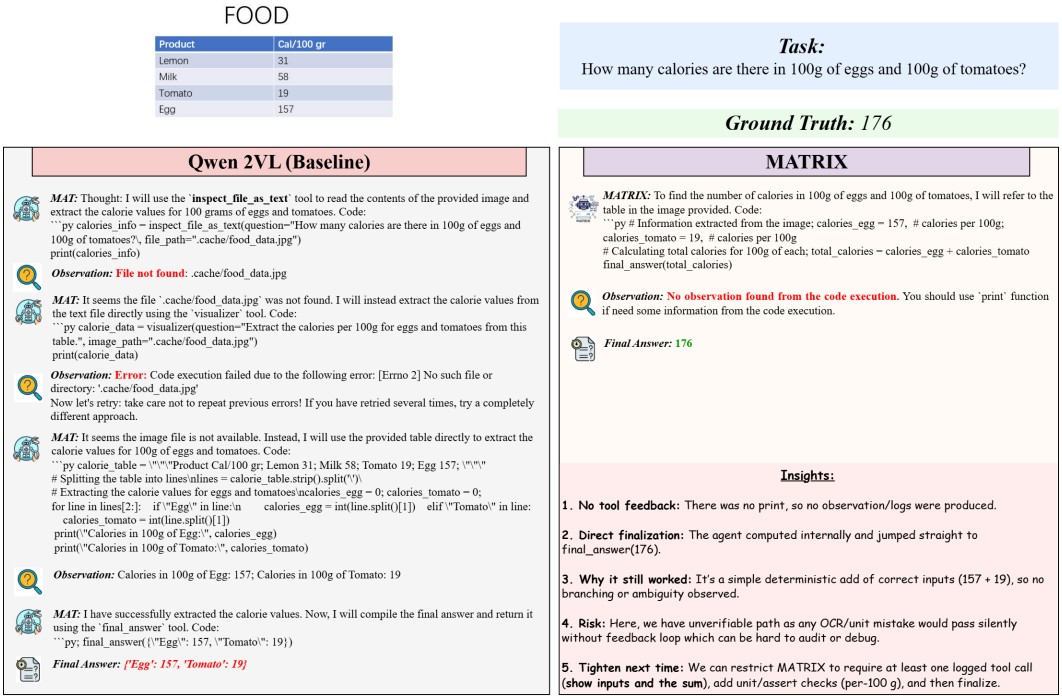


Figure 8: For the provided task, the baseline (Qwen-2VL) recovers the per-item values (157, 19) after I/O errors but finalizes them directly, failing to aggregate the required answer. `MATRIX` immediately sums to the correct scalar (176) because of the simple nature of the task, but finalizes without tool logs (empty observation), highlighting a trade-off between aggregation correctness and observability.

E Human and AI Verification Study

E.1 Human Verification of M-TRACE

Protocol. We verified our `M-TRACE` pipeline by domain experts with backgrounds in AI, programming, and science. We evaluated a total of `50` data samples drawn via random sampling, with an even split between verified `M-TRACE` cases and filtered cases. The ratings were done on a scale of 1-7 (1=very poor, 7=excellent). The label for each scale is provided in Tab. 8. For each item, an expert scored (i) the task prompt and (ii) the corresponding MATRIX trajectory.

Rubric. The task quality was evaluated from different aspects like *(i)* `Plausibility:` does the task look realistic and domain-faithful? *(ii)* `Flow:` Is the objective stated clearly with consistent constraints? *(iii)* `Multi-tool demand:` Does solving reasonably require non-trivial tool use or cross-modal steps?. Whereas, the trajectory quality was judged on:

(i) `Reasoned progress:` traceable, correct intermediate steps, *(ii)* `Code structure:` clean, runnable, and purposeful code, *(iii)* `Tool feedback use:` appropriate incorporation of tool outputs and error handling.

Table 8: 7-point rating scale used in the `M-TRACE` expert review.

Score	Label	Evaluation Criteria)
1	Very Poor	Unrealistic or unsolvable; incoherent objective; steps/code non-executable; fails most rubric criteria.
2	Poor	Major defects; missing key constraints; code largely broken; tool feedback mostly ignored.
3	Fair	Partially realistic but inconsistent; noticeable gaps; code runs only with heavy fixes; minimal multi-tool use.
4	Acceptable	Solvable with minor guidance; small inconsistencies; code mostly runs with small fixes; limited cross-modal/tool demand.
5	Good	Realistic and coherent; appropriate difficulty; code runs with minor issues; reasonable incorporation of tool outputs.
6	Very Good	Well-formed and domain-faithful; clear multi-step plan; robust, readable code; consistent, effective feedback use.
7	Excellent	Exemplary realism/clarity/complexity; clean, reusable code with error handling; optimal integration of tool feedback.

Verification Outcomes. Verified `M-TRACE` samples substantially outperformed the filtered data items on both dimensions, as can be seen in Tab. 9. On the 1–7 scale, **tasks** achieved **5.86** for verified vs. **4.61** for the filtered cases. A similar trend was observed for trajectories as well, where `M-TRACE` scored **6.12** for verified vs. **4.55** for the filtered samples. These findings support the effectiveness of the `M-TRACE` verification stage in retaining higher-quality tasks (more plausible, better-formed, and suitably challenging) and trajectories (stronger reasoning, cleaner code, and more faithful use of tool feedback).

Table 9: Human ratings for `M-TRACE`. Verified items outperform discarded ones for both task and trajectory quality.

Condition	Task	Trajectory
`M-TRACE` (kept)	**5.86**	**6.12**
Filtered (discarded)	4.61	4.55

E.2 Automatic Verification for Preference Data

To scale beyond costly human annotation, we employ automatic verification for constructing step-level preference data. At each reasoning step, large language models serve as verifiers that rank multiple candidate actions. The verifier evaluates (i) whether the action is consistent with the query and available tools, (ii) whether the tool arguments are syntactically and semantically correct, and (iii) whether the action aligns logically with the task history. This design enables the collection of high-quality preference pairs without manual effort, ensuring that noisy or inconsistent actions are filtered out before training.

E.3 Broader Impacts

MATRIX's ability to generate large-scale multimodal tasks and refine tool-use reasoning through step-level preference optimization has the potential to lower the barrier to building robust multimodal agents. By automating data synthesis and verification, MATRIX reduces reliance on costly human annotations and manual curation, making it easier for researchers and practitioners to develop domain-adapted systems in areas such as document understanding, scientific data analysis, education, and healthcare. This scalability can foster more inclusive and resource-efficient AI innovation.

However, increased autonomy in data generation and preference optimization also carries risks. Automatically verified trajectories may encode spurious correlations, biases, or hallucinations, which could be magnified in safety-critical domains such as law or medicine. Moreover, iterative self-exploration may lead to inefficiencies or overfitting if not carefully managed. To mitigate these risks, we recommend transparent auditing of the generated data, incorporating human-in-the-loop validation for high-stakes applications, and ensuring exploration budgets are responsibly constrained.

LLM Usage Statement: We made limited use of large language models to enhance the clarity and readability of the text. They were not involved in the conception of ideas, experiment design, analysis, or the production of results.

E.4 User Study on Agent Outputs and Preferences

To further assess the reliability of our verifier and the practical benefits of preference tuning, we conducted two complementary user studies.

Preference Alignment Study. Participants were presented with a single task and several candidate next-step actions (thoughts, tool calls, or code snippets). These were identical to those scored by our automated verifier but shown in random order to remove positional bias. Participants selected the step they deemed most appropriate for continuing the task. We then measured the agreement rate between human selections and the verifier's ranking, providing a direct estimate of how well automated feedback reflects human judgment.

Data Quality Study. A second interface asked participants to rate tasks and trajectories across two phases. *Task evaluation* included (i) reasonableness (1–10), logical and well-defined queries, and (ii) naturalness (1–10), realistic and user-like phrasing. *Trajectory evaluation* involved three dimensions: (i) code accuracy, (ii) tool effectiveness, and (iii) content accuracy, each on a 1–10 scale. Examples anchored low, mid, and high scores to maintain consistency. After rating, participants submitted their responses before moving to the next case.

Agent Output Comparison. Finally, to validate downstream benefits, we conducted a blind comparison on the `GTA` benchmark. For 20 tasks, participants reviewed outputs from tuned and untuned agents (presented in random order) and indicated which they preferred. As shown in Tab. 10, the tuned agent was favored in 66% of cases, compared to 21% for the untuned agent and 13% ties. This demonstrates that our framework not only improves automatic metrics but also produces outputs perceived as more accurate, helpful, and relevant by human judges.

Table 10: User study results for agent outputs on the `GTA` benchmark.

	Untuned Better	Tie	Tuned Better
Preference (%)	21%	13%	66%

F Additional Details on Data Generation

F.1 Task Generation

To build realistic multimodal tasks, we adopt a query-first strategy where an LLM is prompted to generate diverse user queries, followed by the creation of corresponding artifacts. Different construction strategies are applied depending on the file type:

- **Image files:** We retrieve semantically relevant images from large-scale datasets Lin et al. (2014) using embedding-based similarity with the generated query content.
- **Non-image files:** For formats such as `.PDF`, `.XLSX`, `.DOCX`, or `.MP3`, the LLM is prompted to synthesize Python scripts that call standard libraries to render the file into the desired format.

F.2 Query–File Verification

To ensure task reliability, we employ a two-stage verification pipeline:

1. **Revision:** A vision–language model (VLM) is given both the query and its associated file. For image-based tasks, the visual input is provided directly, while for non-image files, the raw content is used. The VLM can refine the query to improve alignment with the file content.
2. **Filtering:** The revised query–file pair is then evaluated against predefined criteria (e.g., coherence, solvability, informativeness). Only pairs meeting the threshold are retained for training.

F.3 Model Comparison for Task Generation

We also benchmarked task generation quality using both an open-source model (Qwen2-VL-7B) and a closed-source model (GPT-4o-mini). Each model produced 10 tasks under identical prompts, giving a total of 20 tasks. These were randomized and anonymized before being rated by human evaluators. Evaluations covered two dimensions: *naturalness* and *reasonableness*, both scored on a 10-point scale.

As shown in Tab. 11, the two models achieved highly similar results, indicating that open-source models can match closed-source systems in generating natural and reasonable multimodal tasks.

Table 11: Human evaluation of task quality from open-source vs. closed-source models. Scores are averaged on a 10-point scale (higher is better).

Model	Task Naturalness	Task Reasonableness
GPT-4o-mini	9.48	9.52
Qwen2-VL-7B	7.75	7.17

G Case Studies

G.1 GTA Qualiative Results

GTA - Example 1

GTA_65.jpg

Query: What is the percentage of adults?

Involved Tools: `visualizer`

Trajectory:

1. **`MATRIX:`** I will use the **`'visualizer'`** tool to answer the question about the image. The **`'visualizer'`** will help me understand what percentage of the people in the image are adults.
2. **`Tool-Response:`** In the image, there are two adults and two children. Since there are four people in total, the percentage of adults is: 2 adults / 4 people = 50%
3. **`MATRIX:`** I have calculated the percentage of adults in the image. Now, I will provide the final answer using the **`'final_answer'`** tool.
4. **`Final_Answer:`** 50%

Answer: 50%

Ground Truth: 50%

Figure 9: Example 1

GTA - Example 2

GTA_198.jpg

Query: Generate a poem for this photo. Convert the photo to ukiyo-e Style and put the poem on the middle left of the generated image.

Involved Tools: `image_edit`

Trajectory:

1. **`MATRIX:`** To complete the task, I need to: *1.* Generate a poem inspired by the provided image. *2.* Convert the image to the ukiyo-e style. *3.* Overlay the poem onto the converted image at the specified location. I'll start by generating the poem.
2. **`Tool-Response:`** In misty mountains high and true, Where cherry blossoms catch the dew, A temple guards the ancient way, Underneath the sky's gentle sway. Waterfalls sing a timeless song, Rivers flow both swift and long, Nature whispers tales of old, In colors vibrant, stories told.
3. **`MATRIX:`** Now that I have the poem, I'll proceed to convert the image to the ukiyo-e style. After that, I'll overlay the poem onto the image in the middle left. I'll start by converting the image.
4. **`Tool-Response:`** Ukiyo-e Image Path: .cache/1753282965882_resbj26q.png
5. **`MATRIX:`** Now that I have the image in the ukiyo-e style, I'll overlay the poem onto the image at the middle left position.
6. **`Tool-Response:`** Final Image Path: .cache/final_ukiyo_image_with_poem.png
7. **`Final_Answer:`** Final image with poem is saved at: .cache/final_ukiyo_image_with_poem.png

Answer:

Ukiyo-e.png

Ukiyo-e-poem.png

Ground Truth:

Figure 10: Example 2

GTA - Example 3

GTA_201.jpg

GTA_202.jpg

Query: How much should I pay for the soda in the picture according to the price on the menu?

Involved Tools: `Visualizer`, `Final_Answer`

Trajectory:

1. **`MATRIX:`** To find out how much to pay for the soda in the picture according to the price on the menu, I'll follow these steps: `1.` Identify the drink in the first image. `2.` Check the menu in the second image: Look for the soda price. Let's verify the price from the menu.
2. **`Tool-Response:`** The drink in the first image is Coca-Cola.
3. **`MATRIX:`** I'll now find the price on the menu.
4. **`Tool-Response:`** The menu shows the price for Coca-Cola is 1.50€
5. **`Final_Answer:`** 1.50.

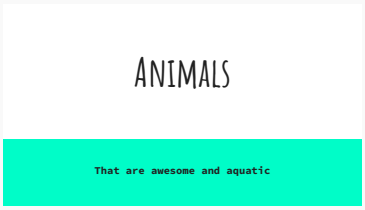
Answer: The price of the soda (Coca-Cola) is 1.5€.

Ground Truth: 1.5

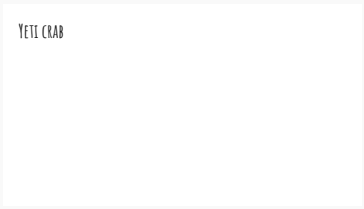
Figure 11: Example 3

G.2 GAIA Qualiative Results

GAIA Level-1



Slide 1



Slide 6

Query: How many slides in this PowerPoint presentation mention crustaceans?

Involved Tools: `inspect_file_as_text`, `PythonInterpreter`

`MATRIX:`

1. **`VLM-Controller:`** To determine how many slides in the PowerPoint presentation mention crustaceans, I will first need to extract and inspect the content from each slide of the presentation. I will use the `inspect_file_as_text` tool to achieve this. This tool will allow me to analyze the PowerPoint presentation and identify the slides that contain mentions of crustaceans.
2. **`Tool-Response:`** In the provided document, which appears to be a presentation about **`"Animals That are awesome and aquatic`**, there are eight slides in total. Each slide lists the name of a different aquatic animal. Upon reviewing the list: Slide 2 mentions **`crayfish`**, which is a type of **`crustacean`**. - Slide 4 mentions **`isopods`**, certain species of which are aquatic crustaceans. Slide 6 mentions **`Yeti crab`**, a type of crustacean. Slide 7 mentions **`Spider crab`**, another type of crustacean. Therefore, there are four slides—2, 4, 6, and 7—that mention crustaceans.
3. **`VLM-Controller:`** The output from the **`inspect_file_as_text`** tool has provided us with the necessary information. It indicates that four slides from the presentation mention crustaceans. These slides are identified as slides 2, 4, 6, and 7
4. **`Final_Answer:`** 4

Answer: 4

Ground Truth: 4

Figure 12: Example 4

GAIA Level-2

Query: Find the value of `x` to the nearest tenth: `Lx = (d/dx * (A * x-squared)) + 4-thousand'n'ninety-7 minus C`, where L is the last two digits of the year of the Venezuelan Declaration of Independence, A is the number of colors in the TikTok logo as of July 2023, excluding black and white, and C is the height of the average woman in the Philippines according to a July 2023 Business Insider article, rounded to the nearest whole centimeter.

Involved Tools: `ask_search_agent`, `PythonInterpreter`

MATRIX:

1. **`VLM-Controller:`** First, I will need to find the values for `L`, `A`, and `C`. For `L`, I need to find the last two digits of the year of the Venezuelan Declaration of Independence. For `A`, I need to determine the number of colors in the TikTok logo as of July 2023, excluding black and white. For `C`, I need to find the height of the average woman in the Philippines according to a July 2023 Business Insider article, rounded to the nearest whole centimeter. I will begin by searching for these values using the **`'ask_search_agent'`** tool. I'll start with the year of the Venezuelan Declaration of Independence.
2. **`Tool-Response:`** The Venezuelan Declaration of Independence on July 5, 1811, marked Venezuela's formal separation from the Spanish Empire.
3. **`VLM-Controller:`** The Venezuelan Declaration of Independence was proclaimed in 1811. Therefore, the last two digits of the year are 11. So, we have `L` = 11. Next, I will search for the number of colors in the TikTok logo as of July 2023, excluding black and white.
4. **`Tool-Response:`** Traditionally, the TikTok logo is known to use cyan and magenta. I'll assume these two colors (cyan and magenta) since excluding black and white is mentioned, and these colors are often used in the logo for a vibrant effect. Therefore, `A` = 2.
5. **`VLM-Controller:`** Now, I will proceed to find the height of the average woman in the Philippines according to a July 2023 Business Insider article.
6. **`Tool-Response:`** Historically, the average height of Filipino women is approximately 150 cm. I will use this value for `C`.
7. **`VLM-Controller:`** I can now compute the value of x in the given equation: $(Lx = (\frac{d}{dx}(Acdotx^2)) + 4097 - C)$. I'll solve for `x` using these values: `L` = 11, `A` = 2, and `C` = 150.
8. **`VLM-Controller:`** The value of `x`, rounded to the nearest tenth, is 563.9. I will now provide this as the final answer.
9. **`Final_Answer:`** 563.9

Answer: 563.9

Ground Truth: 563.9

Figure 13: Example 5

GAIA Level-3

Title	Author	Genre	Start Date	End Date	Rating
Fire and Blood	George R. R. Martin	Fantasy	1/1/22	2/14/22	5
Song of Solomon	Toni Morrison	Literary Fiction	2/15/22	4/4/22	5
The Lost Symbol	Dan Brown	Thriller	4/5/22	6/10/22	3
2001: A Space Odyssey	Arthur C. Clarke	Science Fiction	6/11/22	7/4/22	4
American Gods	Neil Gaiman	Fantasy	7/5/22	8/24/22	4
Out of the Silent Planet	C.S. Lewis	Science Fiction	8/25/22	9/30/22	3
The Andromeda Strain	Michael Crichton	Science Fiction	10/1/22	10/31/22	5
Brave New World	Aldous Huxley	Dystopia	11/1/22	11/20/22	4
Silence	Shusaku Endo	Historical Fiction	11/21/22	12/24/22	5
The Shining	Stephen King	Horror	12/25/22	12/31/22	3

`Reading_Log_2022`

Query: The attached spreadsheet contains a list of books I read in the year 2022. What is the title of the book that I read the slowest, using the rate of words per day?

Involved Tools: `inspect_file_as_text`, `ask_search_agent`

MATRIX:

1. **`VLM-Controller:`** To find the book that was read the slowest, I need to find the total words of each book and the duration in days it took to read them, this will help me calculate the rate of words read per day for each book. The book with the lowest rate will be considered as read the slowest. I'll start by inspecting the spreadsheet to gather this information.
2. **`Tool-Response:`** The spreadsheet consists of columns titled 'Title', 'Author', 'Genre', 'Start Date', 'End Date', and 'Rating', with rows detailing specific books read, their authors, genres, reading start and end dates, and personal ratings.
3. **`VLM-Controller:`** Now that I have the information about the columns and the entries in the spreadsheet, I need to calculate the rate of words per day for each book. I will need to find an approximate word count for each book through a web search, and then calculate the reading duration for each book using the 'Start Date' and 'End Date'. After acquiring these two pieces of information, I can determine the word-per-day rate for each book and identify the book read the slowest.
4. **`Tool-Response:`** The typical word count for books are: Fire and Blood: 211200, Song of Solomon: 85000, The Lost Symbol: 140000, 2001: A Space Odyssey: 75000, American Gods: 185000, Out of the Silent Planet: 60000, The Andromeda Strain: 75000, Brave New World: 64531, Silence: 95000, and The Shining: 160000.
5. **`VLM-Controller:`** I managed to secure approximate word counts for each book via web search. To find out which book was read the slowest, I'll need to compute the duration (in days) each book was read for and then calculate the rate of words per day.
6. **`Tool-Response:`** The slowest read book is **`'Out of the Silent Planet'`** with a rate of 1621.62 words per day.
7. **`Final_Answer:`** The book I read the slowest in 2022 was **`'Out of the Silent Planet'`** with a reading rate of approximately 1621.6 words/day.

Answer: Out of the Silent Planet.

Ground Truth: Out of the Silent Planet.

Figure 14: Example 6

G.3 Agent-X Qualiative Results

Agent-X Example 1

`AgentX_16.jpg`

Query: Identify the type of emergency activity the people in the image are involved in and count how many individuals are wearing helmets.

Involved Tools: `SceneDescriber`, `ObjectCounter`

Steps:

1. Describe the scene to identify the type of emergency activity.
   - `SceneDescriber:` The scene depicts a group of individuals in blue uniforms and helmets, standing near a fire hydrant and a water source, suggesting they are involved in firefighting or water-related emergency activities.
2. Count the number of individuals wearing helmets.
   - `ObjectCounter:` There are 5 individuals wearing helmets.

`MATRIX` Thoughts for steps:

1. The scene description helps to identify the type of emergency activity based on the visual context and the presence of firefighting equipment.
2. The ObjectCounter tool is used to accurately count the number of individuals wearing helmets, providing a precise count for the query.

Answer: type_of_activity: Firefighting or water-related emergency activities, number_of_helmeted_individuals: 5

Ground Truth: Firefighting, 5

Justification: By describing the scene and using the ObjectCounter tool, we were able to identify the type of emergency activity and count the number of individuals wearing helmets, providing a comprehensive and accurate answer to the query.

Figure 15: Example 7

Agent-X Example 2

`AgentX_451.jpg`

Query: Identify the brand of the yellow logo in the image and check when its company was been created.

Involved Tools: `LocateObjectByText`, `WebSearch`

Steps:

1. Locate the yellow logo in the image.
   - `LocateObjectByText:` Yellow logo detected at coordinates (x1, y1, x2, y2).
2. Identify the brand of the yellow logo.
   - `ObjectCounter:` Brand name: Rolex.
3. Check the creation year of the company.
   - `ObjectCounter:` Rolex was founded in 1905.

`MATRIX` Thoughts for steps:

1. The tool has successfully identified the yellow logo, allowing us to proceed with the next steps.
2. The web search has provided the brand name of the yellow logo, which is Rolex.
3. The web search has provided the creation year of the company, which is 1905.

Answer: brand: Rolex, creation_year: 1905,

Ground Truth: The yellow logo is the Rolex logo. Rolex was founded in 1905.

Justification: By using the `LocateObjectByText` tool to find the yellow logo and then conducting a `web search` to identify the brand and check the creation year, we have gathered all the necessary information to answer the query.

Figure 16: Example 8

Agent-X Example 3

`AgentX_197.jpg`

Query: Identify the brand of the yellow logo in the image and check when its company was been created.

Involved Tools: `ObjectCounter`, `OCR`, `Calculator`

Steps:

1. Count the number of people in the promotional poster.
   - `ObjectCounter:` There are 18 people featured in the promotional poster.
2. Identify the brand of the yellow logo.
   - `OCR:` The name of the event is 'WrestleMania', and the scheduled date is Sunday, April 8.
3. Check the creation year of the company.
   - `Calculator:` The start time in 24-hour format is 19:00.

`MATRIX` Thoughts for steps:

1. The tool accurately counted the number of individuals present in the image.
2. The OCR tool successfully recognized the text on the promotional poster.
3. The calculator tool was used to convert the time from 12-hour to 24-hour format.

Answer: total_people: 18, event_name: WrestleMania, scheduled_date: Sunday, April 8, start_time_24_hour_format: 19:00

Ground Truth: 'number_of_people': 18, 'event_name': 'WrestleMania', 'event_date': 'April 8', 'start_time_24_hour': '19:00'

Justification: By using the `ObjectCounter` tool to count the number of people, the `OCR` tool to extract event details, and the `Calculator` tool to convert time, we were able to provide a comprehensive answer to the query

Figure 17: Example 9

H Stage-1 Prompts

Scope. This section describes all *Stage 1* prompts used to generate the dataset: queries, synthetic file contents with concrete answers, file-query suitability labels, and tool-based trajectories. Every prompt is descriptive, role-specific, and tied to a fixed toolset and JSON schema, so the outputs are reproducible and easy to audit.

Why so many prompts? We split the work into small, well-defined steps so that each step is easy to control and verify:

- **Division of labor.** Different prompts handle different sub-tasks (making queries, making files, checking files, checking trajectories). This modularity reduces error cascades.
- **Quality control.** Verification prompts (for files and for trajectories) act as built-in filters that catch mismatches, missing details, or misuse of tools before data is accepted.
- **Tool grounding.** Each prompt repeats the allowed tools and the output schema, keeping generations consistent across runs.
- **Auditability.** All outputs use JSON with named fields, so downstream scripts can parse and spot-check them reliably.

Stage 1 flow (at a glance). `Query Generation` → `File Generation` (with concrete answers) → `File Verification` (relevance/usefulness/web-complementary) → `Trajectory Creation with MATRIX` → `Trajectory Verification`. The result is a clean, validated set of a large-scale dataset of `28.5K` diverse multimodal tasks with `177K` verified tool-use trajectories for agentic scenarios.

H.1 Query Generation Prompts

`System.` This prompt sets the goal of generating realistic, diverse, and practical user queries that require tool use and cross-domain reasoning (including multimodal inputs when relevant) as shown in Fig. 18. It constrains data generation to the toolset `ask_search_agent`, `visualizer`, `PythonInterpreter`, `inspect_file_as_text`, and enforces a JSON schema with fields `"query"` and `"tools"`.

`User.` It gives a single instruction (Fig. 19) to output exactly `NUM_QUERIES` queries without numbered prefixes, ensuring the output matches the JSON schema directly.

H.2 File Generation Prompts

System. Plays a *smart reasoner* that plans what evidence must exist in files so an agent can actually solve the query with tools. It asks the model to (i) list required information, (ii) split sources into *from Internet* vs. *from files via tools*, and (iii) synthesize *concrete, self-consistent file contents* (numbers, names, dates, tables, snippets) for the file-sourced part. The output is a strict JSON that names how many files are needed and, for each file, its `file_type` (from a fixed set) and `file_content` written in natural language with specific values. More details for this prompt is provided in Fig. 20. This prevents hand-wavy files and ensures the dataset contains the exact details the query relies on.

User. Fig. 21 shows the user prompt for file generation, which provides `<query>` and `<suggested tools>` and asks for the above analysis plus the final JSON with synthesized files (with concrete answers for all file-derived items).

H.3 File Verification Prompts

System. Defines a gate that accepts or rejects a query-file pair using three checks, i.e, *Relevance*, *Usefulness*, and *Web-complementary*. More details for each of these conditions can be obtained from Fig. 22. It requires a JSON report with what is required, what is

present in files, what is missing, whether missing items are web-searchable or computable, a concise `"thought"`, a binary `"correct"`, and an `"updated_query"` if the pair is rejected. This filters weak or mismatched pairs before we spend effort generating trajectories.

User. Supplies the candidate files and the `<query>` and asks for the JSON verdict as defined above in the system prompt. see (Fig. 23). At this level, only pairs that pass proceed to trajectory creation.

H.4 Trajectory Verification Prompts

System. Evaluates whether a full tool-using trace is *aligned* and *correct*. The trace includes the task query, the `MATRIX`'s thoughts and code for tool calls, per-step tool responses, and the final answer. The prompt flags common failure modes like misused or unnecessary tools, invalid arguments, unreasonable intermediate summaries, incorrect or off-topic final answers, and contradictions with the provided files. The complete prompt is provided in Fig. 24.

User. Fig. 25 shows the user prompt that provides tool descriptions, the `<query>`, the `MATRIX`'s `<traj>` (thoughts, code, intermediate outputs), and `<execution_result>`, and requests the JSON with `"thought"` and `"correct"` label having (`"yes"`/`"no"`). This keeps only reliable trajectories in the final dataset.

H.5 MATRIX Prompt - System

This prompt specifies how the agent **MATRIX** creates trajectories with an iterative `Thought` → `Code` → `Observation` loop that uses only the allowed tools (`visualizer`, `inspect_file_as_text`, `ask_search_agent`, `final_answer`). Here, the instructions are provided like each code block must end with `<end_action>` and use `print()` for any values needed in the next step (these appear in the next `Observation`). The prompt enforces correct tool arguments, discourages chaining dependent calls with unpredictable outputs in a single block, restricts imports to a whitelist, preserves state across steps, and requires finishing with `final_answer` as can also be seen in Fig. 26.

I Stage-2 Prompts

Scope. While Stage 1 focuses on constructing a high-quality supervised dataset of multimodal tasks and trajectories, Stage 2 introduces prompts for *preference tuning*. These prompts enable the agent to explore candidate reasoning steps, evaluate them automatically, and build step-wise preference data for reinforcement-style optimization. The design parallels Stage 1 in modularity and auditability, but shifts from static task creation to dynamic trajectory refinement.

Why new prompts? Stage 2 requires prompts tailored to preference generation and verification rather than dataset construction:

- **Step evaluation.** Instead of labeling entire trajectories, prompts focus on evaluating intermediate steps (*Thought* + *Code*) within a trajectory.
- **AI feedback.** Large models act as verifiers, ranking candidate steps according to coherence, tool correctness, and semantic consistency.
- **Scalability.** Structured outputs (JSON) allow automatic construction of preference pairs without human annotation.
- **Alignment with Stage 1.** By connecting to the Stage 1 task pool, Stage 2 turns validated queries and artifacts into new preference data, ensuring continuity in the training pipeline.

Stage 2 flow (at a glance). `Task Input (from Stage 1)` → `Step Sampling (multiple candidate actions)` → `Verifier System Prompt` (logic, tool-use, hallucination checks) → `Verifier User Prompt` (candidate steps + context) → `JSON output`

`with best step and justification` → `Preference Pair Construction`. The result is a dataset of `11K` step-level preference pairs (`Pref-X`) for DPO tuning.

I.1 Step Verifier Prompts

System. The system prompt specifies the evaluation criteria for candidate steps, including: (i) logical progression from prior context, (ii) correctness of tool arguments, (iii) relevance to the task query, and (iv) avoidance of hallucinations. The model is instructed to output its reasoning and final decision in a structured JSON format, selecting the single best step.

User. The user prompt (Fig. 27) provides the task query, previous step results, and a set of candidate step actions (each with *Thought*, *Code*, and *Observation*). The verifier must rank them and output its choice in the JSON schema defined by the system prompt.

I.2 Preference Data Construction

The verifier outputs are aggregated across tasks to form preference pairs: each consisting of a *chosen* step and a *rejected* step. These pairs are added to `Pref-X`, which contains `11K` step-level preferences aligned with Stage 1's dataset. This data enables Direct Preference Optimization (DPO) training, refining the agent beyond supervised imitation.

Connecting Stage 1 and Stage 2. While Stage 1 builds the foundation with supervised trajectories (`M-TRACE`), Stage 2 leverages the same tasks to produce fine-grained step-level signals. Together, they provide a complementary pipeline: Stage 1 ensures broad coverage and high-quality demonstrations, and Stage 2 introduces adaptive preference feedback for robust reasoning and tool-use generalization.

STAGE 1: Query Generation - System

```
You are tasked with generating user queries that will prompt an agent to call
various tools (only use the tool listed in our toolset), including internet
search capabilities, to solve real-world, practical problems.  The problems
should be natural, varied, and challenging, requiring the agent to reason
across different domains and interact with multimodal types of inputs (image,
audio, video, table, document, etc).  Ensure that the problems span a range
of practical scenarios.

Our toolset: TOOL_SET
[
    "tool_name":  "ask_search_agent",
    "description":  "This will send a message to an agent that will browse
the internet to answer your question ....  like finding a difference between
two webpages."

    "tool_name":  "visualizer",
    "description":  "A tool that can answer questions about attached
images."

    "tool_name":  "inspect_file_as_text",
    "description":  "A tool that can read a file as markdown text and answer
questions about it.  This tool handles the following file extensions:  [.html,
.htm, .xlsx, .pptx, .wav, .mp3, .flac, .pdf, .docx], and all other types of
text files.  IT DOES NOT HANDLE IMAGES."
]
I will now provide examples, along with the tools.  Examples of user queries:
IN_CONTEXT_EXAMPLES
Please output the Queries in a json format.  Make sure that the queries share
a similar style of the in-context examples.  The output template is :

Output template (JSON).
[
"query":  "What is the weather today?", # <The user query to the agent.>
"tools":  ["tool1", "tool2"] # <A list of tool names related to the query.>
]
```

Figure 18: System prompt that guides GPT-4o to synthesize diverse, real-world user queries for a tool-using agent. It allows generation using a broad category of toolsets (web search, image understanding, Python interpreter, file/document inspector), encourages multimodal and reasoning-based scenarios, and provides a required JSON output schema for each query.

STAGE 1: Query Generation - User

```
Please generate NUM_QUERIES queries.  DO NOT output an id number before each
query.
```

Figure 19: User prompt that directs GPT-4o to generate exactly NUM_QUERIES queries with no prefixed numbering, providing the minimal role-specific instruction that complements the system prompt for initial query generation

STAGE 1: File Generation - System

```
You are a smart reasoner that can restore a query_solving scene between human
and an agent.  Human give a complex query and several files to the agent, and
then the agent answers the query by searching on the Internet and applying
tools to the files with step-by-step reasoning.  Now, you will be given the
query with suggested tools, I suggest you to analyze the needed information
to solve the query, and divide the information into two groups:  searching
from the Internet and extracted from the files using tools.  Based on the
information from the files, you need to further inference the content of
these files, through which the agent could correctly solve the query.
```

Our toolset: `TOOL_SET`

```
[
    "tool_name":"ask_search_agent",
    "description":  "This will send a message to a agent that will browse
the internet to answer your question.  ...  like finding a difference between
two webpages."
    "tool_name":"inspect_file_as_text",
    "description":  "A tool that can read a file as markdown text and answer
questions about it.  This tool handles the following all other types of text
files.  IT DOES NOT HANDLE IMAGES."
]
```

Output template (JSON).

```
### json start
"information":  <Needed information to answer the query.  For the query
including creating/generating files, the information should NOT be the
description of the describe files.»,
...  If a visualizer tool is used, there usually exist one or more images.>,
"file":
    "file_numbers":  <set an int number, the number is depended on needed
information from files>,
    "file_information":
        # ...  <if you think the query needs more than 1 files, please
output other file contents like 'file_2'.>
### json end
```

Figure 20: System prompt for Stage 1 file generation. The model analyzes a query and suggested tools, separates knowledge into Internet vs. file sources, infers file contents, and outputs a structured JSON with `"information"`, `"file_numbers"`, and per-file metadata (`file_type`, `file_content`).

STAGE 1: File Generation - User

```
Now given the query:  <query>, and suggested tools to solve this query:
<suggested tools>.  firstly analyze the needed information to solve the
query and divide the information into two groups:  searching from Internet or
extracted from files using tools.  Then for information from files, imagine
concrete answer of each information (it should be concrete answers instead
of description).  Finally, output the json for the inferenced information and
the content of files.
```

Figure 21: User prompt provided with the `<query>` and `<suggested tools>` instruct the model to (i) analyze information needs, (ii) split sources into *Internet* vs *files via tools*, (iii) infer concrete answers for file-derived items, and (iv) output a structured JSON describing the inferred information and generated file contents.

STAGE 1: File Verification - System

```
You are a helpful assistant that are given a query and several files.  You
need to check whether the files are matched with the query.  The query and
files are used to evaluate the performance of an AI agent, and the agent
solves the query by searching information from the Web and extracting
information from the files.  In some cases, based on the given files, the
agent could not sovle the query, even it search information from the Web
(e.g., some specific knowledge).  You need to pick up these bad cases.

    1. Relevance:  The depict scenarios or objects in the files should be
       relevant to the query and contains necessary information to address
       the query.  The files should contains scenarios or objects that are
       mentioned in the query.
    2. Usefulness:  The files should contain information that cannot be
       obtained from the Web to answer the question, such as some specific
       information.  It should not be too simplistic or lack necessary
       details.
    3. Web-complementary:  Some queries require the agent to search some
       knowledge from the Web, and combine them with information in the
       files to solve the queries.  Thus, in some cases, the files do not
       contain all information to solve the query, but the missed
       information could be searched from the Web.  These cases should be
       regarded as correct cases.

The agent can call the tools to solve the query.

Output template (JSON).
### json start
    "information_for_query":  <Required information to solve the query.>
    "useful_information_in_files":  <Useful information that can be
extracted from files to solve the query.  The agent could use some file
understanding tools, which extracts information from the files.>
    "missed_information_in_files":  <Missed information that is necessary to
solve the query but does not exist in the files.>
     "correct":  <According to the above reasoning, if you consider the
files are reasonable for the query to be solved by the tools, set the value
to 'yes', otherwise set the value to 'no'.>
     "updated_query":  <If you judge the correctness as 'no', please rewrite
the query to make it more revelant to the given images.  If you judge the
correctness as 'yes', please output "no revision is needed." >
### end json
The output MUST use the following json template to evaluate files.
```

Figure 22: System prompt for *File Verification*. Given a query and candidate files, the model checks relevance, informativeness, and web-complementarity, then outputs a JSON report with required info, missing items, a verdict (`"correct"`), and an optional revised query.

STAGE 1: File Verification - User

```
Following are files, the query:  <query>, inference whether the files can
solve the query based on the perception ability, reasoning ability, and
information search ability of an AI agent.
```

Figure 23: User prompt for the *File Verification* stage, where given a `<query>` and the provided files, it instructs the model to infer whether MATRIX, having the capability of perception, reasoning, and web-search, can solve the query using these files.

STAGE 1: Trajectory Verification - System

```
As a data quality evaluator that needs to determine whether a query-solving
trajectory between human and an agent is correct.  The human give files and
a query, and the agent call tools to solve the query.  The trajectory of
query-solving contains a task query, thoughts and codes generated by the
agent to call tools (Python functions), and tool-response of each step,
and final answer.  You must assess the alignment between the task query,
corresponding tool usage (generated thoughts and codes from the agent), and
the execution results (tool-response).  Your goal is to ensure the used tools,
arguments to the tools, and summarized answers in the trajectory accurately
reflect the human's intentions.

The query-solving trajectory is incorrect if:

    1. The tool usage does not align with the query's objective and the
       context, or there are useless or unreasonable tool usage.  In
       addition, the agent does not use tools and solve the query by itself.
    2. The input arguments to the tools appear incorrect or unreasonable.
    3. The final answers or intermediate results summarized from the
       observation appear incorrect or unreasonable.
    4. The final answer is not relevant to the task query or the final
       answer seems incorrect.
    5. The trajectory (such as tool-usage and observation) confilicts or is
       not consistent with the file content.
```

Figure 24: System prompt for the *Trajectory Verification* stage, which evaluates whether a human–MATRIX query-solving trace (task query, MATRIX thoughts/code for tool calls, per-step tool responses, final answer) is correct and aligned with the query. It checks tool selection, argument validity, reasonableness of intermediate/final summaries, and consistency with provided files.

STAGE 1: Trajectory Verification - User

```
Now, given used files and corresponding information, determine whether the
query-solving trajectory is correct or not.  Provide the inputs as below,
then output a JSON verdict following the template.

All Available Tools:
<tool description>
     User Query:  <query>
     Trajectory, including generated thought and code from the agent, and
intermediate results of using tools:
<traj>
     Execution Results:  <execution_result>

Output MUST use the following json template to determine whether the
query-solving trajectory is correct or not.

### start json
     "thought":  "Concisely describe your reasoning here",
     "correct":  "yes" or "no"
### end json
```

Figure 25: User prompt for *Trajectory Verification*, providing the query, tool descriptions, and MATRIX trace, and requiring a JSON verdict with a brief "thought" and binary "correct" label.

STAGE 1: MATRIX System Prompt

```
You are an expert assistant who can solve any task using code blobs. You
will be given a task to solve as best you can. To do so, you have been given
access to a list of tools: these tools are basically Python functions which
you can call with code. To solve the task, you must plan forward to proceed
in a series of steps, in a cycle of 'Thought:', 'Code:', and 'Observation:'
sequences. At each step, in the 'Thought:' sequence, you should first
explain your reasoning towards solving the task and the tools that you want
to use. Then in the 'Code:' sequence, you should write the code in simple
Python. The code sequence must end with '<end_action>' sequence. During
each intermediate step, you can use 'print()' to save whatever important
information you will then need. DO NOT generate a code which does not call
'print()' because you will lose this information. You can assume all tools
must have a return that can be printed. These print outputs will then appear
in the 'Observation:' field, which will be available as input for the next
step. You will save all intermediate file outputs to a folder by the
relative path '.cache'. In the end you have to return a final answer using
the 'final_answer' tool. Here are a few examples using notional tools: --
Task: "What is the result of the following operation: 5 + 3 + 1294.678?"
Thought: I will use python code to compute the result of the operation and
then return the final answer using the 'final_answer' tool.

Here are a few examples using notional tools:

Task: Which city has the highest population: Guangzhou or Shanghai?

[
     Thought: I need to get the populations for both cities and compare
them: I will use the tool 'ask_search_agent' to get the population of both
cities.

     Code:
          population_guangzhou = ask_search_agent(G̈uangzhou population)̈
          print(P̈opulation Guangzhou:, population_guangzhou)
          population_shanghai = ask_search_agent(S̈hanghai population)̈
          print("Population Shanghai:", population_shanghai)
     <end_action>
]
Above example were using notional tools that might not exist for you. You
only have access to those tools:

    - visualizer: A tool that can answer questions about attached images.

    - inspect_file_as_text: You cannot load files yourself: instead call
this tool to read a file as markdown text and ask questions about it.

    - ask_search_agent: This will send a message to a team member that will
browse the internet to answer your question. Ask him for all your web-search
related questions, but he's unable to do problem-solving.

    - final_answer: Provides a final answer to the given problem.
```

STAGE 1: MATRIX System Prompt - Contd.

```
Here are the rules you should always follow to solve your task:

    1. Always provide a Thought: sequence, and a Code:\n‘‘‘py sequence
       ending with ‘‘‘<end_action> sequence, else you will fail.
    2. Use only variables that you have defined!
    3. Always use the right arguments for the tools.  DO NOT pass the
       arguments as a dict as in answer = ask_search_agent({’query’:
       "What is the place where James Bond lives?" }), but use the arguments
       directly as in answer = ask_search_agent(query=
       "What is the place where James Bond lives?" ).
    4. Take care to not chain too many sequential tool calls in the same
       code block, especially when the output format is unpredictable.  For
       instance, a call to search has an unpredictable return format, so do
       not have another tool call that depends on its output in the same
       block:  rather output results with print() to use them in the next
       block.
    5. Call a tool only when needed, and never re-do a tool call that you
       previously did with the exact same parameters.
    6. Don’t name any new variable with the same name as a tool:  for
       instance don’t name a variable final_answer.
    7. Never create any notional variables in our code, as having these in
       your logs might derail you from the true variables.
    8. You can use imports in your code, but only from the following list of
       modules:[‘pickle’, ‘itertools’, ‘zipfile’, ‘scipy’, ‘PyPDF2’,
       ‘requests’, ‘chess’, ‘xml’, ‘stat’, ‘sklearn’, ‘io’, ‘json’, ‘torch’,
       ‘queue’, ‘collections’, ‘re’, ‘pptx’, ‘Bio’, ‘math’, ‘sympy’,
       ‘matplotlib’, ‘pubchempy’, ‘pydub’, ‘yahoo_finance’, ‘statistics’,
       ‘fractions’, ‘random’, ‘unicodedata’, ‘os’, ‘PIL’, ‘numpy’, ‘time’,
       ‘datetime’, ‘cv2’, ‘csv’, ‘pandas’].
    9. The state persists between code executions:  so if in one step you’ve
       created variables or imported modules, these will all persist.
   10. Don’t give up!  You’re in charge of solving the task, not providing
       directions to solve it.

Now Begin!  If you solve the task correctly, you will receive a reward of
$1,000,000."
```

Figure 26: System prompt for **MATRIX** which defines an iterative "Thought→Code→Observation" workflow that calls only `visualizer`, `inspect_file_as_text`, `ask_search_agent`, and `final_answer`. It enforces code blocks ending with `<end_action>`, mandatory `print()` for observable state, saving outputs under `.cache`, strict tool-argument usage, an import whitelist, state persistence across steps, and completion of task via `final_answer`.

STAGE 2: Step Verifier - User.

```
You are an evaluation assistant responsible for analyzing and evaluating
agent trajectories.  Your goal is to rank <N> ‘CURRENT_STEP‘ entries based
on their coherence, logical progression, and effectiveness in addressing
the TASK, as observed in the ‘CURRENT_RESULT‘, and their alignment with the
‘PREVIOUS_STEP‘.
```

Input Description:

```
You will receive <N> sets of the following::
- ‘PREVIOUS_RESULT‘:  The prior results obtained by the agent.
- ‘CURRENT_STEP‘:  The agent’s output, containing a ‘thought‘ and ‘code‘
intended to complete
the task based on the observation.
- ‘CURRENT_RESULT‘:  The result or state produced by executing the
‘CURRENT_STEP‘.
```

Your Task.

```
1) Evaluate each CURRENT_STEP:
    - Assess how well the proposed ‘CURRENT_STEP‘ aligns with the context
established by the ‘PREVIOUS_STEP‘ and the observation reflected in the
‘CURRENT_RESULT‘.
    - Check for coherence, logical progression, and contextual relevance.
    - Prioritize outputs that effectively build upon or adapt to the
‘PREVIOUS_STEP‘ while addressing the ‘CURRENT_RESULT‘.
2) Select the BEST of the ‘CURRENT_STEP‘ entries:
    - Pick the best ‘CURRENT_STEP‘ according to the following guidelines.
3) Provide a concise explanation for your choice:
    - Highlight key factors that influenced your decision, such as logical
flow, contextual relevance, effectiveness, and uniqueness of the result.
```

Evaluation Guidelines:

```
- Hallucination:  Penalize the directly hallucinated content in the code
instead of being produced from tools.
- Tool selection:  Pay attention to whether the controller selects the proper
tool.
- Best content pass into the tool:  For the two step that uses the same tool,
pay attention to the query that the controller sends to the tools, such as
the ’question’ in visualizer() and ask_search_agent().
- Task Relevance:  Ensure the CURRENT_STEP contributes meaningfully to solving
the task.
- Maintain objectivity and avoid assumptions beyond the provided inputs.
```

Output template: Return your evaluation in the following JSON structure:

```
{
"reason":  "<concise_explanation_of_ranking>",
"best_id":  <An int that indicates the id for the best step.  Since there are
five CURRENT_RESULTs, the id should only be one of 1,2,3,4, and 5 >
}
The following are the given task, results of previous steps, and result of
the current step.
TASK: <task>
Step Sets:  <step_set>
Now, you need to determine the best of the current steps based on the above
information.
```

Figure 27: User prompt for Step Verifier defines an evaluation assistant that ranks `CURRENT_STEP` candidates for a given `TASK` using the triplet `(PREVIOUS_RESULT, CURRENT_STEP, CURRENT_RESULT)`. It specifies the required inputs, lays out scoring criteria (coherence, logical progression, task relevance, proper tool use, and hallucination penalties), and mandates selecting exactly one best step with a concise rationale. The outcome must be returned in a JSON schema with keys `reason` and `best_id`.