



JoyInside-VLM-Agent: Product-Conditioned Multimodal Policy Learning for Embodied Agents

Gangyi Li*, LuYao Wang*, Jingfu Sun*, Jingyi Yuan*, Hang Liu, Hao Zong, Yujia Hu, Tingting Gao^{*,†}, Jianyong Wang^{†,‡}, Xu Yang^{†,‡}

JD.COM

*Equal Contribution, [†]Project Lead, [‡]Corresponding Authors

Abstract

Embodied agents must translate multimodal observations and user requests into executable actions while adapting to product constraints, changing goals, and execution feedback. We present JoyInside-VLM-Agent, a ReAct-style multimodal agent developed through product-conditioned trajectory construction and staged policy learning for robotic platforms and smart hardware. A closed-loop sandbox generates multimodal interaction trajectories covering goal revisions and failure recovery, which are checked against execution outcomes and semantic criteria. Supervised fine-tuning and multi-teacher on-policy distillation precede reinforcement learning with DPER (Dual-Source Privileged On-Policy Self-Distillation with Token-Level Evidence Routing). DPER routes token-level evidence between a frozen privileged teacher and a privileged student sharing the rollout policy’s parameters, modulating reward-derived token advantages while preserving their signs. Training rewards combine generative process evaluation, multi-judge rankings, and task-queue consistency. JoyInside-VLM-Agent achieves a BFCL v4 overall score of 77.44, exceeding the evaluated baselines; results on the other public benchmarks are mixed. On JoyInside-Devices and JoyInside-Robotics, the evaluated system, which includes domain-specific skill refinement, leads the baselines on all reported metrics. Two real-world quadruped case studies illustrate visually grounded search, replanning after execution failure, and explicit failure reporting.

1 Introduction

Embodied agents deployed on robotic platforms and smart hardware must translate multimodal observations and user instructions into executable behavior. Across products ranging from bipedal, quadruped, and wheeled robots to AI-enabled plush toys, the appropriate response to a given request depends on the agent’s role, capabilities, and operating constraints. Changes in user goals, new visual evidence, and execution failures can further invalidate earlier plans. We study how to train a shared multimodal policy that adapts to product specifications and revises its decisions as interactions unfold. Figure 1 presents the training framework and deployment examples.

Multimodal training has demonstrated positive transfer across tasks and robotic embodiments [1, 2], while context-aware systems use user intent and environmental feedback to guide interaction [3, 4]. However, reported failures include actions beyond device capabilities and descriptions that no longer match the user’s

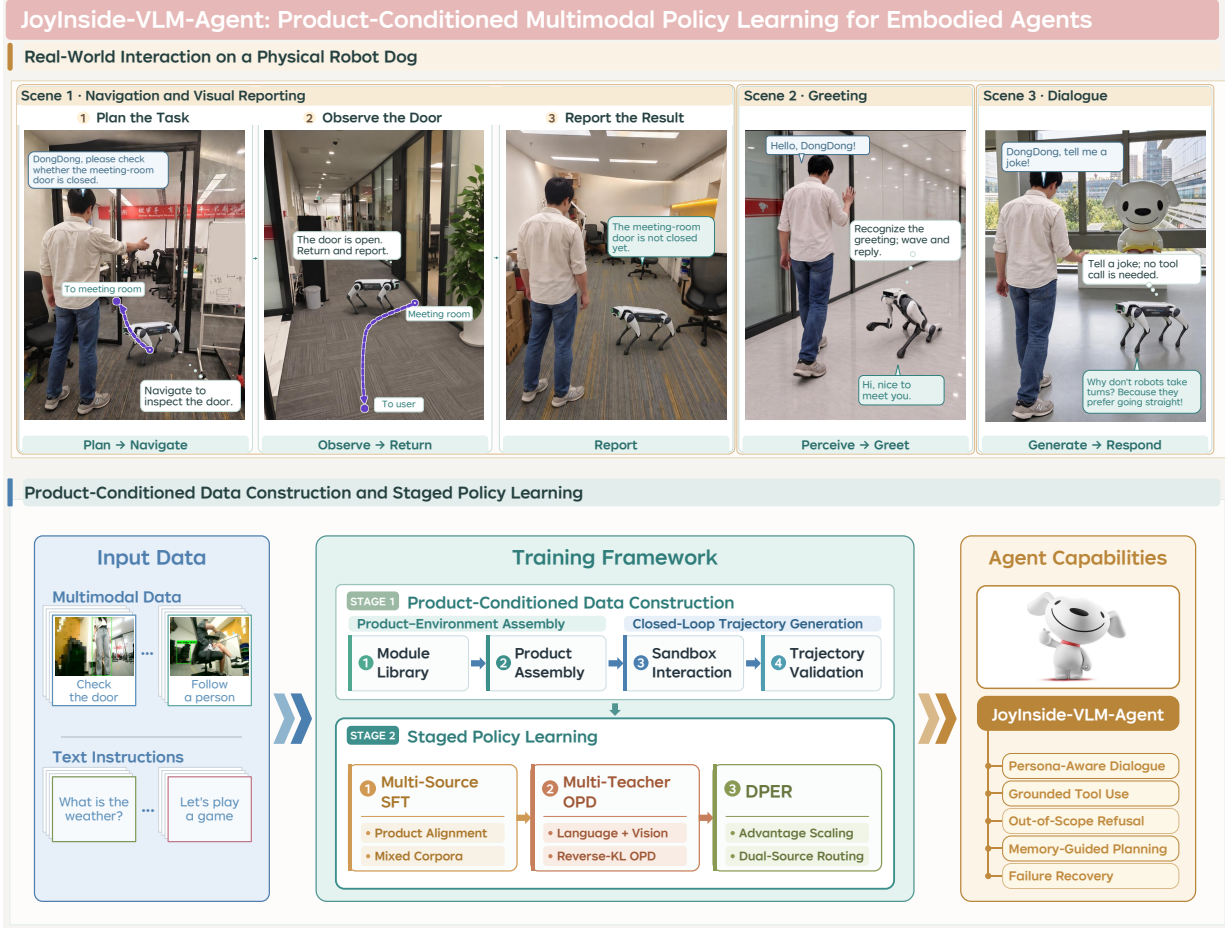


Figure 1 JoyInside-VLM-Agent overview. *Top:* Real-world quadruped interactions show navigation with visual reporting (left), greeting (center), and dialogue (right). *Bottom:* Product-conditioned data construction (Stage 1) supplies trajectories and executable environments for SFT, multi-teacher OPD, and DPER (Stage 2). The right panel summarizes agent capabilities.

current scene or intent [3, 4]. These observations motivate training policies that combine transferable capabilities with adherence to product constraints and sustained alignment with the user’s current goal.

This objective imposes two requirements on training. The data must cover diverse product configurations, goal revisions, and failure recovery while preserving the correspondence between actions and observations. The learning signal must reflect reasoning quality, compliance with product constraints, and progress toward the current goal. Trajectory-level rewards summarize these criteria but do not specify how to distribute credit across tokens. Using additional evidence during training therefore requires selecting an appropriate source for each token and controlling how that evidence affects policy updates.

We present JoyInside-VLM-Agent, a multimodal agent developed through product-conditioned data construction and staged policy learning. Our pipeline combines product roles, workflows, and constraints with available tools and skills to build executable environments. A closed-loop sandbox generates multimodal interaction trajectories, which undergo execution checks and semantic validation. Multi-source supervised fine-tuning establishes behavior consistent with product specifications, while multi-expert on-policy distillation integrates complementary language and visual capabilities. Reinforcement learning then refines the policy using interaction-specific rewards.

To guide reinforcement learning, we combine process-quality scores from a Generative Process Reward Model

(GPRM), multi-judge rankings of the same candidate trajectories, and task-queue assessments of goal alignment and cross-turn continuity. For token-level policy optimization, we introduce DPER (Dual-Source Privileged On-Policy Self-Distillation with Token-Level Evidence Routing), building on RLSD’s sign-preserving advantage modulation [5] and DOPD’s dual-source routing [6]. A frozen privileged teacher and a privileged student sharing the rollout policy’s parameters score tokens along the same student-generated trajectory. DPER selects evidence based on their disagreement and support for each sampled token. The selected evidence rescales token-level advantages while preserving their reward-derived signs. Privileged information is used only during training.

We evaluate JoyInside-VLM-Agent on public tool-use benchmarks and the proprietary JoyInside-Devices and JoyInside-Robotics datasets. The model achieves a BFCL v4 overall score of 77.44, outperforming the evaluated baselines. Case studies on a physical quadruped robot provide qualitative evidence of visually grounded search, replanning in response to execution feedback, and honest failure reporting.

Our contributions are as follows:

- **Product-conditioned trajectory construction.** We compose executable product configurations and generate verified multimodal trajectories covering product variation, goal revisions, and failure recovery.
- **DPER for policy optimization.** We route token-level evidence between two privileged sources to scale token advantages while preserving their reward-derived signs.
- **Multi-source interaction rewards.** We combine process evaluation, multi-model rankings, and task-queue consistency into a learning signal for reasoning quality, adherence to product constraints, and alignment with evolving goals.
- **JoyInside-VLM-Agent and its evaluation.** We integrate these components into a unified training framework and assess the resulting model through public benchmarks and proprietary evaluation sets and qualitative quadruped case studies.

2 Related Work

2.1 Embodied and Context-Aware Interaction

PaLM-E learns shared representations for robotic and vision-language tasks [1], while Open X-Embodiment studies cross-platform transfer using pooled interaction data [2]. Context-aware assistants address how observations and user intent guide behavior: Sasha grounds smart-home requests in available devices and user feedback [3]; GazePointAR combines gaze, pointing, and dialogue history to resolve references [7]; and WorldScribe adapts visual descriptions to intent and environmental changes [4]. Together, these lines connect shared capabilities with decisions grounded in the current interaction.

Interactive benchmarks further make behavioral constraints explicit. τ -bench evaluates conversations under domain policies [8], while τ^2 -Bench considers user-agent coordination in a shared environment [9]. WildToolBench covers intentions distributed across turns and transitions among tool use, conversation, and clarification [10]. These settings motivate evaluating whether behavior respects constraints and follows the current goal. Our work addresses the corresponding training problem across robotic platforms and smart hardware, conditioning a shared policy on each product’s role, capabilities, constraints, and interaction state.

2.2 Multimodal Trajectory Construction

Synthetic data methods expand agent training beyond available interaction logs. ToolACE constructs diverse function-calling dialogues with rule- and model-based verification [11], and T3-Agent generates and verifies multimodal queries and tool-use trajectories [12]. Their focus on coverage and validity provides a basis for learning tool interaction. Training for dynamic tasks additionally requires representing the consequences of intermediate actions and the decisions that follow unexpected feedback.

Agent-R constructs revision trajectories through search and model-guided critique, connecting erroneous behavior to successful continuations [13]. Agent World Model builds executable, stateful environments for

interaction and reinforcement learning [14]. These approaches expose learners to recovery behavior or environmental transitions. Our pipeline couples such execution feedback with compositional product specifications and visual observations, covering both recovery toward an existing goal and adaptation after the user changes it. We jointly vary product roles, workflows, constraints, and capabilities within executable environments.

2.3 Post-Training and Reward-Guided Optimization

AgentTuning and Agent-FLAN study how instruction tuning develops agent behavior from demonstrations [15, 16]. Generalized knowledge distillation obtains teacher feedback on student-generated sequences, addressing the mismatch between fixed demonstrations and the learner’s behavior [17]. H-OPD combines vision-language and text-only teachers through visual-description transfer and token-level arbitration [18]. Agents-A1 integrates domain expertise through staged training and multi-teacher on-policy distillation [19]. RL Starts before RL studies OPD as preparation for reinforcement learning, reporting higher post-RL averages than direct RL and SFT-initialized RL in text and vision-language reasoning [20]. These methods inform capability acquisition and consolidation; our training additionally conditions expert transfer on product specifications and refines the resulting policy against interaction-specific rewards.

Reward modeling determines which behavior to encourage. Process supervision evaluates intermediate reasoning [21], generative verifiers produce correctness judgments [22], and PoLL aggregates assessments from multiple models [23]. Aggregation can retain shared errors [24]; evaluation criteria must still be defined for the target behavior. Using these assessments in optimization is a separate question. RLSD uses privileged evidence to scale token advantages while preserving their reward-derived signs [5]. DOPD routes supervision between privileged teacher and student policies to address differences in token-level supervision [6]. DPER combines these mechanisms: it extends RLSD’s magnitude modulation to dual-source evidence selected through DOPD’s routing principle. Our multi-source rewards separately assess reasoning, product constraints, and goal consistency.

3 Product-Conditioned Multimodal Data Construction

We construct multimodal interaction trajectories to learn a shared policy across heterogeneous robotic platforms—including bipedal, quadruped, and wheeled robots—and smart hardware. Our construction couples compositional product specifications with executable constraints and visual feedback consistent with the resulting state. This produces interactions that cover both recovery from execution failures and adaptation to revised user goals under the same product rules. Figure 2 connects product composition and environment initialization (steps 1–2), closed-loop interaction (step 3), and trajectory validation and reuse (step 4).

3.1 Product and Environment Construction

We use developer configurations, product descriptions, device images or interface screenshots, tool APIs, operational rules, and domain skills as source materials. JoyAI-VL performs product decomposition, code generation, trajectory generation, user simulation, and semantic auditing throughout this pipeline. It decomposes existing products into reusable modules and generates variants that preserve field restrictions and component semantics, following self-instructed data generation [25]. Each product specification is a tuple $P = (I_P, W_P, C_P, S_P)$: I_P defines identity, embodiment, and interaction style; W_P defines task strategies and workflows, including decision branches and recovery rules; C_P specifies permissions, confirmation requirements, and operating constraints; and S_P specifies tools, domain skills, and perceptual and action capabilities. Each component has a corresponding typed module library.

A code-generation agent implements simulated tool and device backends from the resource interfaces. These programs validate arguments and preconditions, execute state transitions, and return results or errors, with branches for success, invalid arguments, insufficient permission, timeout, and partial execution. Calls and state changes are logged. Before admission to a library, modules undergo interface, state-access, and failure-branch tests, together with checks that backend outputs agree with the resulting states and observations.

The composer selects modules from the four libraries and assembles compatible product configurations. Compatibility checks resolve dependencies and conflicts, ensure that workflows use installed resources and obey

Product-Conditioned Multimodal Data

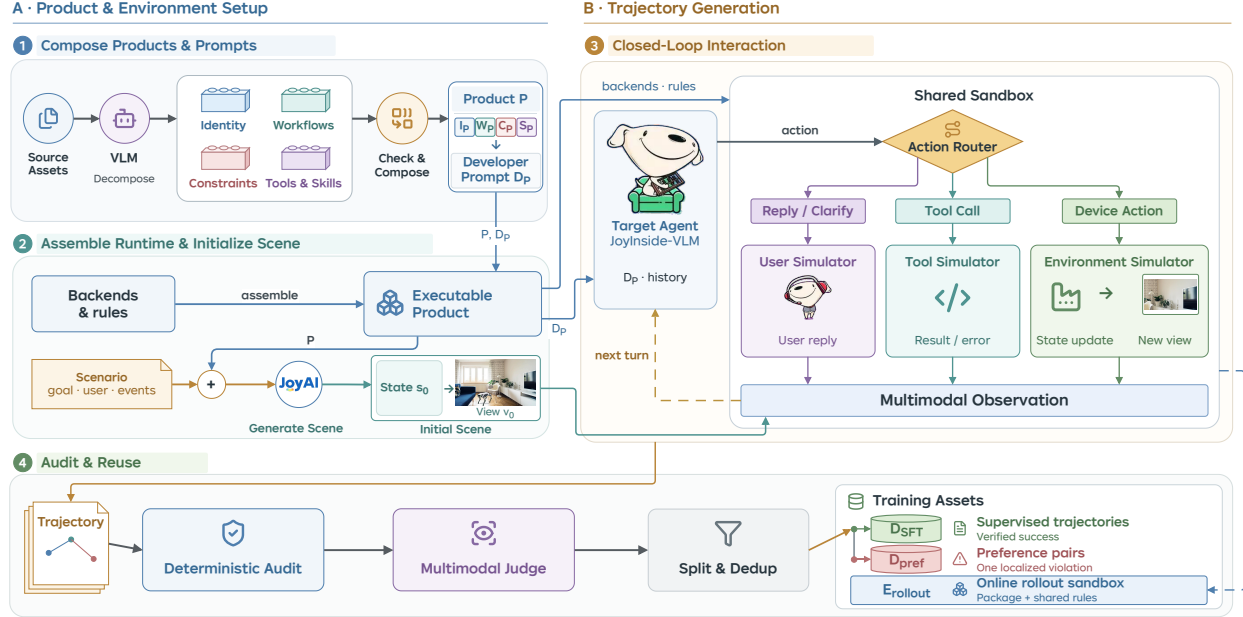


Figure 2 Product-conditioned multimodal data construction. (A, steps 1–2) Compatible product modules supply the developer prompt, executable rules, and initial scene. (B, step 3) Routed actions produce user feedback or execution results; state changes determine subsequent visual observations. (Step 4) Deterministic and multimodal audits yield verified demonstrations and preference pairs for reward-model training. Executable environments retain the product rules for online rollouts. Verified demonstrations include appropriate refusals and compliant recovery from environmental failures as well as successful task completion.

constraints, and require action arguments to be obtainable from user input, observations, prior results, or clarification. The selected capabilities must also match the embodiment and support the intended tasks. Joint composition allows, for example, the same workflow to operate with different embodiments, tool sets, and confirmation policies when compatible; coverage is bounded by the module libraries and compatibility rules.

Compositions that pass integration tests are compiled into a developer prompt D_P , resource interfaces, executable backends, state-transition rules, and observation and validation rules. Descriptive skills enter the prompt, while callable tools and device capabilities additionally expose invocation schemas and backends. The same product constraints inform both the prompt and machine-checkable predicates, aligning the behavior requested of the policy with the rules enforced and audited by the environment. Alternative prompt serializations preserve the same product semantics. Appendix B presents a developer prompt template and its correspondence to the four product components.

For each product, we construct a user goal, user profile, initial hidden state s_0 , visual assets, and a dynamic-event policy (Figure 2, step 2). We specify object identities, locations, attributes, and device states before generating the initial request and image. Initial visual assets may come from real observations, simulation, or image generation, and we check them against s_0 before interaction begins.

3.2 Closed-Loop Trajectory Generation

We load the product and scenario into a sandbox that maintains interaction state, a design also used in ToolSandbox [26]. In turn t , the agent receives the developer prompt D_P , prior interaction history h_t , and current observation $o_t = (x_t, v_t, r_t, e_t)$, where the components denote the user message, visual observation, execution result, and error, respectively. Conditioned on these inputs and the previous task state Q_{t-1} , it updates its task state Q_t and selects an action a_t . Visual observations may contain images or video. Actions

include responses, clarifications, refusals, tool calls, device actions, and termination. For trajectories without an explicit task queue, the policy generates only a_t . Hidden states, reference answers, and future event schedules are withheld from the agent.

The action router in Figure 2 (step 3) sends dialogue actions to a user simulator, as in conversational tool-use environments [8], while tool calls and device actions execute the registered backends. For any environment-changing call or device action, the backend returns an execution result, any error, and state changes, accounting for triggered events that affect execution. The shared sandbox applies these changes together with any remaining environmental effects, then generates visual feedback from the updated state using simulation, interface rendering, or image generation. Object, attribute, text, and multimodal semantic checks assess agreement between this feedback and the recorded state, including after failure or partial completion. User messages, execution results, errors, and images retain their separate sources in the interaction record. Appendix A.1 formalizes the policy interface and this execution-to-observation sequence.

Dynamic events trigger based on specified turn, state, or action conditions. The user simulator introduces goal revisions, additional requirements, or rejected confirmations; backends report timeouts, permission changes, or partial execution; and the state engine applies changes such as object movement or device shutdown. The agent observes the consequences and must update its plan, seek clarification, retry, or terminate. These interactions provide trajectories for both recovery toward an existing goal and adaptation to a revised goal.

3.3 Trajectory Validation and Data Outputs

An interaction ends upon goal completion, cancellation, valid refusal, unrecoverable failure, or exhaustion of the turn budget. Termination reason and task success are recorded separately. Each trajectory τ stores model-visible messages, observations, and actions alongside audit metadata: the product specification, hidden-state history, event log, and backend execution records.

Following execution-based verification in the construction of synthetic tool-use data [27], deterministic checks validate schemas, preconditions, permissions, confirmations, state transitions, provenance of results, and termination. A separate multimodal evaluator assesses interaction naturalness, visual grounding, adaptation to new feedback, and unsupported execution claims. Auditing uses the effective goal at each turn, including recorded user revisions. A trajectory is accepted for supervised learning only if it passes all hard checks and its semantic score reaches the acceptance threshold γ . Semantic judgments cannot override a failed hard check. Recovery demonstrations are retained only when valid actions encounter environmental failures, such as timeouts or device unavailability, and subsequent behavior remains compliant. Trajectories containing agent-side rule violations are excluded from SFT even if the agent later corrects them.

The supervised set $\mathcal{D}_{\text{SFT}}$ contains accepted demonstrations, including appropriate refusals and compliant recovery behavior. We construct preference pairs by matching an accepted demonstration with an audited trajectory containing a localized violation under the same product, task, and initial conditions. Violations include missing confirmations, invalid arguments, incorrect action order, or reliance on outdated visual evidence. These pairs form $\mathcal{D}_{\text{pref}}$ and supply contrasting trajectories for reward-model training (Section 4.4). The executable product, scenario, and validators form $\mathcal{E}_{\text{rollout}}$, where online candidates are evaluated under the same product rules. Figure 2 (step 4) summarizes these outputs. Variants of one trajectory share a split, and deduplication uses product composition, scenario source, and visual assets.

4 Method

We train a shared ReAct-style policy for bipedal, quadruped, and wheeled robots, as well as smart hardware such as interactive toys. Product specifications condition all three training stages (Figure 3): supervised fine-tuning establishes interaction behavior, multi-teacher distillation transfers complementary capabilities, and DPER refines the policy using interaction rewards. Throughout training, the policy reasons and acts on the observations available at each decision. Teachers and privileged references provide training signals; deployment uses a single student with the product configuration and interaction context.

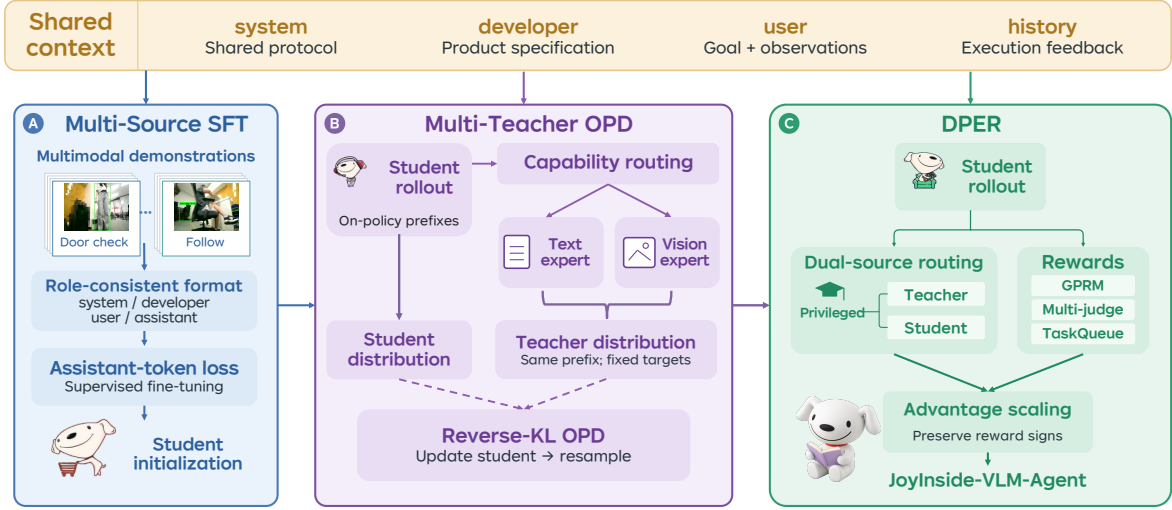


Figure 3 Three-stage policy learning. All stages use the shared product-conditioned context. (A) SFT trains on assistant tokens in multimodal demonstrations. (B) Text and vision teachers supervise student-generated prefixes through reverse-KL OPD. (C) DPER selects privileged teacher or student evidence to rescale reward-derived token advantages while preserving their signs. Rewards combine GPRM, multi-judge rankings, and task-queue consistency. Teachers and privileged references are used only during training; deployment uses a single student.

4.1 Product-Conditioned Supervised Fine-Tuning

Supervised fine-tuning teaches the policy to interpret product specifications and produce context-appropriate responses and tool calls. We combine general instructions, text-based tool interactions, image-conditioned decisions, multi-turn execution feedback, and verified sandbox trajectories (Section 3). Following instruction tuning [28, 29], we organize examples from all sources using consistent message roles: system messages define the interaction protocol, developer messages encode the product specification, and user and tool messages provide requests, observations, and execution results. Our chat template serializes D_P as a separate **developer** message using the same message delimiters as other roles (Appendix B.1). Assistant messages contain the supervised reasoning, responses, and actions.

Let π_θ denote the student policy. For a serialized trajectory τ_i , let $y_{i,n}$ be its token at position n and $c_{i,n}$ the context available before that token: the protocol, product prompt D_P , preceding messages and assistant outputs, and available visual observations and execution feedback. Let $\mathcal{M}_i$ denote the valid assistant-token positions, excluding padding. We minimize cross-entropy over these positions, summing token losses within each trajectory and averaging over demonstrations. User messages, observations, and tool returns remain conditioning inputs, and explicit task queues, when present, are supervised as assistant outputs. Appendix A.2 states the objective in full and specifies the mask.

The product prompt specifies the available tools, their argument requirements, and execution preconditions. Demonstrations include clarification, explanations of capability limits, and recovery from execution failures under different product configurations. Supervision therefore covers both output syntax and whether a proposed action is appropriate for the current product and observed state. The resulting checkpoint initializes multi-teacher distillation.

ReAct decisions and task-state generation. At each ReAct decision, the student generates reasoning followed by either a tool call or a direct response, such as an answer, clarification, or refusal. When required by the interaction protocol, the updated task queue is emitted in the same assistant message for either decision type. All components are generated autoregressively by the same policy and supervised as assistant outputs. The queue contains ordered batches of coarse-grained action labels, while a tool call specifies a registered tool and concrete arguments for an action in the current head batch. Tool results become observations for subsequent

decisions. Neither issuing a call nor producing a direct response establishes completion of a pending tool action; completion must be verified from execution feedback. Appendix B.2 illustrates both output branches.

4.2 Multi-Teacher On-Policy Distillation

We use on-policy distillation [17] to transfer complementary capabilities from text and vision teachers into a single student, following staged multi-teacher training [19]. Text teachers supervise planning, tool selection, argument construction, and feedback interpretation; vision teachers supervise decisions requiring image evidence. Each assigned teacher receives the product constraints and student-generated prefix, together with the visual observation when required. We place this stage before DPER based on evidence that OPD can improve subsequent RL performance even when its immediate accuracy gains are small [20].

Let π_{old} denote the student snapshot used to collect the current rollout batch, and let $\kappa(i, n)$ identify the capability-matched teacher assigned to the decision containing token n . We evaluate the teacher distribution $q_{i,n}(\cdot) = \pi_{T_{\kappa(i,n)}}(\cdot \mid c_{i,n})$ on the student’s sampled context and stop gradients through it. With aligned teacher and student token spaces, we compute the reverse-KL OPD objective [20], minimizing divergence from the student to the assigned teacher:

$$\mathcal{L}_{\text{OPD}}(\theta) = \mathbb{E}_{\tau_i \sim \pi_{\text{old}}} \left[\sum_{n \in \mathcal{S}_i} D_{\text{KL}}(\pi_{\theta}(\cdot \mid c_{i,n}) \parallel q_{i,n}(\cdot)) \right], \quad (1)$$

where $\mathcal{S}_i \subseteq \mathcal{M}_i$ denotes the assistant positions selected for distillation. At each position, the two distributions condition on the same student-generated prefix. The expectation accounts for task sampling, student rollouts, and observations returned by the environment.

Sampled trajectories and teacher targets remain fixed during each update, and only student parameters receive gradients. We collect each subsequent rollout batch with the updated student and obtain new teacher targets. The distilled checkpoint initializes DPER; teacher assignment is used only during training.

4.3 DPER: Token-Level Evidence Routing

Dual-Source Privileged On-Policy Self-Distillation with Token-Level Evidence Routing (DPER) retains RLSD’s group-relative base advantage and sign-preserving advantage modulation [5], and introduces token-level source selection inspired by DOPD [6]. Trajectory rewards evaluate an interaction as a whole, whereas privileged references provide additional evidence about individual sampled tokens. A frozen privileged teacher and a parameter-sharing privileged student score tokens on the same rollout. Routing selects the evidence used to scale each token advantage, while the group-relative reward comparison determines its sign. DPER uses the selected evidence in a policy-gradient objective without adding DOPD’s separate KL/JS distillation losses.

4.3.1 Dual-Source Privileged Scoring

For each product-conditioned task instance, we collect a group $\mathcal{C} = \{\tau_i\}_{i=1}^K$ using π_{old} . Let z_i be a reference answer, solution trace, or tool trajectory available only during training. For each sampled assistant token, we compute

$$\begin{aligned} p_{0,in} &= \pi_{\text{old}}(y_{i,n} \mid c_{i,n}), \\ p_{S,in}^p &= \pi_{\text{old}}(y_{i,n} \mid c_{i,n}, z_i), \\ p_{T,in}^p &= \pi_T(y_{i,n} \mid c_{i,n}, z_i), \end{aligned} \quad (2)$$

where π_T is a frozen teacher with a token space aligned with the student’s. The privileged student uses the same parameter snapshot as the rollout policy with the additional reference. Both privileged branches evaluate the sampled tokens under teacher forcing. Keeping the trajectory fixed makes their scores refer to the same decisions and observed consequences. The privileged student’s comparison with the rollout policy changes the information available to the same model, while the teacher also contributes a different model’s assessment. The context $c_{i,n}$ retains intervening user messages, observations, and tool results across ReAct turns; the generating policy never receives z_i .

4.3.2 Token-Level Source Selection

We measure disagreement by $a_{in} = |\log p_{T,in}^p - \log p_{S,in}^p|$ and joint support by $u_{in} = p_{T,in}^p + p_{S,in}^p$. Their medians, $\tilde{a}$ and $\tilde{u}$, are computed over valid assistant tokens in the rollout batch. High and low are relative to these batch statistics. These median thresholds are our choice for adapting DOPD’s source-selection principle. The selected evidence probability is

$$p_{in}^* = \text{sg} \left[\begin{cases} p_{T,in}^p, & a_{in} < \tilde{a} \text{ and } u_{in} \geq \tilde{u}, \\ p_{T,in}^p, & a_{in} \geq \tilde{a} \text{ and } p_{T,in}^p \geq p_{S,in}^p, \\ p_{S,in}^p, & \text{otherwise,} \end{cases} \right], \quad (3)$$

where sg stops gradients through the selected score. The conditions define four cases, following DOPD’s source-selection structure [6]:

- Low disagreement, high joint support (LH). The two sources agree with relatively high joint support for the sampled token, and we select the privileged teacher.
- Low disagreement, low joint support (LL). Agreement is accompanied by low joint support for the sampled token, and we select the privileged student.
- High disagreement, stronger teacher support (HT). We select the privileged teacher when its probability for the sampled token is at least as high as the student’s, including ties.
- High disagreement, stronger student support (HS). We select the privileged student when its probability for the sampled token exceeds the teacher’s.

All four cases use the same advantage-modulation rule below. Probability support alone does not establish that an action is executable or consistent with the user’s goal.

4.3.3 Evidence-Weighted Policy Optimization

Following RLSD [5], we compute group-relative base advantages A_i from the composite trajectory rewards in Equation (6) within each same-task rollout group. We retain RLSD’s evidence-weighting rule, using the routed probability p_{in}^* as its privileged score:

$$\Delta_{in} = \text{sg}[\log p_{in}^* - \log p_{0,in}], \quad w_{in} = \exp(\text{sign}(A_i)\Delta_{in}). \quad (4)$$

The token advantage is

$$\hat{A}_{in} = A_i [(1 - \lambda) + \lambda \text{clip}(w_{in}, 1 - \epsilon_w, 1 + \epsilon_w)], \quad (5)$$

where $\lambda \in [0, 1]$ controls evidence strength and $\epsilon_w \geq 0$ is the clipping radius. The multiplier is positive, preserving the sign of each nonzero advantage and leaving zero advantages unchanged. Relative to the rollout policy, greater privileged support increases reinforcement for positive advantages, while weaker support increases suppression for negative advantages, subject to clipping. Clipping bounds the influence of large probability contrasts, and mixing with the original advantage controls the contribution of privileged evidence. When $\lambda = 0$ or $\epsilon_w = 0$, the original trajectory advantage is recovered. A selected probability equal to the rollout probability also leaves the corresponding token advantage unchanged. The sign guarantee applies to individual token advantages. If routing always selects $p_{S,in}^p$, the token-advantage formula reduces to RLSD’s self-distillation modulation under the same rewards and hyperparameters.

We maximize a clipped policy surrogate using the evidence-weighted token advantages, averaging over valid assistant tokens within each trajectory and then over task instances and their sampled trajectories. Appendix A.3 gives the full objective and policy-ratio definition. Privileged scores, routing decisions, and token advantages are computed once per rollout batch, detached from gradients, and held fixed during its policy updates. They are recomputed from the next student snapshot after new rollouts are collected. Only the unprivileged policy is optimized for deployment.

4.4 Multi-Source Interaction Rewards

We evaluate process quality through GPRM, product-constrained behavior through multi-judge ranking, and goal-consistent planning through TaskQueue. For the queue-bearing interaction path described here, normal reward aggregation requires a successfully constructed scoring input and a valid GPRM score, including zero. Let $\mathcal{B}_i \subseteq \{\text{GPRM, Judge, Queue}\}$ contain GPRM and the available optional scores for candidate τ_i . Its normal-path reward is

$$R_{\text{content},i} = \frac{\sum_{b \in \mathcal{B}_i} \beta_b R_{b,i}}{\sum_{b \in \mathcal{B}_i} \beta_b}, \quad R_i = R_{\text{content},i} \ell_{\text{format},i} \ell_{\text{len},i}, \quad (6)$$

where $\beta_b \geq 0$ and $\beta_{\text{GPRM}} > 0$, ensuring a positive denominator. The multiplicative factors $\ell_{\text{format},i} \in [0.05, 1]$ and $\ell_{\text{len},i} \in [0.1, 1]$ assess output structure and length. Missing optional scores are excluded and the remaining weights are renormalized; valid zero scores from any branch retain their weights.

The format factor assesses compliance with the required output structure. The interaction protocol determines whether a task queue is required; choosing a direct response does not waive an existing queue requirement. Dialogue-only outputs for which the protocol requires no queue incur no missing-queue penalty. Empty outputs receive the minimum coefficient. An otherwise usable output that omits a required task queue receives a coefficient above this minimum but below that of a complete valid output. If this omission makes the Queue score unavailable on the normal scoring path, the coefficient is capped at the ratio of the remaining available reward weight to that same weight plus the Queue weight. With other scores and the length factor held fixed, this cap prevents renormalization alone from increasing the reward relative to a valid output with a zero Queue score. The length factor penalizes excessive reasoning and response content using task-dependent allowances.

If the scoring input cannot be constructed, the scorer returns $R_i = 0$. If GPRM is unavailable because no process step can be evaluated, its request fails, or no valid score is returned, the scorer instead assigns $R_i = 0.1 \ell_{\text{format},i} \ell_{\text{len},i}$ and skips Judge and Queue evaluation for that candidate. A valid zero GPRM score follows normal aggregation and does not trigger this fallback. This fixed fallback is an implementation rule, not a process-quality judgment. These candidate-level rules precede normal aggregation; group-level exclusions are described below. Appendix A.5 details score availability and fallback handling. The resulting rewards supply DPER’s group-relative advantages.

4.4.1 Generative Process Evaluation

GPRM combines process supervision [21] with generative verification [22]. For each reasoning step s , its input contains the task and trajectory prefix through that step. Prefix evaluation exposes intermediate errors that a final-answer judgment does not separately represent. The evaluator generates a verification rationale, then predicts an ordered quality grade $g \in \mathcal{G} = \{A, B, C, D\}$ at a fixed judgment position. The rationale assesses correctness and contextual relevance and identifies errors in the current reasoning. This evaluation uses the language model’s output distribution without an additional scoring head.

We obtain grade probabilities $p_{s,g}$ by applying softmax only over the four grade-token logits after the rationale (Appendix A.4). With grade weights $(w_A, w_B, w_C, w_D) = (1, \frac{2}{3}, \frac{1}{3}, 0)$, the step reward and trajectory score are

$$r_s^{\text{proc}} = \sum_{g \in \mathcal{G}} w_g p_{s,g}, \quad R_{\text{GPRM},i} = \frac{1}{L_i} \sum_{s=1}^{L_i} r_s^{\text{proc}}, \quad (7)$$

where $L_i > 0$ is the number of evaluated reasoning steps; the candidate index is suppressed on step-level quantities. With no evaluable step, the scorer takes the fallback above without evaluating this average. Grades run from highest quality (A) to lowest (D). The expectation retains uncertainty across grades, and averaging puts trajectories with different numbers of evaluated steps on the same score range, $[0, 1]$. The result measures expected process quality under the evaluator’s grade distribution.

The evaluator’s training data include trajectories from the preference pairs in $\mathcal{D}_{\text{pref}}$ (Section 3). It is trained with autoregressive cross-entropy on annotated sequences containing the task, reasoning prefix, verification

rationale, and quality grade. Rationale annotations record the judgment basis, identified errors, and their implications for subsequent reasoning. These labels supervise process quality independently of the candidate rankings below.

4.4.2 Multi-Judge Ranking

Each judge receives the same complete candidate group $\mathcal{C}$ and produces a tiered ranking with ties. This relative protocol avoids requiring scores from different judges to share an absolute scale [30]. Task-dependent criteria cover product capabilities, behavioral constraints, role consistency, planning dependencies, use of observations, goal completion, and redundancy within and across candidates. Presenting the full group lets judges compare alternative behaviors under the same task and identify repeated content across rollouts. Their outputs are ranking ballots over shared candidate identities.

For a complete ranking group with $K > 1$ and at least one valid ballot, let $\bar{r}_{mi}$ be candidate i ’s average rank in valid ballot m , with rank 1 best. Tied candidates share the mean of their occupied ranks. Invalid ballots are discarded in full; the remaining ballots form $\mathcal{V}$, with nonnegative weights renormalized so that $\sum_{m \in \mathcal{V}} \alpha_m = 1$. We aggregate their normalized rank scores into b_i , rerank candidates by decreasing b_i , and map the resulting average rank $\bar{r}_i$ to the final reward:

$$b_i = \sum_{m \in \mathcal{V}} \alpha_m \frac{K - \bar{r}_{mi}}{K - 1}, \quad R_{\text{Judge},i} = \delta_J + (1 - \delta_J) \frac{K - \bar{r}_i}{K - 1}, \quad (8)$$

where δ_J is a tunable reward-calibration hyperparameter, with its value specified in Section 5.1. Both ranking stages preserve ties. Weighted ballot scores determine the aggregate ordering; the final mapping uses rank spacing rather than numerical gaps between aggregate scores. These scores express relative preferences within the group; they do not measure absolute task success or define GPRM grades. If every candidate in a group triggers a deterministic hard failure, we skip both relative ranking and DPER policy updates for that group. If no valid ballot remains, the Judge branch is unavailable: normal-path candidates use the remaining scores in Equation (6), while candidates assigned an early-exit or fallback reward retain that value.

4.4.3 Task-Queue Consistency

TaskQueue evaluates semantic alignment and continuity across turns. At turn r , let $Q^{(r)} = [B_1^{(r)}, \dots, B_{L_r}^{(r)}]$, with the head batch representing the current tasks. On the first turn, JoyAI-VL inversely rewrites the queue as a natural-language task description under task-granularity rules. The first-turn score compares this reconstructed intent with the initial request x , as defined in Equation (A.5). It evaluates intent coverage without requiring an exact match to a reference queue and uses no continuity term.

Queues use nested batches of coarse-grained action labels without concrete objects or arguments. The grouping rules separate actions from queries, place individual queries or broadcasts in separate batches, and start a new batch when the interaction subject changes. These rules specify the queue representation; inverse rewriting preserves the proposed tasks and grouping without adding, removing, or correcting them.

On feedback turns with a nonempty observation block and a preceding queue, $q_{\text{obs}}^{(r)}$ compares a JoyAI-VL inverse rewrite of the current queue with task requirements updated from the observation history, including user revisions, visual observations, execution feedback, and errors. Qwen3-Embedding-4B encodes the compared texts, and cosine similarity is mapped to $[0, 1]$, consistent with the overall reward range. The continuity term $q_{\text{cont}}^{(r)}$ compares the queue against the preceding queue updated by verified execution: completed heads are removed, failed or unexecuted heads are retained, and partially completed heads retain their unfinished tasks in order. It counts exact batch matches at corresponding positions and normalizes by the longer queue’s length. A batch matches only if all task entries are equal at the same positions; neither batches nor tasks within a batch may be reordered. With both scores available,

$$q_{\text{queue}}^{(r)} = \frac{\eta_{\text{obs}} q_{\text{obs}}^{(r)} + \eta_{\text{cont}} q_{\text{cont}}^{(r)}}{\eta_{\text{obs}} + \eta_{\text{cont}}}, \quad (9)$$

with nonnegative weights and a positive denominator. All components lie in $[0, 1]$. Available components retain their weights, which are renormalized when another component is missing; evaluated zeros remain included. If no component is available, the turn has no Queue score. The two terms balance adaptation to revised goals and preservation of pending task order: a justified revision may reduce positional agreement, so exact queue matching alone is insufficient. The trajectory score $R_{\text{Queue},i}$ averages evaluated turn scores, including the initial score; it is unavailable if no turn is evaluable. Appendix A.6 gives the component definitions and empty-queue handling, and Table 2 provides the inverse-rewriting prompt.

5 Experiments

We evaluate JoyInside-VLM-Agent on four public tool-use benchmarks and two proprietary evaluation sets, analyze training dynamics, and examine two physical-robot interactions. The quantitative results assess tool-call trajectories, step-wise decisions, argument matching, and response overlap; the cases illustrate decisions based on visual evidence and execution feedback.

5.1 Experimental Setup

Training data. Our training corpus comprises approximately 120K examples: 50K proprietary interaction trajectories, 40K supplementary examples adapted from public agent datasets, and 30K video-instruction examples from LLaVA-Video-178K [31] and Video-ChatGPT [32]. The proprietary trajectories span quadruped, bipedal, and wheeled robots and AI-enabled plush toys. The supplementary subset draws on four public sources: τ^2 -Bench [9] for conversational tool use and agent-user coordination, VitaBench [33] for interactive tasks spanning multiple service scenarios, DeepPlanning [34] for long-horizon planning with verifiable constraints, and SWE-bench Verified [35] for repository-level issue resolution. We refine and augment all four sources using the task-variation, interaction-generation, and validation procedures adapted from Section 3, producing supplementary training data for our model. The 40K count refers to the examples retained after augmentation and filtering.

Evaluation datasets. We evaluate on MSAgent [36], BFCL v4 [37], ToolBench [38], and ToolACE [11]. We report benchmark-level scores on all four datasets and additional step-wise and response-overlap metrics on ToolBench and ToolACE. We also evaluate on two proprietary JoyInside datasets. JoyInside-Devices focuses on extended multi-turn dialogue and simple device control across AI-enabled toys, smart appliances, and other AIoT devices. Tasks span conversational assistance, context tracking, evolving user preferences, status queries, and basic control commands. JoyInside-Robotics focuses on long-horizon, visually grounded reasoning and complex robot control. Tasks involve target search and localization, spatial reasoning, multi-step planning and execution, and adaptive replanning based on visual observations and execution feedback.

Models and product configuration. JoyInside-VLM-Agent starts from Qwen3.5-35B-A3B and undergoes supervised fine-tuning, multi-teacher distillation, and DPER (Section 4). We compare the resulting student with the base model and Qwen3.5-397B-A17B. At inference, it uses product specifications and interaction history; privileged references are limited to training. We also use SkillOpt [39] to refine the domain-skill instructions used on JoyInside-Devices and JoyInside-Robotics while keeping model parameters fixed during skill refinement. Results on these sets therefore reflect the trained policy and refined skills together.

Training teachers. For the experiments reported here, we use Qwen3.8-27B for the OPD teachers and the frozen DPER teacher. Teacher selection is configurable, subject to the token-space alignment required by the objectives in Section 4.

DPER training configuration. We train DPER on 16 NVIDIA H200 GPUs for 350 steps with 128 prompts per batch and $K = 8$ responses per prompt, yielding 1,024 sampled responses. The prompt minibatch size is 32, with an actor microbatch size of one sequence per GPU. The learning rate is 5×10^{-6} , with 10 warmup steps and weight decay 0.1. Rollouts use temperature 1.0, top- p 0.95, and top- k 20, with maximum prompt and response lengths of 16,384 and 2,048 tokens, respectively. DPER uses evidence strength $\lambda = 0.5$. Policy losses

are averaged over valid tokens within each response and then over responses. Privileged teacher and student branches provide scores only; separate distillation loss, KL regularization, and the entropy bonus are disabled. In Equation (6), the GPRM, Judge, and Queue weights are 0.4, 0.3, and 0.3, respectively, renormalized over available scores; the fallback content factor is 0.1. We set the Judge calibration hyperparameter in Equation (8) to $\delta_J = 0.3$. Within TaskQueue, Equation (9) uses $\eta_{\text{obs}} = 0.6$ for semantic agreement and $\eta_{\text{cont}} = 0.4$ for continuity. Format and length factors penalize structural violations and excessive output length, with length allowances adjusted to task complexity (Section 4.4).

Benchmark-level evaluation. For MSAgent, ToolBench, and ToolACE, we report trajectory accuracy, measured by exact match of the complete tool-call sequence. For BFCL v4, we use the official evaluation script and report its overall score, following the benchmark’s category-specific evaluation and aggregation rules. Table 1(a) reports the resulting scores on a 0–100 scale.

Step-wise and response metrics. For ToolBench, ToolACE, JoyInside-Devices, and JoyInside-Robotics, Plan.EM measures decision-type exact match at evaluated steps, rather than the correctness of a complete plan. Act.EM measures tool-name exact match, and Avg.F1 gives partial credit for argument matching. HalluRate is the percentage of evaluated tool-call records that select a tool outside the available capability set; it measures unsupported tool selection. ROUGE-L [40] measures lexical overlap between final and reference responses. All scores in Table 1 are reported on a 0–100 scale; only HalluRate is better when lower.

5.2 Training Dynamics

Figure 4 compares recorded training runs of DPER, GRPO, RLSD [5], and DOPD [6] over the first 300 steps. We track actor entropy, response length, and maximum reward throughout optimization. The DPER run uses the configuration in Section 5.1 and continues to 350 steps.

DPER’s maximum reward rises rapidly and remains above the baselines for most plotted steps (panel c), while response length grows moderately before fluctuating near RLSD and below DOPD (panel b). Its entropy rises early and then fluctuates around 0.40–0.48, generally above GRPO but sometimes below RLSD (panel a). Maximum reward characterizes the highest-scoring sampled output and does not establish performance across all rollouts. These curves describe training behavior; held-out evaluations and component ablations are needed to assess generalization and isolate routing and reward contributions.

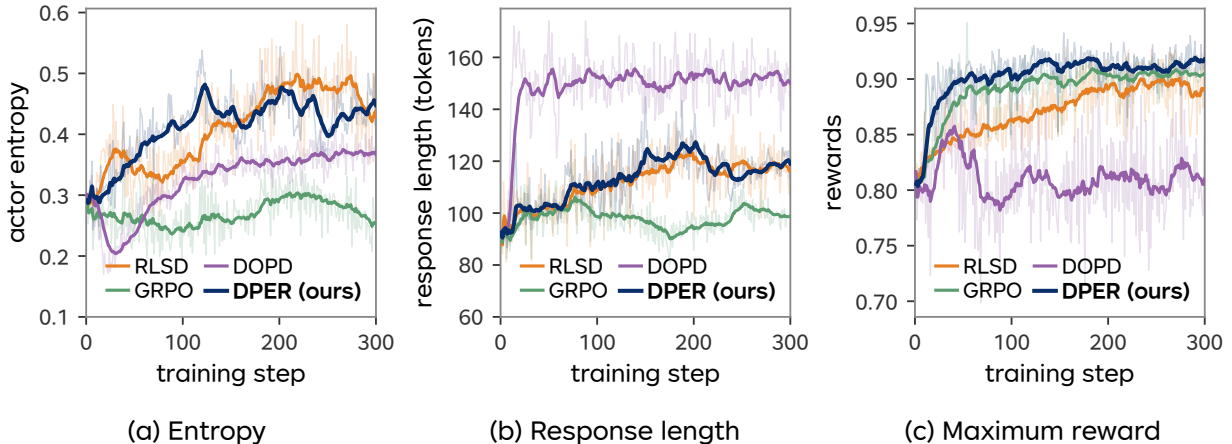


Figure 4 Training dynamics over the first 300 steps. DPER is compared with GRPO, RLSD, and DOPD in terms of (a) actor entropy, (b) response length in tokens, and (c) maximum reward. Dark curves show smoothed trends; faint traces show step-to-step fluctuations.

Table 1 Tool-use performance on public benchmarks and proprietary evaluation sets. Boldface indicates the best result for each dataset and metric.

(a) Benchmark-level scores ($\uparrow$)						
Dataset	Qwen3.5-35B-A3B	Qwen3.5-397B-A17B	JoyInside-VLM-Agent			
MSAgent	88.46	92.41	90.39			
BFCL v4	67.30	72.90	77.44			
ToolBench	18.30	21.00	23.50			
ToolACE	50.92	52.27	51.53			

(b) Step-wise tool-use and response metrics						
Dataset	Model	Plan.EM $\uparrow$	Act.EM $\uparrow$	Avg.F1 $\uparrow$	HalluRate $\downarrow$	ROUGE-L $\uparrow$
<i>Public benchmarks</i>						
ToolBench	Qwen3.5-35B-A3B	74.48	32.72	27.94	1.14	3.92
	Qwen3.5-397B-A17B	80.77	43.55	38.78	0.31	5.11
	JoyInside-VLM-Agent	81.03	54.66	49.17	0.52	11.84
ToolACE	Qwen3.5-35B-A3B	96.16	88.19	75.59	0.70	35.45
	Qwen3.5-397B-A17B	96.70	88.63	75.91	0.44	37.72
	JoyInside-VLM-Agent	96.86	89.06	75.71	0.52	35.24
<i>Proprietary evaluation sets</i>						
JoyInside-Devices	Qwen3.5-35B-A3B	71.63	45.44	33.47	0.19	13.02
	Qwen3.5-397B-A17B	75.64	60.04	45.17	0.10	25.48
	JoyInside-VLM-Agent	89.90	84.99	57.18	0.02	26.57
JoyInside-Robotics	Qwen3.5-35B-A3B	92.76	87.85	74.84	0.43	9.76
	Qwen3.5-397B-A17B	94.10	90.56	77.89	0.25	12.04
	JoyInside-VLM-Agent	97.58	93.73	83.10	0.04	23.30

Note. Scores are reported on a 0–100 scale; arrows indicate the preferred direction. Panel (a) reports trajectory accuracy for MSAgent, ToolBench, and ToolACE, and the BFCL v4 overall score. Boldface marks the best result within each dataset and metric. JoyInside-VLM-Agent is initialized from Qwen3.5-35B-A3B. Trajectory accuracy was not reported for JoyInside-Devices or JoyInside-Robotics. Results for our model on the proprietary evaluation sets include domain-specific skill refinement and therefore reflect the evaluated system configuration.

5.3 Benchmark Results

Table 1 reports benchmark-level scores on four public datasets and step-wise and response-overlap metrics on ToolBench, ToolACE, and the two proprietary sets.

Public benchmarks. JoyInside-VLM-Agent achieves a BFCL v4 overall score of 77.44, exceeding Qwen3.5-397B-A17B and Qwen3.5-35B-A3B by 4.54 and 10.14 points, respectively. On MSAgent, it reaches 90.39% trajectory accuracy, between the larger (92.41%) and base (88.46%) models.

On ToolBench, JoyInside-VLM-Agent leads in trajectory accuracy (23.5%) and four of the five tool-use and response metrics. Its Act.EM of 54.66 exceeds that of the larger model by 11.11 points, although the latter has a lower HalluRate (0.31 versus 0.52). On ToolACE, our model has the highest Plan.EM (96.86) and Act.EM (89.06), but trails the larger model in trajectory accuracy (51.53% versus 52.27%), Avg.F1, HalluRate, and ROUGE-L. Thus, gains in decision-type and tool-name matching do not consistently extend to complete-trajectory accuracy or argument matching.

Proprietary datasets. JoyInside-VLM-Agent leads on all five reported metrics for both JoyInside-Devices and JoyInside-Robotics. Its Act.EM reaches 84.99 and 93.73, exceeding Qwen3.5-397B-A17B by 24.95 and 3.17 points, respectively; Avg.F1 reaches 57.18 and 83.10, while HalluRate falls to 0.02% and 0.04%. These results characterize the trained policy with refined domain skills. We do not report trajectory exact-match accuracy for these two sets.

5.4 Real-World Case Studies

Two office-based ReAct trajectories on a quadruped illustrate how visual evidence and execution feedback guide a greeting task: in one, the target is initially outside the camera view; in the other, the target disappears after an attempted approach.

Greeting under partial observability. The robot is asked to greet a man in a light shirt and black shorts who holds a laptop (Figure 5). Because the target is not initially visible in the camera view, it searches with `turn_left` and checks each observation against the requested attributes. After detecting a match, it calls `visual_approach`, then `perform_greeting`, and reports completion. The transition from search to approach follows new visual evidence.



Figure 5 Visual search before greeting. The requested person is outside the quadruped’s camera view in steps 1–3, identified by clothing and laptop attributes in step 4, and seen at close range in steps 5–6. Frames follow the interaction from left to right; bounding boxes show object detections.

Replanning after a failed approach. The robot is asked to greet a woman in a white shirt and light blue jeans (Figure 6). Its initial `visual_approach` fails and the target leaves the field of view. The robot then searches with `turn_right`, rejects mismatched people, and reports failure after six subsequent search steps without reacquiring the target. Execution feedback thus triggers replanning, and the exhausted search budget triggers termination.



Figure 6 Visual search after a failed approach. The target is visible in step 1 but is not reacquired during the quadruped’s subsequent rotational search in steps 2–7. Frames follow the interaction from left to right; bounding boxes show object detections.

These cases illustrate visually grounded decisions, feedback-driven replanning, and explicit reporting of success or failure. The quadruped trajectories are selected qualitative examples within a broader application scope that also includes humanoid robots and AI devices.

6 Conclusion

We presented JoyInside-VLM-Agent, a multimodal agent that combines verified, product-conditioned trajectories with staged policy learning to adapt behavior to the roles, capabilities, and constraints of robotic

platforms and smart hardware. Multi-source rewards assess reasoning quality, adherence to product constraints, and alignment with evolving goals. DPER routes token-level evidence between privileged teacher and student policies to scale token advantages while preserving their reward-derived signs. Built on Qwen3.5-35B-A3B, the model achieves a BFCL v4 overall score of 77.44, while results on the other public benchmarks are mixed. On JoyInside-Devices and JoyInside-Robotics, the evaluated system, which includes domain-specific skill refinement, leads the baselines on all reported metrics. Two quadruped case studies illustrate visually grounded search, replanning from execution feedback, and explicit failure reporting.

7 Limitations and Future Work

JoyInside-VLM-Agent has been applied to quadruped robots, humanoid robots, and AI devices. Generalization to a broader range of product configurations and operating environments still requires systematic evaluation. Product-conditioned data construction also depends on the coverage of the capability library and the fidelity of sandbox-generated feedback. Future work will expand product coverage and measure task completion and recovery over longer real-world interactions. We also plan to evaluate across more diverse benchmarks in multimodal understanding, reasoning, planning, and tool use to assess the generality of the learned capabilities beyond the current evaluation suite.

The current results do not isolate the contributions of data construction, reward design, and DPER routing, and the proprietary evaluations additionally include domain-specific skill refinement. Controlled ablations are needed to separate these effects. Moreover, model-based rewards and embedding similarity provide imperfect proxies for task success; preserving reward-derived advantage signs does not resolve errors in the underlying reward. We plan to investigate execution-grounded reward calibration and robustness to incomplete or noisy feedback, alongside the component ablations, to better understand how each stage supports reliable product-specific behavior.

References

- [1] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, et al. Palm-e: An embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.
- [2] Open X-Embodiment Collaboration. Open x-embodiment: Robotic learning datasets and rt-x models. arXiv preprint arXiv:2310.08864, 2023.
- [3] Evan King, Haoxiang Yu, Sangsu Lee, and Christine Julien. Sasha: Creative goal-oriented reasoning in smart homes with large language models. In *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, 2024.
- [4] Ruei-Che Chang, Yuxuan Liu, and Anhong Guo. Worldscribe: Towards context-aware live visual descriptions. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, 2024.
- [5] Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled RLVR. arXiv preprint arXiv:2604.03128, 2026. URL <https://arxiv.org/abs/2604.03128>.
- [6] Xinlei Yu, Gen Li, Qingyi Si, Guibin Zhang, Yuqi Xu, Congcong Wang, Shuai Dong, Kaiwen Tuo, Xiangyu Zeng, Kaituo Feng, Qunzhong Wang, Yang Shi, Xiaobin Hu, Xiangyu Yue, Jiaqi Wang, and Shuicheng Yan. DOPD: Dual on-policy distillation. arXiv preprint arXiv:2606.30626, 2026. URL <https://arxiv.org/abs/2606.30626>.
- [7] Jaewook Lee, Jun Wang, Elizabeth Brown, Liam Chu, Sebastian S. Rodriguez, and Jon E. Froehlich. Gaze-pointar: A context-aware multimodal voice assistant for pronoun disambiguation in wearable augmented reality. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI)*, 2024. Article 408.
- [8] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2406.12045.
- [9] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2 -bench: Evaluating conversational agents in a dual-control environment. arXiv preprint arXiv:2506.07982, 2025.

- [10] Peijie Yu, Wei Liu, Yifan Yang, Jinjian Li, Zelong Zhang, Xiao Feng, and Feng Zhang. Benchmarking llm tool-use in the wild. In International Conference on Learning Representations (ICLR), 2026.
- [11] Weiwen Liu et al. Toolace: Winning the points of llm function calling. In International Conference on Learning Representations (ICLR), 2025.
- [12] 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), 2025.
- [13] Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. arXiv preprint arXiv:2501.11425, 2025.
- [14] Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning. arXiv preprint arXiv:2602.10090, 2026.
- [15] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In Findings of the Association for Computational Linguistics (ACL), 2024.
- [16] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-flan: Designing data and methods of effective agent tuning for large language models. arXiv preprint arXiv:2403.12881, 2024.
- [17] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In International Conference on Learning Representations (ICLR), 2024.
- [18] Qixiang Yin, Huanjin Yao, Yuchen Cai, Jianghao Chen, Ziyi Wang, Min Yang, Fei Su, and Zhicheng Zhao. H-opd: Confidence aware heterogeneous multi-teacher multimodal on-policy distillation. arXiv preprint arXiv:2607.02592, 2026.
- [19] Agents-A1 Team. Scaling the horizon, not the parameters: Reaching trillion-parameter performance with a 35B agent. arXiv preprint arXiv:2606.30616, 2026. URL <https://arxiv.org/abs/2606.30616>.
- [20] Shuai Dong, Yongfu Zhu, Yuqi Xu, Weichu Xie, et al. RL starts before RL: On policy distillation for better reinforcement learning. arXiv preprint arXiv:2609.28145, 2026. URL <https://arxiv.org/abs/2609.28145>.
- [21] Hunter Lightman, Vineet Kosaraju, Yura Burda, et al. Let’s verify step by step. arXiv preprint arXiv:2305.20050, 2023.
- [22] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In International Conference on Learning Representations (ICLR), 2025.
- [23] Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. Replacing judges with juries: Evaluating llm generations with a panel of diverse models. arXiv preprint arXiv:2404.18796, 2024.
- [24] Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, et al. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking. In Conference on Language Modeling (CoLM), 2024.
- [25] Yizhong Wang et al. Self-instruct: Aligning language models with self-generated instructions. arXiv preprint arXiv:2212.10560, 2022.
- [26] Jiarui Lu et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. arXiv preprint arXiv:2408.04682, 2024.
- [27] Zuxin Liu et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. arXiv preprint arXiv:2406.18518, 2024.
- [28] Long Ouyang, Jeffrey Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35:27730–27744, 2022.
- [29] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In Advances in Neural Information Processing Systems, 2023.

- [30] W. Niu et al. From absolute to relative: Rethinking reward shaping in group-based reinforcement learning. arXiv preprint arXiv:2601.23058, 2026.
- [31] Yuanhan Zhang, Jinming Wu, Wei Li, Bo Li, Zejun Ma, Ziwei Liu, and Chunyuan Li. LLaVA-Video: Video instruction tuning with synthetic data. arXiv preprint arXiv:2410.02713, 2024.
- [32] Muhammad Maaz, Hanoona Rasheed, Salman Khan, and Fahad Shahbaz Khan. Video-ChatGPT: Towards detailed video understanding via large vision and language models. arXiv preprint arXiv:2306.05424, 2023.
- [33] Wei He, Yueqing Sun, Hongyan Hao, et al. VitaBench: Benchmarking LLM agents with versatile interactive tasks in real-world applications. arXiv preprint arXiv:2509.26490, 2025.
- [34] Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, ChenXu Lv, and Junyang Lin. DeepPlanning: Benchmarking long-horizon agentic planning with verifiable constraints. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 7377–7407. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.335.
- [35] OpenAI. Introducing SWE-bench Verified. <https://openai.com/index/introducing-swe-bench-verified/>, 2024.
- [36] Tongyi Lab. Msagent. arXiv preprint arXiv:2309.00986, 2023.
- [37] Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function-calling leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html, 2024.
- [38] Yujia Qin, Shihao Liang, Yining Ye, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In International Conference on Learning Representations (ICLR), 2024. arXiv:2307.16789.
- [39] Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. arXiv preprint arXiv:2605.23904, 2026.
- [40] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In Text Summarization Branches Out. Association for Computational Linguistics (ACL), 2004.

A Supplementary Formulations

A.1 Policy Decisions and Sandbox Transitions

Let π_{gen} denote the trajectory-generation policy used for offline data construction in Section 3. At turn t , it conditions on the product prompt D_P , prior interaction history h_t , current observation $o_t = (x_t, v_t, r_t, e_t)$, and previous task state Q_{t-1} . It generates an updated task state and an action:

$$(Q_t, a_t) \sim \pi_{\text{gen}}(\cdot \mid D_P, h_t, o_t, Q_{t-1}), \quad (\text{A.1})$$

where h_t excludes o_t ; the history and previous queue are empty before the first decision. Observation components follow Section 3, with $\emptyset$ denoting absence. Queue variables are omitted when unused. Online rollouts substitute the current student for π_{gen} .

For product P , the backend $\mathcal{B}_P$ executes action a_t in hidden state s_t , accounting for triggered events ϵ_t and their timing. It returns the result r_{t+1} , error e_{t+1} , and state changes Δs_t ; F_P applies these changes and any remaining event effects. When visual feedback is produced,

$$\begin{aligned} (r_{t+1}, e_{t+1}, \Delta s_t) &= \mathcal{B}_P(a_t, s_t, \epsilon_t), \\ s_{t+1} &= F_P(s_t, a_t, \Delta s_t, \epsilon_t), \\ v_{t+1} &= G_{\text{vis}}(v_t, s_t, s_{t+1}). \end{aligned} \quad (\text{A.2})$$

The visual generator uses simulation, interface rendering, or image generation; its output is checked against s_{t+1} . Dialogue replies x_{t+1} come from the user simulator, while read-only calls cause no action-induced state changes. These branches need not produce visual feedback. Hidden states and event schedules remain unavailable to the policy.

A.2 Supervised Learning Objective

Using the notation in Section 4.1, we minimize the average negative log-likelihood of the supervised assistant tokens:

$$\mathcal{L}_{\text{SFT}}(\theta) = \mathbb{E}_{\tau_i \sim D_{\text{SFT}}} \left[\sum_{n \in \mathcal{M}_i} -\log \pi_{\theta}(y_{i,n} \mid c_{i,n}) \right]. \quad (\text{A.3})$$

The mask $\mathcal{M}_i$ includes assistant reasoning, responses, actions, and explicit task queues when present, while excluding padding. Structural delimiters such as tags and brackets are supervised together with the surrounding content tokens and carry no separate weight. User messages, observations, and tool returns contribute to $c_{i,n}$ but are not prediction targets. Equation (A.3) sums over the supervised positions of each trajectory and then averages over D_{SFT} , so a demonstration with more supervised tokens contributes proportionally more. We keep this convention for the token-level objectives in Section 4.1 and Section 4.2, and average within each trajectory for the DPER policy loss (Appendix A.3), which weights a single trajectory-level advantage: summing over tokens there would multiply each update by the length of its trajectory and let long trajectories dominate the policy gradient.

A.3 DPER Policy Objective

For same-task rollout groups with $K > 1$, we set all base advantages to zero when the group rewards are identical.

Using the evidence-weighted token advantage $\hat{A}_{in}$ from Equation (5), we optimize the unprivileged policy with the ratio $\rho_{in}(\theta) = \pi_{\theta}(y_{i,n} \mid c_{i,n})/p_{0,in}$, where $p_{0,in}$ is the rollout probability defined in Equation (2). The objective is

$$\begin{aligned} \mathcal{J}_{\text{DPER}}(\theta) &= \mathbb{E}_i \left[\frac{1}{|\mathcal{M}_i|} \sum_{n \in \mathcal{M}_i} \mathcal{U}_{in}(\theta) \right], \\ \mathcal{U}_{in}(\theta) &= \min \left(\rho_{in}(\theta) \hat{A}_{in}, \text{clip}(\rho_{in}(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_{in} \right), \end{aligned} \quad (\text{A.4})$$

where ϵ is the policy-ratio clipping radius. The sum and normalization include only valid assistant-token positions $\mathcal{M}_i$, and the expectation averages over task instances and their sampled trajectories. The rollout probabilities, privileged scores, routing decisions, and token advantages are detached and held fixed during updates on a rollout batch. They are recomputed using the next student snapshot and newly collected rollouts. Gradients update only the unprivileged policy.

A.4 GPRM Grade Normalization

For reasoning step s , we apply softmax only to the logits of the four grade tokens $\mathcal{G} = \{A, B, C, D\}$ at the fixed judgment position after the generated verification rationale. The resulting probabilities $p_{s,g}$ sum to one over these four grades. These probabilities enter the expected-grade reward in Equation (7), which uses weights $(1, \frac{2}{3}, \frac{1}{3}, 0)$ for grades A through D and averages step rewards within each trajectory.

A.5 Reward Availability and Scoring Paths

Failure to construct the scoring input, including missing prompt data, directly returns $R_i = 0$. With valid input but no evaluable process steps, GPRM is unavailable; the scorer skips GPRM, Judge, and Queue calls and returns $0.1 \ell_{\text{format},i} \ell_{\text{len},i}$. A failed GPRM request or a missing or invalid score uses the same fallback and skips Judge and Queue. A valid GPRM score, including zero, instead enters normal aggregation in Equation (6), with β_{GPRM} retained in the denominator; Judge and Queue are evaluated when applicable. Normal aggregation excludes unavailable optional scores and retains valid zeros from every branch. These availability checks are distinct from process-quality judgments. The group-level hard-failure exclusion in Section 4.4 still takes precedence over policy updates.

A.6 Task-Queue Scoring

Section 4.4 combines semantic agreement with queue continuity. Let F_{inv} reconstruct the intent expressed by a queue under task-granularity rules $\mathcal{R}_{\text{gran}}$. In both initial and feedback-turn scoring, sim encodes the compared texts with Qwen3-Embedding-4B and linearly maps their cosine similarity from $[-1, 1]$ to $[0, 1]$.

Initial-queue scoring. On the first turn, the reconstructed intent is compared with the initial request x :

$$\begin{aligned}\hat{x}_{\text{queue}}^{(1)} &= F_{\text{inv}}(Q^{(1)}; \mathcal{R}_{\text{gran}}), \\ q_{\text{queue}}^{(1)} &= \text{sim}(\hat{x}_{\text{queue}}^{(1)}, x).\end{aligned}\tag{A.5}$$

This comparison evaluates intent coverage without requiring an exact match to a reference queue. For queues containing coarse-grained action labels, the score reflects task-type coverage rather than detailed argument correctness or task completion. No continuity score is used on the first turn.

Availability and missing outputs. Initial and feedback-turn scores use Equations (A.5) and (9), respectively, with weights from Section 5.1. Available components are reweighted to sum to one, retaining evaluated zeros. A turn has no Queue score if all components are unavailable or the output lacks the task-queue field; the format factor separately assesses required fields. These rules follow the early-exit checks in Appendix A.5.

Queue inverse rewriting. The inverse rewrite preserves the listed tasks, any arguments present, batch structure, and execution-order or parallelism constraints, without adding, omitting, or correcting tasks. It uses only the queue itself. We use JoyAI-VL for this rewriting in both initial and feedback-turn scoring.

Table 2 Illustrative TaskQueue inverse-rewriting prompt for JoyAI-VL. The placeholder `{task_queue}` is replaced with the current queue.

TaskQueue inverse rewriting
Rewrite the following TaskQueue as a concise natural-language description of the tasks it specifies. Preserve all tasks, arguments, batch structure, and execution-order or parallelism constraints. Do not add, omit, or correct tasks. Return only the description. TaskQueue: {task_queue}

For a turn with a nonempty observation block and a preceding queue, let $o^{(1:r)} = (o^{(1)}, \dots, o^{(r)})$ denote the full observation history available before the current queue update, including user messages, visual observations, execution feedback, and errors. Given the initial request x and this history, F_{obs} rewrites the task requirements to reflect the remaining tasks after accounting for completed work, user revisions, and cancellations. The semantic score compares the updated intent with the intent reconstructed from the current queue:

$$\begin{aligned}
\hat{x}_{\text{obs}}^{(r)} &= F_{\text{obs}}(x; o^{(1:r)}), \\
\hat{x}_{\text{queue}}^{(r)} &= F_{\text{inv}}(Q^{(r)}; \mathcal{R}_{\text{gran}}), \\
q_{\text{obs}}^{(r)} &= \text{sim}(\hat{x}_{\text{obs}}^{(r)}, \hat{x}_{\text{queue}}^{(r)}).
\end{aligned} \tag{A.6}$$

To measure continuity, let $E^{(r-1)}$ be the verified execution outcome for the previous head batch $B_1^{(r-1)}$, with status $\nu(E^{(r-1)})$ equal to complete, incomplete, or partial. Incomplete includes failed or unexecuted heads; ν is determined from execution feedback rather than the queue proposed by the policy. Let $[B_{\text{rem}}^{(r-1)}]$ contain the nonempty unfinished part of the previous head after partial execution, as determined by $E^{(r-1)}$, preserving the original relative order of the remaining tasks. With $\parallel$ denoting queue concatenation, define the execution-conditioned reference queue as

$$Q_{\text{ref}}^{(r)} = \begin{cases} [], & Q^{(r-1)} = [], \\ \text{Tail}(Q^{(r-1)}), & \nu(E^{(r-1)}) = \text{complete}, \\ Q^{(r-1)}, & \nu(E^{(r-1)}) = \text{incomplete}, \\ [B_{\text{rem}}^{(r-1)}] \parallel \text{Tail}(Q^{(r-1)}), & \nu(E^{(r-1)}) = \text{partial}. \end{cases} \tag{A.7}$$

Let M_r count batch pairs that match exactly at corresponding positions in $Q_{\text{ref}}^{(r)}$ and $Q^{(r)}$. A pair matches only when both batches contain the same number of tasks and identical task entries in the same order. Matching permits no reordering of batches or their tasks, including parallel tasks. In the coarse-grained representation, entries are action labels rather than tool-call arguments. Semantic similarity is evaluated separately. When at least one queue is nonempty, the continuation score is

$$q_{\text{cont}}^{(r)} = \frac{M_r}{\max\{|Q_{\text{ref}}^{(r)}|, |Q^{(r)}|\}}, \tag{A.8}$$

where queue length counts task batches. The score is defined as 1 when both queues are empty. Equation (9) combines this score with semantic agreement on feedback turns.

Cross-turn aggregation. The trajectory-level TaskQueue reward averages all evaluated turn scores, including zeros. For a single-turn trajectory, it equals the initial score when evaluable. If no turn is evaluable, normal content aggregation omits the Queue component and renormalizes the remaining weights.

B Product Prompt and Interaction Templates

Table 3 presents an English structural template of the product developer prompt. The prompt specifies five elements: role definition, input specification, output requirements, capability registry, and execution policies. In the notation of Section 3, the role and communication interface instantiate I_P , the capability registry instantiates S_P , and the execution policies encode W_P and C_P . Placeholders (enclosed in curly braces) represent configuration-specific content, including the instruction source and output recipient; concrete tool definitions and policy instances are omitted.

Table 3 Product developer prompt template with configurable role, interaction interface, capabilities, and execution policies.

Developer prompt structure
1. Role Definition Role and embodiment: {role_definition}. Execute assigned tasks using the registered capabilities and prescribed execution policies. 2. Input Specification Instruction source: {instruction_source}. Task representation: {task_input_schema}. 3. Output Requirements Recipient: {output_recipient}. Style: objective, concise, and factual. Execution report format: {execution_report_schema}. 4. Capability Registry For each registered capability, specify: Name: {capability_name}. Description: {capability_description}. Parameters: {parameter_schema}, including types, units, and default values where applicable. 5. Execution Policies For each applicable policy, specify: Policy name: {policy_name}. Execution procedure and conditions: {execution_rules}. Operational constraints: {applicable_constraints}.

B.1 Chat-Template Serialization

The developer prompt is the content of a distinct **developer** message. Table 4 reproduces the corresponding Jinja branch and illustrates the resulting input sequence. The role name follows `<|im_start|>`, and the message ends with `<|im_end|>`; the product specification is supplied as text within these boundaries. This serialization distinguishes the shared system instructions from the product specification without changing the message delimiters. Tool results are serialized inside `<tool_response>` blocks within user messages, preserving their source through these tags.

Table 4 Developer-role serialization. Top: the exact branch from the chat template. Bottom: a representative input with `add_generation_prompt` enabled and thinking enabled. Braces denote substituted content; optional tool definitions, visual-token expansion, and interaction history are omitted.

Developer role in the chat template
Jinja branch <pre>{%- elif message.role == "developer" %} {%- set dev_content = render_content(message.content, true) %} {{- '< im_start >developer\n' + dev_content + '< im_end >' + '\n' -}}</pre> Serialized model input <pre>< im_start >system {shared_interaction_protocol}< im_end > < im_start >developer {product_specification}< im_end > < im_start >user {task_request_and_multimodal_content}< im_end > < im_start >assistant <think></pre>

B.2 ReAct Assistant Output Template

Table 5 illustrates the ReAct decisions described in Section 4.1. A single assistant message combines the shared prefix with either continuation. The `task` block is included in either branch when required by the interaction protocol and omitted otherwise. Tool execution feedback conditions the next policy decision; direct responses address the user without issuing a tool call.

Table 5 Illustrative ReAct assistant output. The shared prefix is followed by one of the two continuations. Braced entries are placeholders; the `task` block is included when required by the interaction protocol.

ReAct assistant output: tool call or direct response
Shared prefix <pre><think>{reasoning based on the current context}</think> <task> [["{current_action_label}"], ["{pending_action_label}"]] </task></pre> Continuation A: tool call <pre><tool_call> {"name": "{tool_name}", "arguments": {"{argument_name}": "{argument_value}"}} </tool_call></pre> Continuation B: direct response <pre>{answer, clarification, or refusal addressed to the user}</pre>