CRUCIBLE: COUNTERFACTUAL RLVR FOR DEXTEROUS MANIPULATION PLANNERS

Anonymous authors
Paper under double-blind review

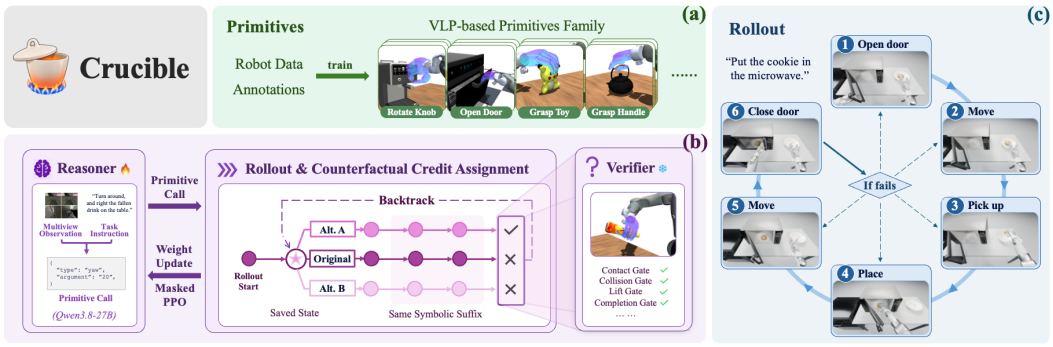


Figure 1: CRUCIBLE is a counterfactual RLVR framework that trains the Reasoner using verifiable physical feedback. (a) Using robot demonstrations collected through teleoperation, we train a vision–language–primitive model (VLP) to provide arm–hand targets for executing manipulation primitives. (b) During training, CRUCIBLE revisits earlier decisions in failed rollouts and compares physical outcomes of alternative calls from the same saved state. These comparisons guide masked PPO updates to the Reasoner, while the execution stack and Verifier remain fixed. (c) At deployment, the Reasoner selects primitives using task instructions and current observations; rule-based controllers connect actions into closed-loop manipulation sequences.

ABSTRACT

Combining vision-language planners with control primitives is a promising paradigm for robotic manipulation. However, reliable execution still requires high-level planners to improve from the outcomes of their own actions, particularly when hallucinations or incorrect reasoning lead to failures. Learning from such feedback is challenging because a primitive may appear locally correct but cause failure many steps later, while a task-level failure signal does not reveal which decision should be corrected. This lack of *credit assignment* makes planner learning inefficient. We present CRUCIBLE, a reinforcement learning from verifiable rewards (RLVR) framework with *counterfactual credit assignment*. CRUCIBLE builds on a new vision-language-primitive (VLP) model constructed from robot demonstrations, with each primitive paired with a privileged verifier. For a failed trajectory, our counterfactual RLVR algorithm intervenes on individual primitive decisions from saved states and replays the recorded suffix, identifying decisions whose replacement repairs the observed verifier failure under fixed-suffix replay. Full-episode returns from these alternative executions then define advantages for masked PPO that updates only the identified decision tokens. We evaluate cross-step dependencies in tabletop grasping, where base motions affect the feasibility of a later grasp. A replay probe rescues 49.5% of eligible failures, demonstrating the availability of executable repairs. In closed-loop evaluation, counterfactual PPO achieves 62.38% overall success, compared with 55.45% for call-level episode-return PPO and 56.44% for the shared SFT baseline. Project Page: cruciblex2026.github.io.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Dexterous manipulation is a key capability for general-purpose robots, yet learning policies that generalize across objects, scenes, and tasks remains difficult. Recent flow-based models learn continuous action chunks from egocentric human video, tactile-rich demonstrations, and cross-embodiment robot data (Zheng et al., 2026; Niu et al., 2026; Wu et al., 2026). Yet their generalization remains limited. Long-horizon dexterous manipulation therefore benefits from a high-level planner that maps language instructions and fresh visual observations to semantic primitive calls, while a fixed execution stack grounds and executes each call. This hierarchy supports task composition, but existing systems primarily use high-level reasoners for planning, search, or execution-time correction (Qi et al., 2025; Ravichandran et al., 2025; Xiao et al., 2026), without turning verified physical outcomes into localized updates that improve the planner's primitive decisions.

RLVR trains policies with deterministic, programmatic reward checks. In digital-agent sandboxes, such checks can match a purchased item against requested attributes or execute generated code and inspect the resulting state (Yao et al., 2022; Yang et al., 2023). In embodied planning, analogous checks can reject LLM-generated plans that violate reachability or collision constraints and reward only physically valid, task-completing plans (Ji et al., 2025). However, a trajectory-level reward indicates only whether the overall plan succeeded, not which primitive decision should change: a grasp may fail because an earlier base-motion call left the object outside the arm's reachable workspace. Counterfactual comparisons and sandbox branching (Chen et al., 2026; He et al., 2026) suggest testing alternative decisions from a shared state.

We present CRUCIBLE, an RLVR framework with counterfactual credit assignment for learning primitive decisions through saved-state physical interventions. CRUCIBLE builds on a vision–language–primitive (VLP) model that we train from robot demonstrations for prehensile and non-prehensile interactions, together with a common primitive interface that aligns execution, verification, and state snapshots. A vision-language reasoner observes four fresh RGB views and emits one next call. The fixed VLP executor maps RGB observations and semantic primitive requests to arm–hand targets and executes them through classical planning and control. Each primitive is paired with fixed physical checks evaluated by a privileged verifier: the verifier reads simulator state, while the reasoner receives images and task context. Figure 1 illustrates the interface among the reasoner, VLP executor, and verifier, together with the rollout process used for counterfactual training.

Starting from a Qwen3.8-27B reasoner (Qwen Team, 2026) initialized by supervised fine-tuning (SFT) on robot primitive-decision examples and general visual-instruction data, CRUCIBLE searches backward from a failed call. At each saved state, it replaces one call and replays the recorded symbolic suffix, re-grounding subsequent calls in the resulting branch. The nearest sampled replacement that restores every physical check through the failure selects the decision to refine. Full episode returns from sibling branches define leave-one-out advantages for PPO (Schulman et al., 2017) masked to selected call tokens. After each masked-PPO update, the updated Reasoner collects fresh rollouts to construct the next round of counterfactual training data. The VLP Executor and Verifier remain fixed, so optimization changes only the Reasoner's weights.

A closed-loop tabletop grasping benchmark tests cross-step dependencies: earlier base motions determine the conditions for a later grasp. Paired random-start episodes cover seen and held-out objects. We compare rescue distillation, episode-return PPO, and counterfactual PPO as alternative updates of the same SFT reasoner under a shared VLP executor. Supporting tests examine observation context, training recipes, and general capabilities. A replay probe finds full-return rescues in 49.5% of eligible failures, demonstrating supervision availability. Hardware deployment connects RGB-conditioned primitive selection to robot execution.

Our contributions are as follows:

- A counterfactual RLVR algorithm that localizes sampled single-call repairs and constructs full-return sibling advantages for masked reasoner optimization.
- An integrated primitive-based manipulation system that connects teleoperated data collection, VLP learning, verifiable execution, and saved-state replay, supporting high-level policy learning and RGB-conditioned robot deployment.
- A systematic evaluation of primitive-based manipulation, examining reasoner optimization, observation context, training data, and VLP distillation through controlled comparisons at both the primitive and closed-loop task levels.

2 Related Work

Semantic dexterous primitives. Dexterous grasps can be optimized analytically or learned from large corpora (Liu et al., 2021; Turpin et al., 2022; Wang et al., 2023; Ye et al., 2025); recent generators amortize synthesis at inference (Xu et al., 2023; Weng et al., 2024; Huang et al., 2026). Semantic conditioning ranges from grasp types to free-form instructions (Chen et al., 2025; Li et al., 2024). Language-conditioned generators such as DexVLG and OmniDexVLG (He et al., 2025; Zhang et al., 2025) provide semantic grasp targets. Our grasp VLP provides pregrasp and goal configurations for the arm and hand within a fixed execution stack. The resulting primitive boundary supplies both a physical outcome and a saved state for intervening on the reasoner's decision.

Language-grounded primitives and action representation. Language can specify policy code, value maps, or keypoint and orientation constraints that classical solvers execute (Liang et al., 2023; Huang et al., 2023; 2024; Qi et al., 2025); static-goal predictors likewise delegate motion (Zeng et al., 2020; Shridhar et al., 2021). Such postcondition-bounded actions connect options to task and motion planning (Sutton et al., 1999; Garrett et al., 2021). VLA and world-action models instead learn temporal action sequences (Black et al., 2025; NVIDIA, 2026b; Ye et al., 2026); PrimitiveVLA retains fixed-length action chunks (Li et al., 2026). CRUCIBLE represents each learned action by two arm–hand keyframes and a termination condition; fixed planning supplies the trajectory, and the high-level reasoner continues learning after supervised initialization.

Verifier-guided credit assignment and policy optimization. RLVR improves a generator against fixed programmatic checks (Lambert et al., 2024; Guo et al., 2025), including physical checks in simulation (Ji et al., 2025). An episode check, however, does not identify the decision to refine. C3 assigns counterfactual credit among language-agent contributions (Chen et al., 2026); BPO branches sibling action rollouts from restored sandbox states and trains on their leave-one-out advantages (He et al., 2026). CRUCIBLE instead backtracks from a rejected primitive and intervenes on that call and its predecessors while holding the emitted suffix and evaluation conditions fixed. Parameter updates for language-model policies have been studied with proximal policy optimization and related policy-gradient objectives (Schulman et al., 2017; Shao et al., 2024). Whereas PRISM distills a planner once and ENPIRE keeps its coding agents fixed while refining an external policy (Ravichandran et al., 2025; Xiao et al., 2026), CRUCIBLE uses primitive-level counterfactual returns to construct targeted weight-update supervision. Its credit-assignment rule selects the call spans and advantages used by PPO.

3 Method

CRUCIBLE contains three components: a **Reasoner** that selects one semantic primitive call from the task context and current observations, a fixed **VLP Executor** that executes the call, and a privileged **Verifier** that checks its physical outcome. During a rollout, the Reasoner selects one call at a time, the VLP Executor executes it, and the Verifier checks the result. Before each call, the system records the current context and a point from which execution can later resume. When the Verifier detects a failure, CRUCIBLE examines the failed call and then earlier calls in reverse order. For each one, it resumes execution from before that call, substitutes an alternative, and re-executes the remaining calls already produced by the Reasoner. A replacement that resolves the failure identifies the call to correct and provides targeted feedback for training. The updated Reasoner then collects new rollouts, and the process repeats.

Section 3.1 introduces the three system components and their closed-loop interaction, corresponding to Figure 1(c). Section 3.2 describes the rule-based and VLP-based execution shown in Figure 1(a), and Section 3.3 describes the initialization of the Reasoner. Section 3.4 explains how failed rollouts are converted into targeted training signals, and Section 3.5 explains how those signals update the Reasoner, together corresponding to Figure 1(b).

3.1 System components

The vision–language Reasoner π_θ maps the task context and current observations to a semantic primitive call. The VLP Executor Exec carries out that call through learned arm–hand target pre-

diction, motion planning, and rule-based control. The privileged Verifier V then checks the physical outcome using simulator state that is unavailable to the Reasoner.

Treating each completed primitive as one decision step, we model this closed-loop interaction as a finite-horizon partially observable Markov decision process (POMDP). Given a task instruction g, the Reasoner uses context $h_k = (g, o_{k-1})$, where o_{k-1} contains the latest four RGB views and target-object information. The full physical state s_k remains hidden from the Reasoner. At decision step k,

$$a_k \sim \pi_\theta(\cdot \mid h_k), \qquad s_k = \mathrm{Exec}(s_{k-1}, a_k), \qquad v_k = V(s_{k-1}, a_k, s_k). \tag{1}$$

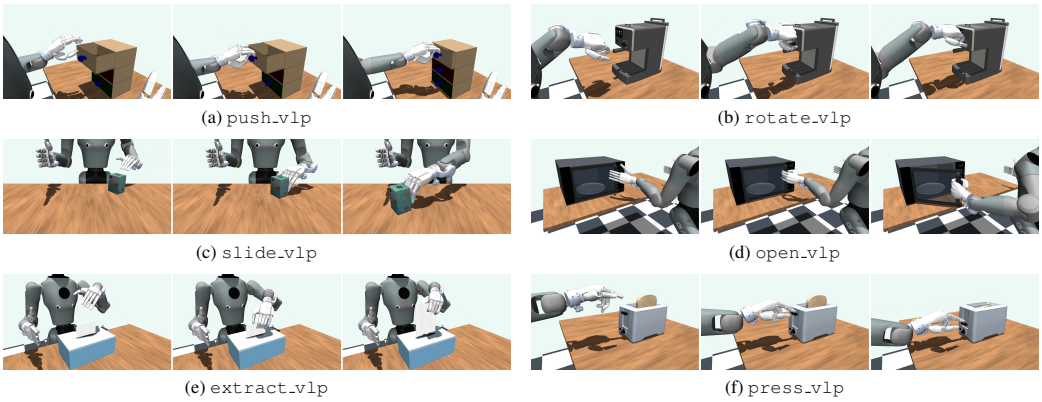
The Reasoner emits the primitive call a_k, the VLP Executor advances the system from s_{k-1} to s_k, and the Verifier records the outcome v_k. The Verifier's full record is retained offline and excluded from o_k. Task-specific rules map the physical fields in v_k to a local acceptance bit $\mathrm{pass}(v_k)$ and aggregate $v_{1:K}$ into the episode return R_V. The same success criteria terminate the primitive and define the acceptance bit; timeout or safety termination sets it to zero. The bit marks where failure became visible, not whether an earlier call caused it. Appendix C.3 details the Verifier record and reward construction.

3.2 Primitive Execution

At decision step k, the Reasoner emits one schema-constrained primitive call a_k. The VLP Executor converts the call into a primitive-level target specification z_k: it is constructed deterministically for rule-based calls and predicted from four calibrated RGB views, the primitive request, and the active-hand identifier for VLP-based calls. For grasping, z_k consists of pregrasp and grasp arm–hand keyframes. The deployed RGB student is trained with both ground-truth flow matching and the flow predictions of a geometry-conditioned teacher that additionally receives the target-object point cloud (Appendix B.4). The target specification is produced once per primitive; a primitive-specific controller then expands it into dense commands,

$$\begin{aligned} s_{k,0} &= s_{k-1}, \\ u_{k,t} &= \mathcal{C}_{a_k}(s_{k,t-1}; z_k), \qquad s_{k,t} = f_{\mathrm{phys}}(s_{k,t-1}, u_{k,t}), \quad t = 1, \dots, T_k, \\ \mathrm{Exec}(s_{k-1}, a_k) &= s_{k,T_k} = s_k, \end{aligned} \tag{2}$$

where $\mathcal{C}_{a_k}$ includes collision-checked motion planning, interpolation, and rule-based control, and f_{phys} is one physical control-step transition. The VLP Executor therefore produces a command and resulting state at every control step without another Reasoner or VLP query. A call ends when its success criterion is met or an execution failure, timeout, or safety condition is reached. A failed call terminates the rollout. After a successful nonterminal call, the client captures four fresh RGB views for the next decision. Appendix F.4 details the execution procedures. Figure 2 illustrates six non-prehensile VLP-based calls.



(a) `push_vlp` (b) `rotate_vlp` (c) `slide_vlp` (d) `open_vlp` (e) `extract_vlp` (f) `press_vlp`

Figure 2: Representative simulated executions of six non-prehensile VLP-based primitives.

3.3 INITIALIZATION OF THE REASONER

We initialize the Reasoner from Qwen3.8-27B (Qwen Team, 2026) by SFT on $\mathcal{D}_{\text{init}} = \mathcal{D}_{\text{call}} \cup \mathcal{D}_{\text{general}}$. Robot examples in $\mathcal{D}_{\text{call}}$ pair visual and task context with a rationale and a target primitive call. General visual-instruction examples in $\mathcal{D}_{\text{general}}$ come from Vision-Flan (Xu et al., 2024). Each optimizer update uses a batch from one source; robot and general batches receive 80% and 20% of updates, respectively. For a full distributed batch $\mathcal{B}$ of input contexts x and target responses $y = (y_1, \ldots, y_{|y|})$, the initialization objective is

$$\mathcal{L}_{\text{SFT}}(\theta; \mathcal{B}) = -\frac{\sum_{(x,y)\in\mathcal{B}} \sum_{\ell=1}^{|y|} \log \pi_\theta(y_\ell \mid x, y_{<\ell})}{\sum_{(x,y)\in\mathcal{B}} |y|}. \tag{3}$$

The loss averages only target response tokens across the distributed batch. The resulting SFT checkpoint θ_{ref} initializes policy optimization and remains its fixed KL reference. Appendix C.1 describes data collection, and Appendix C.2 specifies the training sets, SFT settings, and training costs.

3.4 CONSTRUCTING PRIMITIVE-LEVEL SUPERVISION

Verifier-observed failure events. Policy optimization starts from $\theta^{(0)} = \theta_{\text{ref}}$. At round r, we freeze the behavior Reasoner $\pi_b \leftarrow \pi_{\theta^{(r)}}$ and collect fresh reference executions $a_{1:K}$. Before each call a_k, we save its Reasoner context h_k and a snapshot σ_{k-1} containing the simulator and random-number-generator state. The Verifier records physical checks, planning status, contacts, safety violations, and timeouts. For every unsuccessful episode, we collect all calls rejected by their physical acceptance criteria,

$$\mathcal{F} = \{t \in \{1, \ldots, K\} : \text{pass}(v_t) = 0\}. \tag{4}$$

A rejected call terminates the reference rollout. Its index t marks where V observes the failure; an earlier decision may still be the one to refine. An unsuccessful episode with $\mathcal{F} = \emptyset$ supplies no primitive-level update. A snapshot is eligible only if restoring it and replaying the recorded calls reproduces the reference execution; failed fidelity checks are discarded rather than attributed.

Backward saved-state intervention. For a failed call t, we inspect $k = t, t-1, \ldots, 1$. At each k, the observed call $a_{k,1} = a_k$ and $J-1$ alternatives form a sibling group. Policy-sampled candidates are independent schema-constrained calls $a_{k,j} \sim \pi_b(\cdot \mid h_k)$. For sibling j, restore the pre-call snapshot σ_{k-1}, inject $a_{k,j}$, and replay the recorded symbolic suffix $a_{k+1:K}$ without querying the Reasoner. The VLP Executor executes each later call from the resulting branch state. Let $v_i^{(k,j)}$ be the resulting Verifier record at call i. An execution failure terminates the branch with zero return; any remaining suffix calls receive zero acceptance bits. We define

$$\begin{aligned} y_{k,j}^{(t)} &= \prod_{i=k}^{t} \text{pass}\left(v_i^{(k,j)}\right), \\ k_t^\star &= \max\left\{k \le t : \max_{j=2,\ldots,J} y_{k,j}^{(t)} = 1\right\}. \end{aligned} \tag{5}$$

Because the original branch fails at t, $y_{k,1}^{(t)} = 0$. Equation 5 selects the closest decision at or before t for which a single replacement makes every call through t pass. If the set is empty, the failure remains unlocalized and produces no call-level update. The result is the nearest sampled single-call repair, not a claim of a unique causal root.

Sibling return and advantage. For a localized failure, set $k = k_t^\star$. Each of the J replayed siblings also yields a full physical return and a leave-one-out (LOO) advantage computed against the mean return of the other siblings:

$$\begin{aligned} \widehat{Q}_{k,j} &= R_V\left(v_{1:k-1}, v_{k:K}^{(k,j)}\right), \\ \widehat{A}_{k,j} &= \widehat{Q}_{k,j} - \frac{1}{J-1} \sum_{j' \neq j} \widehat{Q}_{k,j'}. \end{aligned} \tag{6}$$

Here $v_{1:k-1}$ is the unchanged reference prefix, and $v_{k:K}^{(k,j)}$ contains the replacement call's Verifier record followed by those of the replayed calls. Thus $\widehat{Q}_{k,j}$ scores the complete episode, while $\widehat{A}_{k,j}$ is its leave-one-out contrast with the other siblings (Ahmadian et al., 2024). The local rescue indicator $y_{k,j}^{(t)}$ identifies the decision to repair, whereas the full return weights that repair and additionally requires completion of the terminal task.

The attributed set $\mathcal{G} = \{k_t^\star : t \in \mathcal{F},\ k_t^\star \text{ exists}\}$ contains each localized decision step once, together with its sibling replays and diagnostics. Appendix C.4 details the collection protocol.

3.5 Counterfactual data for weight optimization

For each localized decision step $k \in \mathcal{G}$, let τ_k be the Reasoner-visible reference trace and let $d_{k,j}$ contain the physical Verifier diagnostics from sibling j, including the failure events assigned to k. The counterfactual record is

$$\mathcal{D}_{\mathrm{CF}} = \{(\tau_k, \{(a_{k,j}, \widehat{Q}_{k,j}, \widehat{A}_{k,j}, d_{k,j})\}_{j=1}^{J}) : k \in \mathcal{G}\}. \tag{7}$$

Masked PPO on attributed call spans. We use *masked PPO* for the clipped PPO objective (Schulman et al., 2017) with the policy term and KL penalty evaluated only on tokens encoding attributed primitive calls. The recorded context remains the policy input. Token-level loss masking is also used in Search-R1 and AdaCoT (Jin et al., 2025; Lou et al., 2025); here, saved-state physical interventions determine the selected call spans and sibling-return advantages. At PPO round r, the collection procedure uses $\pi_b = \pi_{\theta^{(r)}}$ to produce $\mathcal{D}_{\mathrm{CF}}^{(r)}$. The round-$r$ weights $\theta^{(r)}$ therefore supply the probability-ratio denominator, while the reference Reasoner θ_{ref} remains the fixed KL reference. Let $\mathcal{T}_{k,j}$ be the token span that emits $a_{k,j}$ and define

$$r_{k,j,\ell}(\theta') = \frac{\pi_{\theta'}(x_\ell \mid h_k, x_{<\ell})}{\pi_{\theta^{(r)}}(x_\ell \mid h_k, x_{<\ell})}. \tag{8}$$

The corresponding masked objective is

$$\mathcal{J}_{\mathrm{PPO}}(\theta') = \mathbb{E}_{\mathcal{R}^{(r)}}\Bigg[\frac{1}{|\mathcal{T}_{k,j}|}\sum_{\ell \in \mathcal{T}_{k,j}} \min\Big(r_{k,j,\ell}(\theta')\widehat{A}_{k,j}, \mathrm{clip}(r_{k,j,\ell}(\theta'), 1-\epsilon_{\mathrm{clip}}, 1+\epsilon_{\mathrm{clip}})\widehat{A}_{k,j}\Big) - \beta D_{\mathrm{KL}}^{\mathcal{T}_{k,j}}(\pi_{\theta'} \,\|\, \pi_{\theta_{\mathrm{ref}}})\Bigg]. \tag{9}$$

Here $\mathcal{R}^{(r)} \subseteq \mathcal{D}_{\mathrm{CF}}^{(r)}$ contains the policy-sampled siblings with valid action spans and behavior log-probabilities; the index k of a record is its localized decision step. The normalization group is a second, coarser partition than the sibling set: the eligible records of one episode form a group $\mathcal{B}_e$, and we normalize the advantages within it,

$$\widehat{A}_{k,j} \leftarrow \frac{\widehat{A}_{k,j} - \mathrm{mean}_{\mathcal{B}_e}(\widehat{A})}{\mathrm{std}_{\mathcal{B}_e}(\widehat{A}) + \epsilon}, \tag{10}$$

where $\epsilon > 0$ is a numerical stabilizer. This centers the group advantages at zero. A group whose returns do not spread contributes no return-based policy gradient; because such a group is still in the batch, its KL term continues to act. The loss is masked to the attributed call. Each round initializes the optimization at $\theta' = \theta^{(r)}$ and uses its newly collected batch for a bounded number of minibatch epochs, producing $\theta^{(r+1)}$. The updated Reasoner then becomes the behavior policy that collects the next round's reference rollouts and siblings. Algorithm 1 summarizes the successive collection and PPO rounds, starting directly from the SFT checkpoint θ_{ref}.

Algorithm 1 Counterfactual supervision collection and masked PPO update

fixed: environments, Verifier V, VLP Executor Exec, θ_{ref}
initialize $\theta^{(0)} \leftarrow \theta_{\text{ref}}$
for $r = 0, \ldots, R-1$ optimization rounds **do**
  freeze $\pi_b \leftarrow \pi_{\theta^{(r)}}$; collect fresh rollouts and save pre-call states and traces
  $\mathcal{D}_{\text{CF}}^{(r)} \leftarrow$ localize failures, replay siblings, and compute returns and LOO advantages by Eqs. 4–7
  $\mathcal{R}^{(r)} \leftarrow$ policy-sampled records in $\mathcal{D}_{\text{CF}}^{(r)}$ with valid spans and behavior log-probabilities
  $\theta^{(r+1)} \leftarrow \theta^{(r)}$
  if $\mathcal{R}^{(r)} \neq \emptyset$ **then**
    $\theta^{(r+1)} \leftarrow \text{PPO}(\theta^{(r)}, \mathcal{R}^{(r)})$ using Eq. 9
  end if
end for
return $\theta^{(R)}$

4 EXPERIMENTS

We evaluate CRUCIBLE at both the primitive and closed-loop task levels. With a shared executor and Verifier, we compare Reasoner baselines, SFT, rescue distillation, and PPO, and examine observation context and training recipes. VLP ablations compare geometry-conditioned prediction, GT-only RGB learning, and RGB distillation, measuring physical success for individual primitives and complete tasks. General-capability tests and robot deployment further characterize the learned Reasoner and the integrated system. Appendix D.4 reports separate single-primitive tests for the non-prehensile VLPs illustrated in Figure 2.

4.1 EXPERIMENTAL SETUP

The primary reasoner comparison tests cross-step dependencies between base translation, turning, and a subsequent grasp, pairing methods on the same randomized initial scenes with seen and held-out objects. Seen and held-out refer to the Reasoner SFT object split; VLP data use a separate split by source demonstration. Object positions and orientations are sampled from the empirical annotation-validation distribution, and robot base starts are randomized. All reasoners share the fixed distilled RGB VLP Executor, Verifier, motion controllers, and evaluation-time execution budgets. At each decision, the reasoner receives four fresh RGB views and emits one next primitive as JSON. Episodes allow at most 12 reasoner decisions and one physical grasp attempt.

Complete-task success requires all primitive calls to pass, completion of the grasp-and-lift protocol, and final object contact with the active hand only. The metric covers execution from the full initial scene. All sampled episodes enter the denominator, including invalid calls, planning failures, timeouts, and decision-cap exhaustion. We report success rates and paired differences in percentage points. Experimental settings are detailed in Appendix E.1.

4.2 CLOSED-LOOP TABLETOP GRASPING

Table 1 compares external vision-language reasoners before and after primitive-call SFT and Qwen3.8-27B variants obtained through SFT, rescue distillation, and policy optimization. All conditions share the current-observation interface and the paired evaluation design of Section 4.1. Execution and verification remain fixed while the reasoner and its training procedure vary. External-model results are system-level comparisons, with different architectures, training data, and adaptation budgets. Our episode-return PPO baseline assigns a minibatch-centred episode return to one call per episode: the verifier-observed failing call on failure, or the terminal call on success. Counterfactual PPO instead selects a call by backward replay and constructs advantages from same-state sibling returns. Their comparison evaluates the complete update procedures, including both call selection and advantage construction.

Reasoner variant	Seen ↑	Held-out ↑	Overall ↑
Cosmos3-Nano (pretrained)	0.00	0.00	0.00
+ SFT	26.42	31.25	28.71
RoboBrain2.5-8B-NV (pretrained)	0.00	0.00	0.00
+ SFT	35.85	41.67	38.61
Qwen3.8-27B (pretrained)	5.66	6.25	5.94
+ SFT (robot + general)	**66.04**	45.83	56.44
+ Rescue distillation	37.74	47.92	42.57
+ Episode-return PPO	56.60	54.17	55.45
+ **Counterfactual PPO (CRUCIBLE)**	62.26	**62.50**	**62.38**

Table 1: Complete-task physical success (%) on paired randomized scenes with seen and held-out objects; higher is better. The VLP Executor, Verifier, and execution budgets are fixed. External + SFT rows use primitive-call supervision (Appendix E.3). The pretrained Qwen baseline uses JSON-schema-constrained decoding; the SFT and rescue models use greedy decoding. Best scores are in bold.

Complete-task success is 5.94% for the pretrained reasoner and 56.44% for robot-plus-general SFT. Rescue distillation achieves 42.57%, and episode-return PPO achieves 55.45%. Counterfactual PPO reaches 62.38%, a difference of $+6.93$ percentage points relative to episode-return PPO and $+5.94$ points relative to SFT. Observation-context and training-recipe comparisons and general VLM tests appear in Appendix E.

Robot-plus-general SFT uses eight training GPUs for 68.90 hours (551.20 GPU-hours); Appendix C.2 details the SFT costs. Counterfactual PPO uses parallel rollout workers on eight GPUs, synchronized at round boundaries, and completes 8 rounds over 14.96 hours (119.68 GPU-hours); Appendix F.5 describes the framework and Appendix C.6 the update settings. After primitive-call SFT, Cosmos3-Nano and RoboBrain2.5-8B-NV achieve 28.71% and 38.61% overall success, respectively. Their adaptation settings and comparison scope are detailed in Appendix E.3. Robot-plus-general SFT scores 85.19% on MMBench, 87.17% on CV-Bench, and 67.21% on BLINK, compared with 85.01%, 84.60%, and 61.48% for the pretrained model. Table 17 reports the score changes.

Grasp-executor ablation. Table 2 compares a geometry-conditioned teacher, a GT-only RGB student, and a distilled RGB student. The students share architecture, initialization, training data, and optimization budget, and differ in teacher-flow supervision. Their comparison measures this supervision at the primitive and closed-loop levels while holding the remaining system components fixed. Teacher–student comparisons examine the tradeoff between geometric input and RGB-only deployment.

VLP	Cloud at inference	Teacher-flow loss	Primitive SR (%)	Closed-loop SR (%)
Geometry-conditioned teacher	Yes	—	79.95	72.28
RGB student, GT-only	No	No	67.87	47.52
RGB student, distilled	No	Yes	70.42	56.44

Table 2: VLP ablation with success rates in percent. Primitive tests start at the corrected predicted pregrasp; closed-loop tests include navigation and physical approach from paired randomized initial scenes with seen and held-out objects. Motion controllers, execution rules, the Verifier, and execution budgets are fixed.

4.3 COUNTERFACTUAL SUPERVISION AND OPTIMIZATION

Counterfactual replacement branches rescue 49.5% of eligible failed reference episodes. Of the executed replacement branches, 9.14% succeed, and 100% of tested replay boundaries pass the

replay-fidelity check. These probe statistics use the recorded collection executor; policy evaluation is reported separately. Episode rescue yield measures how often a failed reference admits a successful replacement; branch success measures the yield per executed candidate. Closed-loop performance is reported in Table 1. Appendices C.5, C.6, and E.2 describe the rescue corpus, PPO updates, and evaluation details, respectively.

4.4 Real-world deployment

On the Galbot G3, the trained reasoner selects a primitive from RGB observations and task context. For grasp calls, the distilled RGB VLP supplies arm–hand targets; motion planning and rule-based controllers execute and connect primitive calls. Fresh observations inform the next decision. Appendix F specifies the platform and execution details, and Appendix E.7 reports model-service timings.

5 Conclusion

Crucible addresses cross-step credit assignment in manipulation: a failed task reveals little about which earlier decision should change. The VLP-based system gives data collection, policy learning, and robot execution a common, verifiable action interface. Counterfactual comparisons turn failed executions into feedback on specific decisions, allowing the Reasoner to improve while its execution stack remains fixed. In the replay probe, 49.5% of eligible failures admit a full-return repair, showing that unsuccessful trajectories contain actionable supervision. In closed-loop tabletop grasping, counterfactual PPO achieves 62.38% success, compared with 55.45% for episode-return PPO and 56.44% for its SFT initialization. These findings support primitive-level credit assignment as a way to learn from physical interaction. Verifiable outcomes can do more than score a completed task: paired with counterfactual interventions, they provide focused feedback for learning better high-level decisions.

Limitations

The formulation assumes quasi-static primitives, and the quantitative evaluation centers on simulated tabletop grasping. The rescue probe measures the availability of verified corrections.

Attribution identifies the nearest sampled single-call repair under the recorded symbolic suffix, rather than a unique causal root. Finite sibling sets can miss repairs, and a single replacement cannot insert an omitted primitive or coordinate changes to multiple calls. Holding the suffix fixed can misvalue an intervention when the reasoner would choose a different continuation. Only eligible policy-sampled groups containing a sampled rescue enter PPO, so their advantages are selection-conditioned supervision, not an unbiased full-policy gradient.

Fixed simulation-based checks can be exploited. The reward measures physical outcomes, while grasp semantics require separate evaluation and do not enter optimization. The frozen VLP limits the reasoner's attainable performance, and random-start grasp success also depends on target generation, motion planning, and contact execution. The expanded robot-and-general comparison changes corpus scale and training budget together, so it evaluates the complete recipe.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

We used generative AI tools for literature search and for drafting and revising manuscript text, and to assist with code. Quantitative claims attributed to prior work were checked against the primary sources. We have reviewed all AI-assisted work and take responsibility for the final content of this paper.

For reasoner initialization, a geometric teacher supplies the action trajectories, and language models provide grasp descriptions and action rationales (Appendix C.1). The VLP training descriptions of Appendix A.3 are generated by GPT-5.6 Luna. These annotation models use fixed prompts and are not fine-tuned.

ETHICS STATEMENT

This work is conducted in simulation and on a robot in a laboratory setting; it involves no human subjects and no personally identifying data. Simulation experiments use the tabletop object assets described in Appendix C.1. No human subjects, personal data or human-robot contact is involved.

REPRODUCIBILITY STATEMENT

Appendices A–C and F.3–F.4 specify the VLP data construction, VLP teacher and student, Reasoner training, and robot interface and execution. The primitive grasp test and random-start closed-loop benchmark have separate populations and execution boundaries (Appendices D and E.1). Paired comparisons hold scene seeds and execution rules fixed. Counterfactual collection and policy optimization record their rollout, branch, and update budgets separately. The hardware deployment interface is described in Appendix F.

Results records are stored per episode with the configuration identifier that produced them, so a reported number can be traced to its weights, prompt, seeds and episode IDs.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.662. URL https://aclanthology.org/2024.acl-long.662/.

Rajarshi Biswas, Xiaoyi Lu, and Dhabaleswar K. Panda. Designing a micro-benchmark suite to evaluate gRPC for TensorFlow: Early experiences. In *Proceedings of the 9th Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, BPOE '18, 2018. URL https://arxiv.org/abs/1804.01138.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, et al. $\pi_{0.5}$: a Vision-Language-Action model with Open-World generalization. In Joseph Lim, Shuran Song, and Hae-Won Park (eds.), *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025. URL https://proceedings.mlr.press/v305/black25a.html.

Jiayi Chen, Yubin Ke, Lin Peng, and He Wang. Dexonomy: Synthesizing All Dexterous Grasp Types in a Grasp Taxonomy. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025. doi: 10.15607/RSS.2025.XXI.105.

Yanjun Chen, Yirong Sun, Hanlin Wang, Jinghan Wang, Xinming Zhang, Xiaoyu Shen, Wenjie Li, and Wei Zhang. Exact is easier: Credit assignment for cooperative LLM agents. *arXiv preprint arXiv:2603.06859*, 2026. URL https://arxiv.org/abs/2603.06859.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):265–293, 2021.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Bowei He, Yankai Chen, Xiaokun Zhang, and Xue Liu. Branching policy optimization: Sandbox-native language agent reinforcement learning. *arXiv preprint arXiv:2607.14171*, 2026. URL https://arxiv.org/abs/2607.14171.

Jiawei He, Danshi Li, Xinqiang Yu, Zekun Qi, Wenyao Zhang, Jiayi Chen, Zhaoxiang Zhang, Zhizheng Zhang, Li Yi, and He Wang. Dexvlg: Dexterous vision-language-grasp model at scale. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14248–14258. IEEE, 2025.

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. *arXiv preprint arXiv:2106.09685*, 2021. URL https://arxiv.org/abs/2106.09685.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D value maps for robotic manipulation with language models. In *Conference on Robot Learning (CoRL)*, pp. 540–562, 2023.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. ReKep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. In *Conference on Robot Learning (CoRL)*, pp. 4573–4602, 2024. arXiv:2409.01652.

Yuansen Huang, Jiayi Chen, Haoran Liu, Yubin Ke, Bing Han, Jiangran Lyu, Mi Yan, Li Yi, and He Wang. Kpgrasp: Scalable keypoint flow matching for dexterous grasp generation. *arXiv preprint arXiv:2606.09314*, 2026.

Jiabao Ji, Yongchao Chen, Yang Zhang, Ramana Rao Kompella, Chuchu Fan, Gaowen Liu, and Shiyu Chang. Collision- and reachability-aware multi-robot control with grounded LLM planners. *arXiv preprint arXiv:2505.20573*, 2025. URL https://arxiv.org/abs/2505.20573.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025. URL https://arxiv.org/abs/2503.09516.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tülu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Kailin Li, Jingbo Wang, Lixin Yang, Cewu Lu, and Bo Dai. Semgrasp: Semantic grasp generation via language aligned discretization. In *European Conference on Computer Vision*, pp. 109–127. Springer, 2024.

Yutai Li, Shaohui Peng, Jiaming Guo, Di Huang, Zihao Zhang, Yuxuan Guo, Yunkai Gao, Siming Lan, Ling Li, Xing Hu, and Yunji Chen. PrimitiveVLA: Learning reusable motion primitives for efficient and generalizable robotic manipulation. *arXiv preprint arXiv:2605.28634*, 2026.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.neurips.cc/paper_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0-Abstract-Conference.html.

Tengyu Liu, Zeyu Liu, Ziyuan Jiao, Yixin Zhu, and Song-Chun Zhu. Synthesizing diverse and physically stable grasps with arbitrary hand structures using differentiable force closure estimator. *IEEE Robotics and Automation Letters*, 7(1):470–477, 2021.

David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1511.03643.

Chenwei Lou, Zewei Sun, Xinnian Liang, Meng Qu, Wei Shen, Wenqi Wang, Yuntao Li, Qingping Yang, and Shuangzhi Wu. AdaCoT: Pareto-optimal adaptive chain-of-thought triggering via reinforcement learning. *arXiv preprint arXiv:2505.11896*, 2025. URL https://arxiv.org/abs/2505.11896.

MANUS. Metagloves Pro. Product documentation, 2026. URL https://www.manus-meta.com/products/metagloves-pro. Accessed September 2026.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2263–2279. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

Dantong Niu, Zhuoyang Liu, Zekai Wang, Boning Shao, Zhao-Heng Yin, Anirudh Pai, Yuvan Sharma, Stefano Saravalle, Ruijie Zheng, Jing Wang, Ryan Punamiya, Mengda Xu, Yuqi Xie, Yunfan Jiang, Letian Fu, Konstantinos Kallidromitis, Matteo Gioia, Junyi Zhang, Jiaxin Ge, Haiwen Feng, Fabio Galasso, Wei Zhan, David M. Chan, Yutong Bai, Roei Herzig, Jiahui Lei, Fei-Fei Li, Ken Goldberg, Jitendra Malik, Pieter Abbeel, Yuke Zhu, Danfei Xu, Linxi Fan, and Trevor Darrell. T-Rex: Tactile-reactive dexterous manipulation. *arXiv preprint arXiv:2606.17055*, 2026. URL https://arxiv.org/abs/2606.17055.

NVIDIA. Cosmos3-Nano. Hugging Face model release, 2026a. URL https://huggingface.co/nvidia/Cosmos3-Nano.

NVIDIA. NVIDIA Isaac GR00T N1.7: Open reasoning VLA model for humanoid robots. Blog post and model release, https://huggingface.co/blog/nvidia/gr00t-n1-7, 2026b.

Zekun Qi, Wenyao Zhang, Yufei Ding, Runpei Dong, Xinqiang Yu, Jingwen Li, et al. SoFar: Language-grounded orientation bridges spatial reasoning and object manipulation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. Spotlight. arXiv:2502.13143.

Qwen Team. Qwen3.8-27B. Hugging Face model release, August 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B.

Zachary Ravichandran, Ignacio Hounie, Fernando Cladera, Alejandro Ribeiro, George J. Pappas, and Vijay Kumar. Distilling on-device language models for robot planning with minimal human intervention. In *Conference on Robot Learning (CoRL)*, pp. 4859–4884, 2025. arXiv:2506.17486.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. CLIPort: What and where pathways for robotic manipulation. In *Conference on Robot Learning (CoRL)*, pp. 894–906, 2021.

Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, et al. CuRobo: Parallelized collision-free robot motion generation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8112–8119, 2023.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.

Huajie Tan, Enshen Zhou, Zhiyu Li, Yijie Xu, Yuheng Ji, Xiansheng Chen, Cheng Chi, Pengwei Wang, Huizhu Jia, Yulong Ao, et al. RoboBrain 2.5: Depth in sight, time in mind. *arXiv preprint arXiv:2601.14352*, 2026. URL https://arxiv.org/abs/2601.14352.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033, 2012.

Dylan Turpin, Liquan Wang, Eric Heiden, Yun-Chun Chen, Miles Macklin, Stavros Tsogkas, Sven Dickinson, and Animesh Garg. Grasp'd: Differentiable contact-rich grasp synthesis for multi-fingered hands. In *European Conference on Computer Vision*, pp. 201–221. Springer, 2022.

Ruicheng Wang, Jialiang Zhang, Jiayi Chen, Yinzhen Xu, Puhao Li, Tengyu Liu, and He Wang. Dexgraspnet: A large-scale robotic dexterous grasp dataset for general objects based on simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11359–11366, 2023. doi: 10.1109/ICRA48891.2023.10160982.

Zehang Weng, Haofei Lu, Danica Kragic, and Jens Lundell. Dexdiffuser: Generating dexterous grasps with diffusion models. *IEEE Robotics and Automation Letters*, 9(12):11834–11840, 2024.

Wei Wu, Fangjing Wang, Fan Lu, He Sun, Shi Liu, Yunnan Wang, Yibin Yan, Yong Wang, Shuailei Ma, Xinyang Wang, Yibin Liu, Shuai Yang, Tianxiang Zhou, Kejia Zhang, Lei Zhou, Cheng Su, Nan Xue, Bin Tan, Han Zhang, Youchao Zhang, Fei Liao, Xing Zhu, Yujun Shen, and Kecheng Zheng. From foundation to application: Improving VLA models in practice. *arXiv preprint arXiv:2607.06403*, 2026. URL https://arxiv.org/abs/2607.06403.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026.

Yinzhen Xu, Weikang Wan, Jialiang Zhang, Haoran Liu, Zikang Shan, Hao Shen, Ruicheng Wang, Haoran Geng, Yijia Weng, Jiayi Chen, et al. Unidexgrasp: Universal robotic dexterous grasping via learning diverse proposal generation and goal-conditioned policy. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4737–4746. IEEE, 2023.

Zhiyang Xu, Chao Feng, Rulin Shao, Trevor Ashby, Ying Shen, Di Jin, Yu Cheng, Qifan Wang, and Lifu Huang. Vision-Flan: Scaling human-labeled tasks in visual instruction tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 15271–15342. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.905. URL https://aclanthology.org/2024.findings-acl.905/.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 23826–23854. Curran Associates, Inc., 2023. doi: 10.52202/075280-1035. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/4b175d846fb008d540d233c188379ff9-Abstract-Datasets_and_Benchmarks.html.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757. Curran Associates, Inc., 2022. doi: 10.52202/068431-1508. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

Jianglong Ye, Keyi Wang, Chengjing Yuan, Ruihan Yang, Yiquan Li, Jiyue Zhu, Yuzhe Qin, Xueyan Zou, and Xiaolong Wang. Dex1B: Learning with 1B Demonstrations for Dexterous Manipulation. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025. doi: 10.15607/RSS.2025.XXI.106.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi Fan, and Joel Jang. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026. URL https://arxiv.org/abs/2602.15922.

Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning (CoRL)*, pp. 726–747, 2020.

Lei Zhang, Diwen Zheng, Kaixin Bai, Zhenshan Bing, Zoltan-Csaba Marton, Zhaopeng Chen, Alois Christian Knoll, and Jianwei Zhang. OmniDexVLG: Learning dexterous grasp generation from vision language model-guided grasp semantics, taxonomy and functional affordance. *arXiv preprint arXiv:2512.03874*, 2025.

Ruijie Zheng, Dantong Niu, Yuqi Xie, Jing Wang, Mengda Xu, Yunfan Jiang, Fernando Castañeda, Fengyuan Hu, You Liang Tan, Letian Fu, Trevor Darrell, Furong Huang, Yuke Zhu, Danfei Xu, and Linxi Fan. EgoScale: Scaling dexterous manipulation with diverse egocentric human data. *arXiv preprint arXiv:2602.16710*, 2026. URL https://arxiv.org/abs/2602.16710.

Yiming Zhong, Qi Jiang, Jingyi Yu, and Yuexin Ma. Dexgrasp anything: Towards universal robotic dexterous grasping with physics awareness. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22584–22594. IEEE, 2025.

Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv preprint arXiv:1801.09847*, 2018.

A VLP TRAINING DATA

This appendix describes the data used to train the arm–hand VLPs introduced in Section 3.1. Each training sample corresponds to a physically verified single-hand operation and comprises visual observations, a target-object point cloud, a description, an active-hand identifier, and two action keyframes. We first define the training samples and supervision, and then describe demonstration augmentation and language annotation.

A.1 TRAINING SAMPLE SPECIFICATION

Figure 3 illustrates the semantics of the VLP-based primitives, and Figure 4 shows the complete collection pool before filtering, grouped by grasp type and non-prehensile primitive. The grasp pool contains 1,818,811 samples; Appendix B.3 reports the filtered dataset used for training and evaluation.

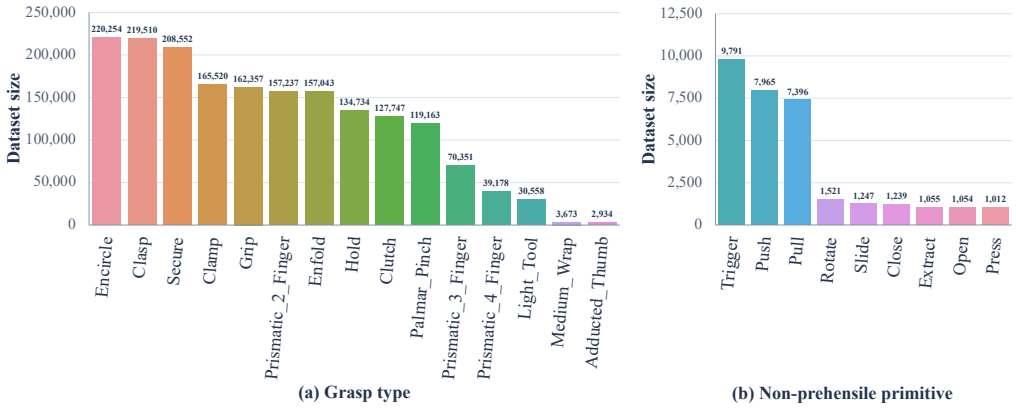


Figure 4: VLP sample counts in the complete collection pool before filtering. (a) Grasp samples grouped by grasp type. (b) Samples grouped by non-prehensile primitive.

Visual observations, point cloud, and language condition. We place the robot in the HOME configuration specified in Appendix F.1 and the target object at the initial pose of the sample, and render four synchronized RGB images in MuJoCo, $I_k = (I_k^{\text{left eye}}, I_k^{\text{right eye}}, I_k^{\text{left wrist}}, I_k^{\text{right wrist}})$. The target-object point cloud $P_k \in \mathbb{R}^{1024\times 3}$ is constructed as described in Appendix B.1. The language condition ℓ_k is the operation description, and $h_k \in \{\text{left}, \text{right}\}$ identifies the active hand. Table 3 summarizes the principal fields of a training sample.

Targets and supervision. Each training sample contains two action keyframes. Each keyframe consists of a 7-dimensional arm-joint target and a 22-dimensional finger-joint target:

$$\mathbf{y}_k = \begin{bmatrix} \mathbf{y}_k^{\text{pre-contact}} \\ \mathbf{y}_k^{\text{goal}} \end{bmatrix} \in \mathbb{R}^{58}, \quad \mathbf{y}_k^s = [\mathbf{q}_{k,\text{arm}}^s; \mathbf{q}_{k,\text{hand}}^s] \in \mathbb{R}^{29}, \quad s \in \{\text{pre-contact}, \text{goal}\}. \tag{11}$$

The VLP with point-cloud input learns the mapping

$$(I_k, P_k, \ell_k, h_k) \longmapsto \mathbf{y}_k. \tag{12}$$

The distilled model without point-cloud input is conditioned on (I_k, ℓ_k, h_k) and uses the same action supervision. Appendices B.1 and B.2 describe point-cloud construction and the network architecture, respectively.

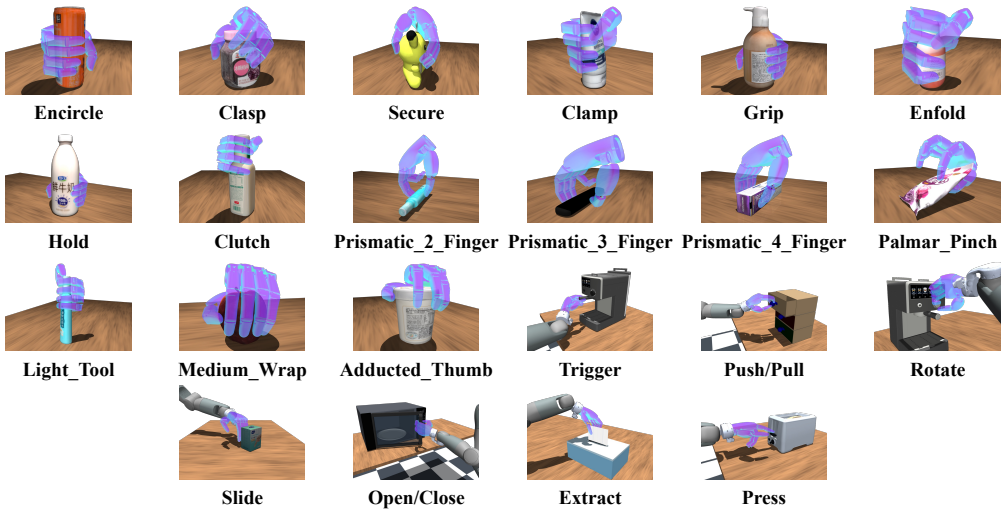


Figure 3: Semantic illustrations of the primitives.

Table 3: Principal fields of a VLP training sample.

Field	Content
`sample_id`	Sample identifier.
`observation`	Four synchronized RGB images in the HOME configuration.
`pointcloud`	The 1,024-point target-object cloud, expressed in the robot-base frame in meters.
`description`, `side`	Language description of the operation and the active side.
`target_action29x2`	2×29 absolute joint-space action targets.

A.2 Verified demonstration augmentation

We construct the dataset from teleoperated demonstrations and their augmented variants. During teleoperation, an operator performs hand motions while sensors capture the hand and wrist states; these measurements drive the simulated dexterous hand in real time. The resulting episodes provide human manipulation demonstrations for VLP training. The capture hardware, calibration, and retargeting implementation are specified in Appendix F.2.

Reference demonstrations and keyframes. Each teleoperation episode stores the target palm pose and Sharpa finger commands, the global physical state, contact records, and the scene assets. We extract pre-contact and goal action keyframes from every qualified demonstration; these keyframes are annotated during capture.

Augmentation. We augment each demonstration by randomly sampling robot-base and object poses and solving inverse kinematics. For every episode, an augmentation samples one robot-base pose and one object pose within predefined ranges. For tabletop operations, we additionally randomize the table height continuously from 74 to 82 cm, around the default height of 78 cm. We then rigidly transform the pre-contact and goal keyframes to preserve the hand pose relative to the object. For each transformed keyframe, arm inverse kinematics converts the 6-DoF palm pose into seven arm-joint targets, while the 22 finger commands are retained as finger-joint targets. Together they form the two 29-dimensional keyframes used by the VLP.

Physical verification. Every augmented example undergoes physical verification, and only successful examples enter the VLP training set. The execution procedure and success criterion are

identical to those used when the corresponding VLP-based primitive is invoked. Figure 5 summarizes the augmentation pipeline.

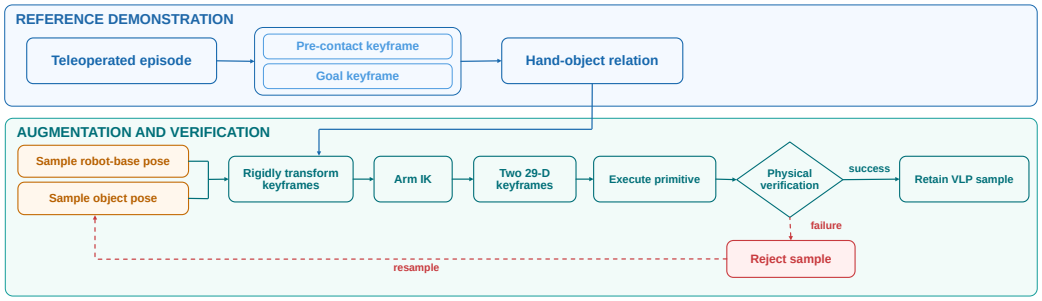


Figure 5: Keyframe augmentation from teleoperated demonstrations. Each augmentation samples robot and object poses, applies rigid transforms that preserve the hand–object relation, resolves arm inverse kinematics for both keyframes, and retains only examples that pass the execution and physical-verification procedure of the corresponding primitive.

A.3 Grasp description annotation

Grasp descriptions provide the VLP with a language condition for the hand–object configuration. Each object-centric description specifies the grasped region, opposition between the thumb and the other fingers, finger flexion and enclosure, and the palm pose relative to the object. We generate annotations with GPT-5.6 Luna at medium reasoning effort. Each request contains the following information:

1. **Four target-hand renderings.** We clamp the squeezed finger action target to the joint limits and place the hand mesh by forward kinematics. The hand–object relative pose is preserved, and both visual meshes are rendered. The hand is opaque and the object has opacity $\alpha = 0.30$. Four opposing oblique inspection views are rendered for language annotation.
2. **Finger joint angles.** We provide the 22 joint names of the active hand, the clamped squeezed target angles, and their respective limits, all converted to degrees and rounded to integers.
3. **Fixed system prompt.** The prompt is reproduced verbatim below:

> You describe dexterous grasp ACTION TARGETS for a robot policy. The four images show the SAME squeezed hand target and the SAME object from four free inspection viewpoints. They are NOT four on-robot cameras and are NOT successive time steps. The hand is opaque; the object is intentionally semi-transparent so finger placement inside its outline remains visible.
>
> The hand mesh was placed by forward kinematics at its commanded joint angles, after actuator/joint saturation, WITHOUT a physics step. Target penetration is intentional. Do not interpret overlap as measured contact, deformability, force, pressure, grip strength or physical success. Joint values and limits are rounded degrees, not precision measurements. Fractions of joint range do not by themselves mean a finger is more closed: FE, AA and CMC axes differ.
>
> Describe the hand's configuration relative to the object, not to a camera or image border. Use anatomy: thumb, index, middle, ring, little finger, palm. State the grasped region, opposition, enclosure, and differences among digits only when supported by the geometry and joint values. Avoid unsupported grasp-type labels or force claims. Put ambiguity in uncertainties, not in confident description text. No robot-base position, arm configuration, table, transport, lifting or release instructions belong in these descriptions.
>
> First extract one shared set of semantic facts. Then produce the requested number of DIFFERENT WORDINGS of exactly those same facts, each in Chinese and English. Do not add or remove a distinguishing finger/region/orientation fact between variants. Wording, order and syntax may vary; the requested grasp must not. Descriptions should guide a

> policy rather than narrate images. Do not mention transparency, rendering, images, coordinate numbers, limits, or evidential caveats in description_variants; keep caveats in uncertainties. Descriptions must be detailed enough to distinguish this grasp without padding. Never claim equivalence has been independently verified.

Diverse phrasings. Each request first generates one shared set of semantic facts and then returns four descriptions. The four descriptions may vary in wording, order, and syntax, but the prompt requires them to preserve the same grasped region, digit-specific distinctions, and orientation relations. The output follows a fixed JSON schema with the fields `semantic_facts`, `description_variants`, `uncertainties`, and `needs_review`, summarized in Table 4. During training, we randomly select one phrasing from the description set, associating the same action target with multiple verbal formulations.

Table 4: Output fields of grasp-description annotation.

Field	Meaning
`semantic_facts`	The semantic facts shared by all four descriptions, including object shape, grasped region, the palm–object relation, the configuration of each of the five digits, and opposition and enclosure relations.
`description_variants`	Four semantically equivalent descriptions with different wording. One description is sampled at random during training.
`uncertainties`	Relations or ambiguities that the model cannot determine from the images and joint information; used only to support review.
`needs_review`	The model's review flag. When `true`, the result is marked for manual review; unreviewed annotations are excluded from training.

Example grasp description. Figure 6 shows the grasp configuration corresponding to the following example language condition provided to the grasp VLP:

> Relative to the object, the palm lies over and behind its broad upper mass, forming an arch with the right hand. The thumb travels downward along one side toward the upper center, while the index, middle, ring, and little fingers curve downward from the opposing side. The index has the least flexion, the middle and ring have similar moderate flexion, and the little is the most folded; the target is the upper central body, not a thin lateral protrusion or the lower extension.

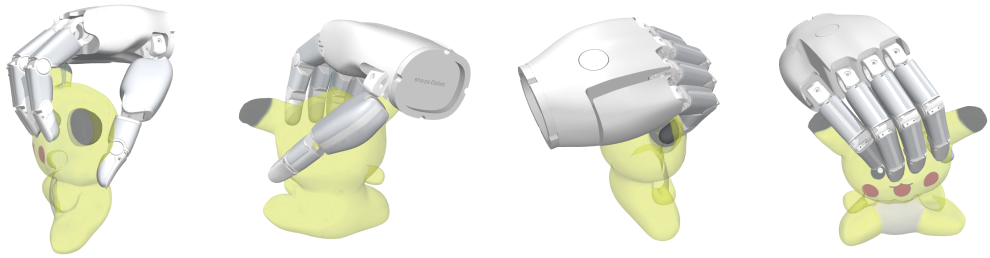


Figure 6: Four views of the grasp configuration described in the example language condition.

B VLP Teacher–Student Training

This appendix specifies the grasp VLP teacher and deployed RGB student introduced in Section 3.1. It describes the teacher's point-cloud input and the two network architectures, followed by teacher training and the distillation procedure used to train the RGB student.

B.1 INPUT CONSTRUCTION

Each RGB image is resized to 320×256 using bicubic interpolation. The images and text are jointly provided to Qwen3-VL-4B-Instruct; the text contains the action description, active-hand identifier, and the names of the four views.

Let O denote the object frame. We cast rays inward from virtual viewpoints distributed approximately uniformly on a sphere enclosing the object (Zhou et al., 2018) and retain only the first intersection: ray k, with origin $\mathbf{o}_k$, unit direction $\mathbf{d}_k$, and first valid hit distance t_k, gives the surface point $\mathbf{p}_k = \mathbf{o}_k + t_k \mathbf{d}_k$ in O. We then apply adaptive voxel downsampling. The initial voxel size is $1/256$ of the object bounding-box diagonal and is adjusted so that the number of candidate points lies between 8192 and 65536; each occupied voxel retains the point nearest its center. Farthest point sampling (FPS) subsequently reduces the cloud to a fixed size $N = 1024$. The seed used to select the first FPS candidate is derived deterministically from the asset identifier and a hash of the source mesh. Figure 7 shows representative objects and the corresponding point clouds produced by this procedure. Writing W for the simulation world frame, B for the robot base frame, and ${}^{X}\mathbf{T}_{Y}$ for the homogeneous transform carrying points from frame Y into frame X, a trajectory supplies ${}^{W}\mathbf{T}_{O}$ and ${}^{W}\mathbf{T}_{B}$ as positions and quaternions. A point enters the base frame as

$$ {}^{B}\tilde{\mathbf{p}}_i = {}^{B}\mathbf{T}_{W}\, {}^{W}\mathbf{T}_{O}\, {}^{O}\tilde{\mathbf{p}}_i. \tag{13} $$

Here, $\tilde{\mathbf{p}}$ denotes homogeneous coordinates and ${}^{B}\mathbf{T}_{W} = ({}^{W}\mathbf{T}_{B})^{-1}$; dehomogenization gives ${}^{B}\mathbf{p}_i$. All point-cloud coordinates are expressed in meters.



Figure 7: Representative objects and their corresponding point clouds after ray casting, adaptive voxel downsampling, and farthest point sampling.

B.2 VLP ARCHITECTURE

This section presents the network architecture that implements the mappings defined in Appendix A.

Corresponding to the two mappings defined in Appendix A, we construct a teacher $f_T(I_k, P_k, \ell_k, h_k)$ that additionally receives the target-object point cloud and a student $f_S(I_k, \ell_k, h_k)$ conditioned only on vision and language; both predict the same two-keyframe action $\mathbf{y}_k$. The two models share the same frozen Qwen vision–language encoder and use identical structures for the Qwen adapter, active-hand embedding, action input embedding, keyframe-role embedding, two-keyframe ActionDiT, and action output head. The teacher uses a point-cloud encoder that converts the target-object point cloud into 128 geometry condition tokens. The student instead uses a visual spatial encoding branch that generates 128 spatial condition tokens from high-resolution features of the four RGB views.

Vision–language encoder. Qwen's visual tower, projector, and language backbone are frozen.

At this input resolution, each image forms a 16×20 patch grid, which Qwen's 2×2 spatial merger reduces to 80 visual tokens. After joint encoding, we retain 320 visual tokens and at most 704 nonvisual tokens; masked padding fills unused nonvisual slots, producing a 1024×2560 vision–language representation. To match the 4096-dimensional condition interface of the Wan2.2-initialized ActionDiT, a learned linear adapter maps each token from width 2560 to 4096.

Point-cloud encoder. For the point cloud constructed in Appendix B.1, deterministic FPS starts with the point farthest from the centroid and selects 128 points for encoding.

Each point is represented by its three-dimensional coordinates and a deterministic 32-dimensional descriptor. The descriptor comprises the 3-D centroid-relative position, 3-D scale-normalized coordinates, six polynomial terms, one radial feature, 18 Fourier features, and one constant term. Two independent learned projectors map the LayerNorm-normalized descriptor and the three-dimensional coordinates to width 512, and their outputs are added elementwise. A geometry adapter comprising SiLU, a linear projection, and LayerNorm then maps the fused features to width 4096. A learned geometry-modality embedding is added elementwise, yielding a 128×4096 sequence of geometry tokens.

Visual spatial encoding branch. The student constructs visual memory from the high-resolution features output by the frozen Qwen visual tower before the spatial merger. Each view provides a 16×20 grid of 1024-dimensional tokens, giving a 1280×1024 visual sequence across four views. A visual adapter maps the visual features to width 512; a separate MLP maps two-dimensional patch coordinates normalized to $[-1, 1]^2$ to width 512, and learned embeddings represent the identities of the four cameras. The three terms are added elementwise to form the visual memory of the spatial encoder.

The spatial encoder contains 128 learned queries and four attention blocks, with hidden width 512, feed-forward width 2048, and eight attention heads of dimension 64. Within each block, the queries sequentially undergo query-to-image cross-attention, query self-attention, and a feed-forward network. After LayerNorm, the final 128×512 representation is mapped to width 4096 by a spatial adapter comprising SiLU, a linear projection, and LayerNorm. A learned spatial-modality embedding is added elementwise, yielding a 128×4096 sequence of spatial condition tokens.

Two-keyframe ActionDiT. The flow state is represented by two 29-dimensional action tokens. A learned dense action input embedding projects each token into the 1024-dimensional hidden space of ActionDiT, after which its learned keyframe-role embedding is added elementwise. Self-attention uses rotary position embeddings (RoPE). The flow timestep t is encoded with a sinusoidal embedding and modulates each DiT block through adaptive Layer Normalization (AdaLN).

The teacher condition sequence concatenates one active-hand token, 1024 adapted Qwen vision–language tokens, and 128 geometry tokens. The student replaces the geometry tokens with 128 spatial tokens. Both condition sequences therefore have shape 1153×4096. The two models use identically structured condition MLPs that map this sequence as $4096 \to 1024 \to 1024$, with GELU between the two projections, to produce the condition representation used by every block.

ActionDiT contains 30 blocks, an action hidden width of 1024, a feed-forward width of 4096, and 24 attention heads of dimension 128. Each block applies self-attention between the two action tokens, cross-attention to the condition sequence, and a feed-forward network. A linear output head predicts a 29-dimensional flow velocity for each token. Figure 8 shows the conditioning and flow-prediction pipeline, and Table 5 reports the module dimensions and trainable parameter counts.

Table 5: GraspVLP module dimensions and trainable parameter counts. Shapes exclude the batch dimension. Qwen is fully frozen, and all remaining modules are trainable. ActionDiT parameters include the condition MLP, time embedding, action input and output layers, and Transformer blocks. Parameter counts are rounded and expressed in millions.

Component	Output or configuration	Trainable params (M)
Qwen3-VL-4B-Instruct	1024×2560	0
Shared Qwen adapter	1024×4096	10.49
Side embedding	1×4096	0.008
Geometry projector	128×4096	2.13
Keyframe-role embeddings	2×1024	0.002
ActionDiT	30 blocks; 2×29 output	1020.93
Total		**1033.56**

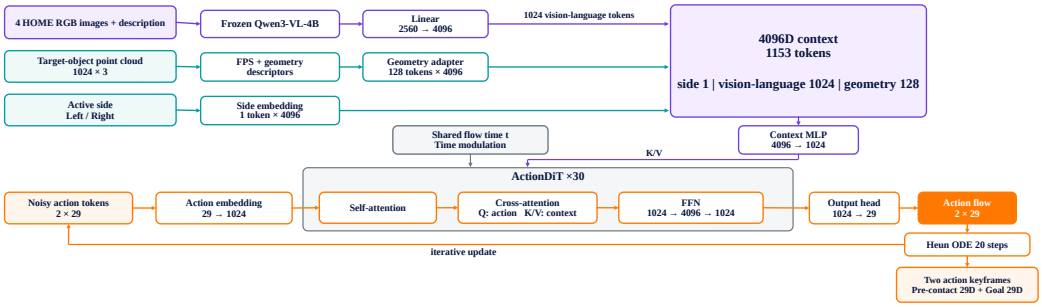


Figure 8: Teacher conditioning and flow prediction. Frozen Qwen3-VL-4B-Instruct extracts vision–language features, combined with geometry and active-side embeddings to condition the 30-block ActionDiT. The deployed student substitutes RGB spatial tokens for the geometry tokens. Both models predict flow velocities for two 29-dimensional action tokens.

B.3 Geometry-conditioned teacher training

Data split. The filtered grasp dataset contains 1,450,381 samples: 1,308,778 for training, 68,695 for validation, and 72,908 for testing. Entire sample groups are assigned to the training, validation, and test sets in an approximately $90\%/5\%/5\%$ ratio using a deterministic hash. Augmented samples derived from the same demonstration remain in the same split, preventing source-related leakage from the test set into the training and validation sets.

Flow-matching objective. Targets are standardized using per-dimension means and standard deviations computed on the training set. Given $\mathbf{x}_1 \in \mathbb{R}^{58}$, we sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I)$ and $t \sim \mathcal{U}(0,1)$ and construct

$$\mathbf{x}_t = (1-t)\boldsymbol{\epsilon} + t\mathbf{x}_1. \tag{14}$$

Training uses only the following flow-matching objective:

$$\mathcal{L}_{\text{flow}} = \mathbb{E}_{\mathbf{x}_1,\boldsymbol{\epsilon},t}\left[\left\|\mathbf{v}_\theta(\mathbf{x}_t, t, \mathbf{c}) - (\mathbf{x}_1 - \boldsymbol{\epsilon})\right\|_2^2\right]. \tag{15}$$

Optimization. The ActionDiT backbone is initialized from the pretrained Wan2.2-TI2V-5B video diffusion model, using linear interpolation of weights and scale correction to match the dimensions of the action network. The action input and output layers, Qwen adapter, side embedding, keyframe-role embeddings, and geometry projector are initialized randomly. Training uses PyTorch DDP. Table 6 reports the full configuration. Training uses a constant learning rate throughout. Frozen Qwen features are cached using group-wise INT8 quantization with group size 128 and BF16 scale factors; they are restored to BF16 when loaded.

We monitor flow loss on a fixed set of 4,096 validation samples with shared random-noise seeds and evaluate the teacher after the training budget in Table 6.

B.4 RGB student distillation

Flow-matching objective. Distillation uses end-to-end training. The teacher and student receive the same action description, active-hand identifier, four images, noisy action, and flow time. The teacher additionally receives the target-object point cloud. The teacher-flow objective transfers information available to the teacher during training into the deployed student's action predictions, following the privileged-information view of distillation (Lopez-Paz et al., 2016).

Let $\mathbf{v}^* = \mathbf{x}_1 - \boldsymbol{\epsilon}$ denote the ground-truth flow velocity, and let $\hat{\mathbf{v}}_S$ and $\hat{\mathbf{v}}_T$ denote the velocities predicted by the student and frozen teacher, respectively, at the same $\mathbf{x}_t$ and t. The training objective is

$$\mathcal{L}_{\text{distill}} = \underbrace{\left\|\hat{\mathbf{v}}_S - \mathbf{v}^*\right\|_2^2}_{\mathcal{L}_{\text{GT-flow}}} + \lambda_{\text{KD}}(s)\underbrace{\left\|\hat{\mathbf{v}}_S - \hat{\mathbf{v}}_T\right\|_2^2}_{\mathcal{L}_{\text{teacher-flow}}}, \tag{16}$$

Table 6: GraspVLP training configuration. BF16 is used for mixed-precision computation, while parameters and optimizer states use FP32.

Setting	GraspVLP
Training steps	25,993
Training epochs	Approximately 40.67
GPU configuration	8 nodes, 64 RTX 5090 GPUs
Global batch size	2,048
Per-GPU microbatch	32
Precision	FP32 parameters and Adam states; BF16 autocast
Optimizer	Fused AdamW
Learning rate	6.90×10^{-7}, constant, no warmup
Adam betas	$(0.9, 0.999)$
Weight decay	0.01
Gradient norm limit	1.0
Validation frequency	Every 500 steps

where s denotes the current training step. Let $S = 20000$ be the total number of training steps and $r = (s-1)/(S-1)$. Then

$$\lambda_{\mathrm{KD}}(s) = \begin{cases} 1, & r \le 0.6, \\ 1 - 0.75\dfrac{r - 0.6}{0.4}, & r > 0.6. \end{cases} \tag{17}$$

Thus, the teacher-flow weight remains 1 for the first 60% of training and then decays linearly to 0.25, while the ground-truth flow weight remains 1 throughout.

Optimization. The student's visual spatial encoder, spatial adapter, and spatial-modality embedding are initialized randomly; all other parameters are initialized from the teacher. During distillation, all student parameters except the Qwen vision–language encoder are optimized jointly. The student is trained for 20,000 optimization steps on 64 RTX 5090 GPUs across eight nodes, with a global batch size of 2,048. The optimizer, learning rate, weight decay, gradient clipping, and mixed-precision configuration follow Table 6. Every 500 steps, we compute the unweighted ground-truth flow loss on a fixed set of 4,096 validation samples and save a checkpoint.

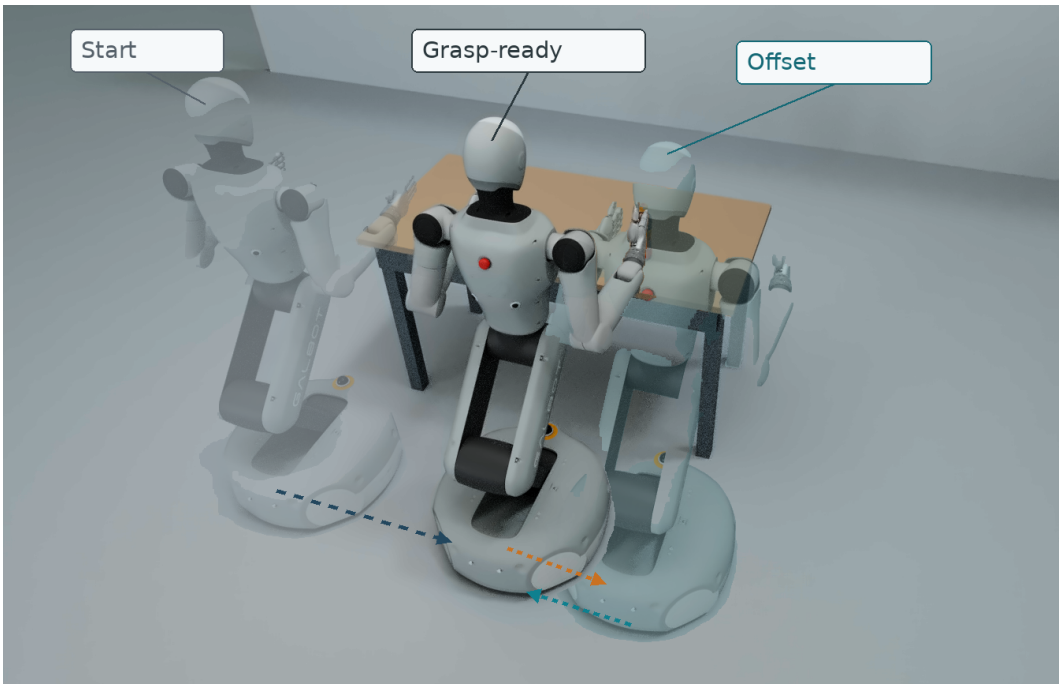
GT-only control. The GT-only and distilled students share the network architecture, teacher initialization, training examples, and optimization configuration. The GT-only objective uses the ground-truth flow term in equation 16; the distilled objective adds the teacher-flow term.

C Reasoner Training

C.1 Supervised data collection

We collect right-hand tabletop grasping episodes with a geometric teacher and a frozen grasp VLP (Figure 9). The teacher chooses the next base motion or grasp call. A language model writes the grasp description from the initial images and target, then explains each supplied teacher action at its decision state.

Scene and motion sampling. Each seed selects an object from the scene catalog. Objects are placed on a $0.78\,\mathrm{m}$ table with $x \sim \mathcal{U}[-0.137, 0.163]\,\mathrm{m}$, $y \sim \mathcal{U}[-0.20, 0.30]\,\mathrm{m}$, and yaw in $\{0, 45, \ldots, 315\}^\circ$. The base starts 0.30–$0.80\,\mathrm{m}$ behind home and -0.50–$0.50\,\mathrm{m}$ laterally, on a $0.10\,\mathrm{m}$ grid, with yaw in $\{-90, -60, \ldots, 90\}^\circ$ relative to home. Using simulator poses, the teacher brings the target to $(0.62, -0.25)\,\mathrm{m}$ in the base frame through yaw, local-x, then local-y motions, using 30° and $0.10\,\mathrm{m}$ increments and capping translations at $0.50\,\mathrm{m}$. Each seed selects one of six route forms (Figure 10): nominal, split translation, overtravel, undertravel, lateral offset, or yaw offset. Split translation divides the same base displacement into shorter consecutive `xy_move` calls, adding intermediate decision states without changing the endpoint or grasp target. The offset forms instead add a deviation and inverse correction before the same grasp call. Yaw detours occur at route entry, while translation detours occur at the grasp handoff.



Teacher: 1. base motion: `xy_move`$(x = 0, y = -0.4\,\mathrm{m}) \rightarrow$ 2. `grasp_vlp`.
Injected: 1. base motion: `xy_move`$(x = 0, y = -0.4\,\mathrm{m}) \rightarrow$ 2. base motion: `xy_move`$(x = 0, y = -0.2\,\mathrm{m})$ [offset] $\rightarrow$ 3. base motion: `xy_move`$(x = 0, y = +0.2\,\mathrm{m})$ [recovery] $\rightarrow$ 4. `grasp_vlp`.

Figure 9: A lateral-offset route. An Isaac Sim schematic using robot poses and the target object from one rollout. The start and offset poses are translucent, and their spacing is enlarged for clarity. The arrows show teacher approach (navy), injected offset (orange), and recovery (teal). The text below the rendering lists the corresponding primitive sequences; coordinates are base-frame displacements.

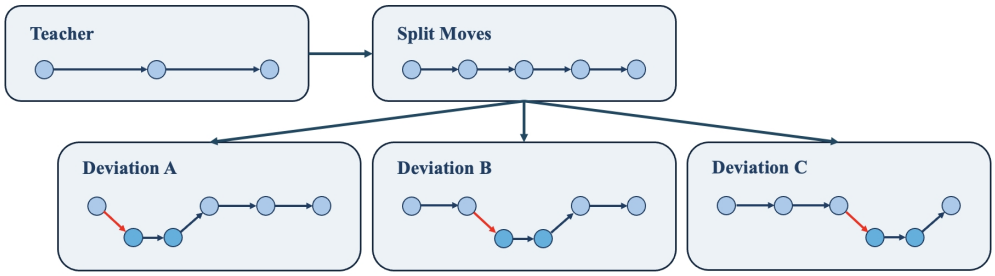


Figure 10: Route variants for Reasoner SFT (schematic). Split moves add intermediate decision states; deviation variants inject an offset (red arrows) followed by teacher-guided recovery. Recovery actions provide supervised targets for perturbed observations. Each rollout selects one variant.

Decision records. Each record contains four RGB views taken before the action, the teacher target $a_t^\star$, and the action actually executed e_t. When a deviation is injected, the supervised target remains the nominal teacher action; the following off-route observation is paired with the recovery action for Reasoner SFT. Histories store the executed actions and their roles. Decision states from both successful and failed episodes are retained. The language-model annotation pairs each target primitive with a one-sentence English rationale.

Data item	Train (%)	Validation (%)	Test (%)
Tabletop object assets	85.81	5.16	9.03
Fully annotated robot rollouts	85.81	4.96	9.22
Robot decision samples	85.82	4.96	9.22
Vision-Flan examples	79.66	10.18	10.16

Table 7: Data split proportions. Robot examples use fully annotated episodes, including both successful and failed grasps. Each row is normalized by its own total; rounding may affect the sum. One decision sample pairs a synchronized four-view observation with its rationale and teacher-action target.

Corpus inventory. The final inventory records 92,080 unique rollout IDs. Of these, 78,706 have a label for every state of a nonempty exported decision trace, yielding 443,869 decision samples across 155 object assets (Table 7). Rollouts are deduplicated by ID and decisions by rollout ID and step index. By executed-action role, the samples comprise 374,161 ordinary-teacher states, 34,854 injected-deviation states, and 34,854 recovery-teacher states. The train/validation/test splits contain 133/8/14 object assets, 67,539/3,907/7,260 annotated rollouts, and 380,939/22,022/40,908 decision samples, respectively. These are corpus-level statistics; the training sets for individual Reasoner conditions are specified in Appendix C.2.

We also prepare 19,967 general visual-instruction examples from the 191-task release of Vision-Flan (Xu et al., 2024), covering all 191 tasks. Robot splits group all poses and decisions of an object asset together; general examples sharing an image hash remain in the same split. Appendix C.2 describes the training recipes. The Vision-Flan split contains 15,906 training, 2,032 validation, and 2,029 test examples.

Additional general data. The supplementary sources contain 157,712 conversations from LLaVA-Instruct-150K (Liu et al., 2023) and 28,299 question–answer pairs from the ChartQA training split (Masry et al., 2022). We validate annotations and image decoding, remove exact duplicate annotations, and exclude exact image-byte hashes or normalized questions shared with MMBench, CV-Bench, or BLINK, as well as image hashes shared with Vision-Flan. Image-hash groups receive a deterministic split with target train/validation/test proportions of 80%/10%/10%. The retained corpus contains 151,326 LLaVA conversations (121,106/14,896/15,324 train/validation/test) and 28,127 ChartQA pairs (22,558/2,892/2,677). Together, the 179,453 records comprise 143,664 training, 17,788 validation, and 18,001 test examples, with disjoint image hashes across splits.

C.2 Reasoner initialization

Training samples. For task and target I, observation O_t, and executed-action entry E_t, we form two inputs with the same JSON target:

$$C_t^{\text{current}} = [I, O_t], \quad C_t^{\text{history}} = [I, O_0, E_0, \ldots, E_{t-1}, O_t], \quad Y_t = [r_t, a_t^\star]. \tag{18}$$

The target fields are `reason` and `next_primitive`.

Prediction target and context policies. At each decision state, the Reasoner outputs one JSON object with a concise English `reason` and a `next_primitive`: a base-frame `xy_move`, a `yaw_rotate`, or a right-hand `grasp_vlp` call containing a target ID and grasp description. All policies predict the same supervised target $Y_t = [r_t, a_t^\star]$ and receive the task instruction and target identity. The four synchronized RGB views in each observation are $I_t = (I_t^{\text{left eye}}, I_t^{\text{right eye}}, I_t^{\text{left wrist}}, I_t^{\text{right wrist}})$. Simulator object and robot poses are never supplied to the Reasoner. The main context comparison uses the following two policies:

- **Current-observation policy.** The prompt contains the task, target, and the latest four-view observation O_t. Its supervised input is C_t^{current} in this subsection.
- **Full visual-and-action history policy.** The prompt interleaves all four-view observations from the episode start through O_t with the intervening executed primitive entries:

Two context policies initialized from the pretrained checkpoint	
Distributed batch	Six GPUs; one decision per GPU; eight accumulation steps; global batch 48
LoRA	Rank 16; $\alpha = 32$; dropout 0.05; FP32 adapters on a BF16 base
Optimizer	AdamW; learning rate 5×10^{-5}; weight decay 0.01; norm clip 1
Schedule	3% linear warmup, then cosine decay; seed 42
Input budget	16,384 serialized tokens; 65,536–262,144 pixels per image

Table 8: Training settings shared by the current-observation and full-history policies initialized from the pretrained model. A sample is one decision state.

$[I, O_0, E_0, \ldots, E_{t-1}, O_t]$. An entry records the action actually executed and its role when relevant (including an injected deviation).

Shared supervised fine-tuning. The context comparison and robot-plus-general recipe use fixed snapshots of the collection described in Appendix C.1. The context comparison uses the initial collection, while the mixed-data recipe uses an expanded snapshot. Both retain teacher decisions from successful and failed episodes, require complete annotations for each nonempty decision trace, and use the same deterministic object-asset split. The final corpus inventory includes subsequent additions to these training snapshots.

The current-observation and full-history policies initialize independently from the same pretrained Qwen3.8-27B model (Qwen Team, 2026) and use the same 19,246 robot primitive-decision training targets collected with the procedure in Appendix C.1. Rank-16 LoRA (Hu et al., 2021) adapters cover all 496 language-model linear modules, with 116,727,808 trainable parameters. The vision encoder, visual merger, token embeddings, pretrained language weights, and output head remain frozen. Pretrained weights use BF16 and adapters FP32. The training mixture comprises robot primitive-decision examples only. Each decision state is a separate chat sequence. Only the final JSON response, including its end-of-sequence token, is supervised; system text, images, instructions, and earlier assistant messages serve as context. With $m_{ij} = 1$ only at those supervised completion tokens, the teacher-forced objective is

$$\mathcal{L}(\phi; \mathcal{B}) = -\frac{\sum_{i \in \mathcal{B}} \sum_j m_{ij} \log p_{\theta_0, \phi}(x_{ij} \mid \mathbf{x}_{i,<j})}{\sum_{i \in \mathcal{B}} \sum_j m_{ij}}, \tag{19}$$

where only adapter parameters ϕ are updated. The normalization covers supervised tokens over the full distributed update.

Training shuffles the decision states. The context comparison uses the same training data and matched optimizer-update counts.

General-data mixture. The robot-plus-general SFT initialization in Section 3.3 uses $\mathcal{D}_{\text{call}}$ with 223,603 robot primitive-decision examples and $\mathcal{D}_{\text{general}}$ with 15,906 Vision-Flan training examples. Robot targets contain a rationale and the next primitive call; general targets follow the visual instruction for each example. Robot and general data receive 80% and 20% of optimizer updates, respectively. We use the current-observation model for closed-loop and general-capability evaluation. Training uses eight GPUs, one example per GPU, and six accumulation steps, for a global batch of 48; the LoRA and optimizer settings follow Table 8. This SFT model is the shared initialization for rescue distillation and the two PPO conditions.

Training cost. Table 9 reports wall time from trainer launch through saving the model used for evaluation, including initialization, in-training validation, and checkpoint writing. Queueing, dataset construction, and independent evaluation are excluded. GPU-hours equal wall time multiplied by the allocated GPU count.

C.3 Verifiable physical feedback

Every executed call returns a record containing execution status, physical diagnostics, and an acceptance bit from the frozen success rule. Task returns aggregate these physical fields. Semantic

Context used for SFT	Training data	GPUs	Time (h)	GPU-hours
Current observation	Robot-only	6	20.15	120.91
Full visual-and-action history	Robot-only	6	65.22	391.32
Current observation	Robot + general	8	68.90	551.20

Table 9: Reasoner SFT conditions and training costs. The current-observation and full-history robot-only policies have matched update counts. The robot-and-general recipe changes data scale, mixture, and training budget together.

Table 10: Completion and failure criteria of the primitives.

Primitive	Completion and failure criteria
`xy_move`	The arguments pass the validation in Table 23 and all steps complete. This primitive uses no motion planner and imposes no additional terminal-pose tolerance.
`z_move`	Each component of torso translation error is at most $0.3\,$mm, rotation error is at most 1°, linear speed is at most $2\,$mm/s, and angular speed is at most 0.5°/s for five consecutive control ticks. Infeasible IK at any waypoint or failure to settle within $5\,$s is a failure.
`yaw_rotate`	The arguments pass validation and all steps complete. This primitive uses no motion planner and imposes no additional terminal-pose tolerance.
`hand_move`	Endpoint position error is at most $3\,$mm; endpoint and maximum path rotation error are at most 3°; the 22-D finger command is unchanged. Any IK, planning, collision-recheck, runtime-collision, or tracking failure restores the pre-call state.
`release`	Arm and hand interpolation speeds are at most 0.20 and $0.40\,$rad/s, respectively, and no contact may remain between the target object and the active side at the endpoint. A construction or execution failure restores the pre-call state.
`reset`	Both arms and both hands must reach their HOME joint configurations defined in Appendix F.1. The base, legs, and head remain at their current configurations. cuRobo planning failure, collision, tracking failure, or timeout fails the call.
`grasp_vlp`	The grasp-and-lift protocol must complete, and the target must contact the active hand and no other geometry at the final check. Planning or IK failure, timeout, and incomplete execution fail the call. Grasp type, contact region, and finger-role agreement are separate evaluation fields.
`trigger_vlp`	The target lever must reach the scene-prespecified angular range. During the actuation phase, contact coverage between the hand and target lever must be at least 0.70, and simultaneous contact coverage of the target lever by the thumb and index finger must be at least 0.60; the hand must not contact the lever during the withdrawal phase.
`push_vlp`	The target drawer must reach the scene-prespecified position range along its rail. During the pushing phase, contact coverage between the hand and handle must be at least 0.70; the hand must not contact the handle during the withdrawal phase.
`pull_vlp`	The target drawer must reach the scene-prespecified position range along its rail. During the pulling phase, contact coverage between the hand and handle must be at least 0.70; the hand must not contact the handle during the withdrawal phase.
`rotate_vlp`	The target knob must reach the scene-prespecified angular range. During the rotation phase, contact coverage of the target knob by the thumb and index finger must be at least 0.80.
`slide_vlp`	The target object must slide along the tabletop and reach the scene-prespecified planar position range. During the sliding phase, contact coverage between the hand and target object must be at least 0.70, and the target object must neither topple nor leave the tabletop; the hand must not contact the target object during the withdrawal phase.
`open_vlp`	The target door joint must reach the scene-prespecified angular range. During the opening phase, contact coverage between the hand and handle must be at least 0.70; the hand must not contact the handle during the approach or withdrawal phase.
`close_vlp`	The target door joint must reach the scene-prespecified angular range. During the closing phase, contact coverage between the hand and handle must be at least 0.70; the hand must not contact the handle during the approach or withdrawal phase.
`extract_vlp`	The target tissue must be pulled through the tissue-box opening and reach the scene-prespecified extraction-length range. During the extraction phase, contact coverage between the hand and target tissue must be at least 0.70, and the hand must not contact the tissue box.
`press_vlp`	The target button latch must transition from disengaged to engaged. During the pressing phase, contact coverage between the index finger and the top surface of the button must be at least 0.95.

requirements, when annotated, are evaluated separately from the physical reward. Table 10 lists the primitive completion and failure criteria; Table 11 specifies the record returned by the frozen protocol.

Physical acceptance. A call is executed by the protocol of Appendix F.4, which returns the frozen configuration it ran under, the outcome of each phase and one acceptance bit. For a grasp the scene success rule of Appendix C.3 is strict active-hand-only contact: after the $0.5\,$s hold that follows the vertical lift, the target must touch at least one collision geometry of the active hand and no other

geometry, and the call must have completed its phases, so the bit is `protocol_complete` $\wedge$ `active_hand_contact_count` $> 0 \wedge$ `forbidden_contact_count` $= 0$. The lift itself is accepted when the realized vertical displacement stays within $4\,\text{cm}$ of the $0.10\,\text{m}$ lift executed at $0.025\,\text{m/s}$ and the orientation drift stays below $30°$, with the squeeze hold and the post-lift hold each held for $0.5\,\text{s}$. Failures are reported as codes (`executed_vertical_lift_short_of_10cm`, `executed_vertical_lift_exceeds_10cm`, `executed_lift_orientation_drift`, `vertical_ik_failed_at_tick_`$\star$), a call whose online plan is rejected is not accepted, and a call that times out or that is terminated by a safety stop is not accepted.

The other primitives are accepted on their own recorded quantities: `press_vlp` when the actuated surface changes state exactly once during the call under sustained normal contact on the named surface, `pull_vlp` when the realized axial displacement reaches the requested displacement along the named axis with the named contact set maintained, and the base-motion and body-height primitives (`xy_move`, `yaw_rotate` and `z_move`) when the realized pose of the moved frame reaches the target with the motion completed.

Semantic agreement. For examples with grasp-type, contact-region, and finger-role annotations, semantic agreement is defined by a separate frozen rule set over the physical trace. It is excluded from the reasoner reward and the physical grasp success rates of Section 4.2.

For a grasp with requested grasp type $\gamma^\star$, contact region $r^\star$, and finger roles $\phi^\star$, the rules infer the realized type from the contact topology of the call, including which geometries of the active hand touch which annotated regions, which finger provides opposition, and how much the palm contributes. They also compute the realized contact-region set $\{r\}$ and the realized finger roles, and agreement requires the realized type to match the request, the requested contact region to be part of the realized set, and the requested opposing fingers to be the load-bearing set.

The broader interface can express surface and finger requirements for pressing, recess and contact requirements for pulling, and approach or posture requirements for base and body motion. Evaluating these fields requires the corresponding annotations and observable execution evidence.

Episode return. Tabletop task success and the counterfactual episode return use the same strict binary physical criterion:

$$R_V(v_{1:K}) = \mathbb{1}\left[\bigwedge_{i=1}^{K} \text{pass}(v_i) = 1 \ \wedge \ G_{\text{grasp}}(v_K) = 1\right]. \tag{20}$$

Here G_{grasp} is the terminal physical grasp-success condition for the target task. A rejected call, timeout, safety stop, missing required call, or failed terminal grasp sets the return to zero. A failed replay branch terminates, and any unexecuted suffix calls receive zero acceptance bits (Appendix C.4); semantic agreement remains an evaluation-only field.

Reasoner-visible projection. For the current-observation policy, o_k consists of the four current RGB views and target-object information; $h_{k+1} = (g, o_k)$ also contains the task instruction. The full-history condition adds preceding visual observations and executed calls. Image preprocessing and serialization follow the policy settings in Appendix C.2. Oracle poses, simulator contacts, point clouds, and the semantic evaluation fields are excluded from both reasoner formats. The grasp VLP receives its own calibrated RGB views, grasp description, and active-hand identifier, as specified in Appendix A. Each call produces one verifier record for physical return computation in Eq. 20.

C.4 Counterfactual rollout collection

The reference-rollout collection procedure is described in Section 3.4.

Sibling generation. Each tested boundary records the behavior-policy call together with the configured sibling alternatives. The default policy group draws independent, schema-constrained samples $a_{k,j} \sim \pi_b(\cdot \mid h_k)$ under separate decoding seeds; local action-neighborhood candidates carry an explicit source tag and are retained in the physical counterfactual record. Each sibling runs once

Table 11: Fields of one verifier record, as written by the frozen execution protocol. A record is nested: the protocol block carries the frozen configuration the call ran under and its phase outcome, the success block carries the frozen rule and the acceptance bit, and the trace carries one row per control tick. A record whose call never executed carries its status, the target and the rule identifier, with the lift and the trace empty. The contact-derived entries (`success.final_target_contacts`, `success.contact_pass` and the contact counts in the trace) are read from the simulator's contact list and therefore exist only where that list exists; the reasoner receives none of them.

Field	Type	Content
`side`	string	hand side that executes the call
`target_object_id`	string	target object of the call
`prepared`	dict	the 58-D policy state measured when execution starts and the 58-D command the protocol starts from
`squeezed_target_29d`	vector	the `squeezed_29` half of the 58-D target of Appendix F.3 that the protocol executed
`evaluation_status`	string	`invalid_prediction`, `online_plan_rejected`, `complete` or `protocol_failed`
`protocol`	dict	the frozen configuration the call ran under: command rate, arm and hand joint speed limits, squeeze and post-lift hold times, lift distance and speed, IK tolerances, executed-lift position and rotation tolerances, target validation, actuator-saturation policy, IK solver and branch-planner identifiers, joint-limit clip count, `protocol_complete` and the failure codes
`executed_lift`	dict	lift start, target and final end-effector positions, realized displacement, orientation drift, and the per-tick IK records with their residuals, joint corrections and integrated bias
`success.rule_id`, `success.definition`	string	identifier and definition text of the frozen success rule that was applied
`success.protocol_complete`, `success.protocol_failures`	bool, list	phase completion and the failure codes behind it
`success.final_target_contacts`	dict	contact snapshot at the end of the hold: per contact the target geometry, the other geometry, its category (active hand, other hand, wrist, arm or body, table, other scene collision), whether it is allowed and its distance, with the counts `active_hand_contact_count`, `forbidden_contact_count` and `target_contact_count`
`success.contact_pass`, `success.success`	bool	the active-hand-only contact condition and the acceptance bit, the latter being the conjunction of phase completion with the former
`passed`	bool	the acceptance bit the caller aggregates
`trace`	list	one row per control tick: `phase`, `tick`, `simulation_time_s`, end-effector and target-object positions, `active_hand_contact_count` and `forbidden_contact_count`

after restoring the full simulator and random-number-generator state; no reranking or best-of-N selection is used. Only policy-sampled siblings with valid action spans and behavior log probabilities are eligible for the PPO training view.

The scan runs backward from each rejected primitive to the first boundary with a rescuing sibling; the recorded branch must replay reproducibly, otherwise the snapshot and its siblings are discarded and the count is reported alongside the coverage statistics, and simulator repetitions never enter the estimator. The rescue probe of Section 4.3 replays valid calls in a neighborhood of each recorded planar action under the same snapshot and fixed-suffix protocol.

Branch execution and snapshots. After injecting $a_{k,j}$ the recorded symbolic suffix $a_{k+1}, \ldots, a_K$ is replayed through the execution client without querying the reasoner, so the VLP, motion planner and controllers see the state produced by the intervention. A rejected call, execution failure, timeout, or safety stop terminates the branch with zero return. The failed call and any remaining suffix calls receive $\mathrm{pass} = 0$, following the termination rules of Appendix F.4.

Before call a_k the client snapshots σ_{k-1}: the full simulator state (body poses and velocities, controller integrator states, contact solver state), the random state of every stochastic stream (reasoner decoding, physics, sensor noise, rendering), the recorded observation o_{k-1} and the recorded executor status. Snapshots are keyed by `episode_id`/`call_index`, carry the hash of the random state they restore, and are released once their group is consumed by data export. The recorded branch ($j = 1$) must replay with identical discrete verifier fields and continuous fields inside fixed tolerances around the recorded values. For fidelity checks, the reference branch reuses its recorded grounding, whereas a counterfactual branch re-renders and re-grounds from the intervened state.

Unlocalized failures and stored records. A rejected call whose scan finds no rescuing sibling is *unlocalized*: it receives no call span, it does not enter $\mathcal{G}$ and it does not enter PPO, but its trace and diagnostics are kept for the coverage statistics and for data export. Each localized episode contributes one boundary k, its sibling executions, diagnostics $d_{k,j}$, and call spans. Repeated processing reuses this group, so it enters the counterfactual data record and PPO minibatch once; the same call index in two episodes gives two groups with their respective contexts h_k. The trainer writes one record per call: the reference call with the reasoner-visible history, the executed action and the verifier fields; the sampled siblings with their branches, rescue bits, returns, wall and simulated time and behavior-policy log probabilities; and the localized boundary of the terminal failure, or the fact that it is unlocalized. The counterfactual-data loader takes traces, calls, returns and diagnostics; the PPO loader takes the histories, call spans, advantages and the tokenwise behavior-policy log probabilities, computed once at generation from that round's frozen behavior reasoner and held fixed throughout its PPO update; the semantic evaluation block goes to the evaluation record only.

C.5 Rescue distillation

Rescue distillation is a supervised baseline initialized from the same robot-plus-general SFT Reasoner as the two PPO conditions. Verified replacement calls from counterfactual replay provide supervised targets under their recorded contexts. Each of the three optimization conditions starts independently from the SFT model.

Training. Supervised fine-tuning trains the LoRA adapter of the SFT reasoner on the verified records: rank 16, $\alpha = 32$, dropout 0.05 on the linear layers of the language model (116.7M trainable parameters), with the vision encoder, connector, embeddings, and base weights frozen. The 559 verified planar-move replacement records are split by object asset into 486 training, 18 validation, and 55 test examples. Loss covers the next-primitive call and closing tokens, with the fixed rationale prefix masked. We use AdamW (learning rate 2×10^{-5}, weight decay 0.01, warmup 3%, cosine schedule), a gradient-norm limit of 1.0, bfloat16 with activation checkpointing, and batch size 1 with gradient accumulation 8. Two GPUs yield an effective batch size of 16. The additional supervised stage takes at most 0.55 hours (1.10 GPU-hours), including initialization, in-training validation, and saving. This upper bound spans the final pre-training input check through saving the evaluated model. The shared SFT initialization and collection of verified replacements are outside this interval.

C.6 Counterfactual RLVR

The SFT Reasoner initializes counterfactual PPO directly, with $\theta_0 = \theta_{\mathrm{ref}}$. The execution stack remains fixed after SFT. Each failed reference rollout is replayed from saved states, and the resulting sibling actions, physical returns, localized boundary, leave-one-out advantages, and token spans are exported as call-level supervision. The masked PPO stage uses each round's policy-sampled records for a bounded number of minibatch epochs, then refreshes the behavior reasoner for the next round's rollouts and sibling sampling. The SFT policy remains the fixed KL reference; semantic evaluation fields are withheld from the training projection. The reward of a candidate is the physical return of its single replayed branch by Eq. 6, and its advantage contrasts that return with the other siblings of the same boundary; a candidate is replayed once, and a call that fails schema validation or exhausts the decision budget does not meet the success criteria and scores zero. A localized rescue whose branch returns do not differ yields a zero contrast and no return-based gradient.

The local rescue indicator selects the call to refine, whereas the full-episode sibling returns determine its leave-one-out advantages (Eq. 6). The policy term and KL penalty are evaluated on the attributed primitive-call tokens under the recorded Reasoner context (Eq. 9).

Group sampling and log probabilities. For a condition c the policy-sampled group issues N independent plan requests with $\mathrm{rollout_id}_i = \mathrm{uuid5}(c, s_b, i)$ and $\mathrm{seed}_i = s_b + i$ for a base seed s_b, so a group is exactly reproducible and no two samples collide in the rollout cache. The sampler stores per-token log-probabilities under the distribution it actually samples from, with the configured temperature and the schema constraint applied, and the corresponding raw model log-probabilities separately; the ratio of Eq. 8 uses the former, and the summed `logprob_total` is telemetry for joining and monitoring rather than a substitute for the per-token values. One telemetry record is written per sample (`schema`, `rollout_id`, `seed`, `temperature`, `description`, `logprob_total`,

num_tokens, side, target_object_id) and joined with the rollout results by rollout_id; a record whose sampling log-probabilities were not captured is not eligible for the weight update. When a record already carries a reference log probability, the reference model is not loaded.

Weight update. Advantages are normalized within an episode group by Eq. 10. The update maximizes Eq. 9, with the surrogate clipped at $\epsilon_{\text{clip}} = 0.2$ and the reverse-KL anchor at $\beta = 0.05$; the ratio is formed from each record's per-token log-probabilities under the behavior policy that produced it, so records that do not carry them are ineligible. Training uses AdamW at 1×10^{-6} with a gradient-norm limit of 1.0 and batch size 4 with gradient accumulation 8, for one epoch over each round's records. Only the LoRA parameters are trained. A weight update consumes the round's policy-sampled records and their replayed branches.

Training setup. The optimization conditions in Table 1 update the same adapter of the robot-plus-general SFT Reasoner. Rescue distillation uses two training GPUs and an effective batch of 16; Appendix C.5 reports its incremental training cost. Counterfactual PPO and episode-return PPO each train on eight GPUs, split evenly between the trainer's policy devices and the simulation devices that render and step the parallel environments of Appendix F.5. Episode-return PPO uses minibatch-centred episode returns at the verifier-observed failing call or successful terminal call (Appendix E.2); a minibatch with identical returns contributes no return-based policy gradient. Counterfactual PPO completes eight collection-and-update rounds in 14.96 hours (119.68 GPU-hours).

D Primitive-Level VLP Evaluation

This appendix evaluates the VLP primitives. The grasp experiment uses the test split defined in Appendix B.3 to compare physical success across methods, while the remaining primitives are evaluated individually in single-primitive physical tests.

D.1 Grasp benchmark protocol

Test set and execution. The grasp test set comprises the 72,908 examples in the test split of Appendix B.3.

For G3+Sharpa, execution and physical success follow Appendices F.4 and C.3, including execution of the two action keyframes, post-grasp lift and hold, and checks of contact, orientation, collision, and execution completeness. The commanded lift is $0.10\,\text{m}$ with a tolerance of $\pm 0.04\,\text{m}$. We note when a baseline uses another embodiment or execution protocol.

Grasp execution initializes the active arm and hand at the predicted pregrasp after bounded contact-normal repair. It then holds the pregrasp hand joint angles while interpolating the seven arm joints to the squeeze arm joint angles, followed by interpolation of the 22 hand joints to the squeeze hand joint angles. The lift-and-hold protocol follows.

D.2 Grasp baseline comparisons

Pretrained Dexonomy model. We use the released weights without fine-tuning on our data. The model takes a single-view depth point cloud with simulator ground-truth object segmentation. It generates 100 candidates per scene; one is selected by model density and executed by the native free-floating Shadow Hand using pregrasp–grasp–squeeze. The test scene retains the table. Success requires the object to remain held at the end and its center of mass to rise by 0.06–$0.14\,\text{m}$. Because Dexonomy uses a different embodiment and executor, its result is a cross-embodiment reference (Chen et al., 2025).

DexGrasp Anything. We retain its point-cloud-conditioned diffusion model and geometric loss (Zhong et al., 2025), adapt its output to the executed squeezed joint configuration of the seven-joint G3 active arm and 22-joint Sharpa hand, and train separate left- and right-hand models on the same data split. The input is a complete object surface point cloud with normals. A fixed adapter converts the predicted configuration into an action and constructs a collision-free pregrasp. Sub-

sequent grasping and lifting follow the same execution procedure and physical success criterion as GraspVLP in this comparison.

GraspVLP (ours). We report the distilled student. It receives only four RGB views, the language instruction, and the active-hand identifier. On the same 72,908 test examples, it achieves 70.42% physical success (Table 12).

Table 12: Physical grasp success on the same test-scene manifest. Each method is evaluated under the execution protocol specified above.

Method	Physical SR (%)
Dexonomy	29.45
DexGrasp Anything	48.91
GraspVLP (ours)	**70.42**

D.3 VLP DISTILLATION ABLATION

Test-set evaluation. All models in Table 13 are evaluated physically on the test split defined in Appendix B.3. The GT-only and distilled RGB student pair isolates teacher-flow supervision with initialization held fixed. Table 2 reports the same primitive results alongside the separate random-start closed-loop test.

Table 13: Physical grasp success under the shared two-keyframe execution protocol. All conditions use the same test scenes, descriptions, and sampling seeds; the student receives only four RGB views, language, and the active-hand identifier at deployment.

Model	Cloud at inference	Physical SR (%)
Point-cloud-conditioned teacher	Yes	79.95
RGB student, GT-only	No	67.87
RGB student, distilled (ours)	No	70.42

The GT-only RGB student succeeds on 49,480 of the 72,908 test examples and fails on 23,428, giving a physical success rate of 67.87%.

In the closed-loop comparison (Table 2), success rates are 72.28% for the teacher, 47.52% for GT-only RGB, and 56.44% for distilled RGB. The distilled–GT-only difference is $+8.91$ percentage points.

D.4 OTHER PRIMITIVE BENCHMARKS

We also evaluate `trigger_vlp`, `push_vlp`, `pull_vlp`, `rotate_vlp`, `slide_vlp`, `open_vlp`, `close_vlp`, `extract_vlp`, and `press_vlp`. Each primitive uses its own operation descriptions and two-keyframe targets in the sample format of Appendix A.1, and is tested separately on a fixed set of initial scenes and operation instructions using the corresponding physical completion criterion of Appendix C.3. The teacher–student comparisons in Appendices B and D.3 concern the grasp model. Invalid outputs and failed executions remain in the denominator. Table 14 reports the number of tests and physical success rate for each primitive.

For example, the drawer-pulling test covers different handle geometries and pull directions. Physical completion requires the drawer to reach the prescribed displacement along its axis of motion.

Table 14: Single-primitive physical test results for the other VLPs. Each row uses its own fixed test set and completion criterion.

VLP primitive	Tests	Physical SR (%)
`trigger_vlp`	1,925	79.58
`push_vlp`	1,579	49.46
`pull_vlp`	1,098	48.18
`rotate_vlp`	265	80.75
`slide_vlp`	117	57.26
`open_vlp`	179	79.89
`close_vlp`	67	65.67
`extract_vlp`	179	38.55
`press_vlp`	192	100.00

E System Evaluation

E.1 Closed-loop grasp benchmark

Scenes and initial states. The benchmark evaluates cross-step dependencies in tabletop grasping with seen and held-out object assets. These labels refer to the Reasoner SFT object split; VLP data use a separate split by source demonstration. Assets are sampled uniformly within each split. Object positions and orientations follow the empirical annotation-validation distribution, and robot base positions and headings are randomized independently. All conditions use paired initial scenes.

The instruction asks the robot to pick up the target with its right hand. The reasoner receives the instruction, target identity, and four freshly rendered RGB views at each boundary. It selects one primitive at a time, and execution continues from the live state for at most 12 decisions, with at most one physical grasp attempt. The final grasp call obtains VLP targets from the current observation and executes the approach from the current robot configuration.

Shared execution and verification. Reasoner comparisons share the distilled RGB VLP, motion controllers, execution budgets, and fixed Verifier. A failed primitive terminates the episode. The RGB VLP comparison varies teacher-flow supervision while keeping all other system components fixed. Appendix F.4 describes how rule-based control connects primitive targets.

Success and failure stages. Complete-task success requires all calls to pass and completion of the approach, grasp, and lift, with the object contacting only the active hand at the final check, following Eq. 20. The same frozen acceptance rules apply to every condition. All assigned episodes enter the denominator, including invalid calls, planning failures, timeouts, and exhausted decision budgets. Physical grasp success is distinct from semantic agreement (Appendix C.3).

Paired comparisons and reporting. Methods are evaluated on the same initial scenes. We report physical success rates and paired differences in percentage points for the evaluated policies. Training conditions are described in Appendix C.2.

Figure 11 illustrates three closed-loop evaluation rollouts of the trained reasoner and VLP.

E.2 Reasoner optimization evaluation

Optimization comparisons. Table 1 compares three independent updates of the same robot-plus-general SFT Reasoner: rescue distillation, episode-return PPO, and counterfactual PPO. Rescue distillation fits verified replacement calls by supervised learning. Both PPO conditions start directly from the SFT model and retain it as their fixed KL reference. All conditions share the prompt and execution stack. Episode-return PPO attaches a minibatch-centred episode return to one span per collected episode: the verifier-observed failing call for a failing episode, and the terminal call that completes the task for a successful one. Successful episodes therefore share the minibatch and give the control a non-degenerate baseline; a batch whose episode returns do not vary yields no return-based gradient. Counterfactual PPO uses the localized call and sibling leave-one-out advantages in Eq. 6. Both use the masked update of Appendix C.6.

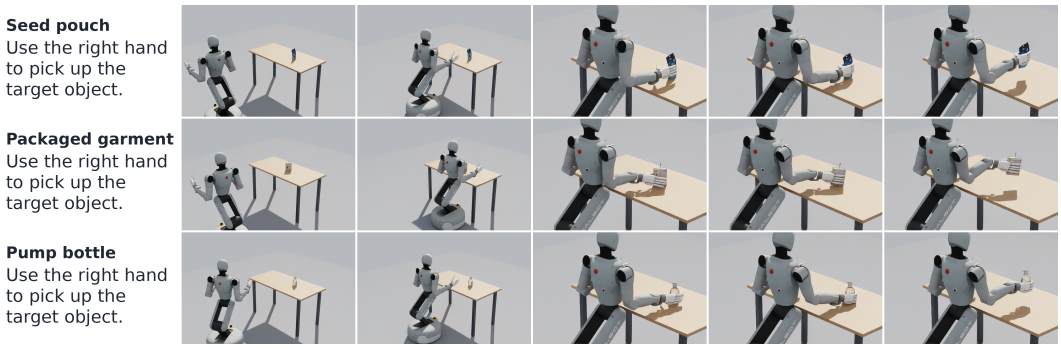


Figure 11: Closed-loop rollout examples for a seed pouch, a packaged garment, and a pump bottle (top to bottom). The left column lists the task instruction and target. The five image columns show the initial state, an intermediate navigation state, pregrasp, hand closure, and lift. Navigation states come from archived evaluation rollouts; grasp states come from physical replay of their saved VLP targets. All panels are re-rendered in Isaac Sim.

Single-attempt evaluation. Each condition runs one closed-loop episode per task from the same initial state, with common decoding settings and execution budgets. The table's physical success requires every required physical subgoal and the terminal goal to pass within the budget. Fine-grained semantic agreement is a separate field. Invalid outputs, planning failures, and exhausted budgets remain in the denominator. Mixed-data SFT achieves 56.44% success. Rescue distillation achieves 42.57%, a change of -13.86 percentage points relative to its SFT initialization. Episode-return PPO achieves 55.45% overall, with 56.60% on seen objects and 54.17% on held-out objects. Counterfactual PPO achieves 62.38% overall, with 62.26% on seen objects and 62.50% on held-out objects. Its overall differences are $+6.93$ percentage points relative to episode-return PPO and $+5.94$ points relative to SFT. The counterfactual replay probe is evaluated separately from policy performance.

Counterfactual replay probe. Of 841 eligible failed reference episodes, 416 admit a full-return rescue. The probe executes 6,115 replacement branches, of which 559 succeed; all 1,923 tested replay boundaries pass the fidelity check. These counts underlie the supervision-yield percentages in Section 4.3 and are separate from closed-loop policy evaluation.

E.3 Reasoner configuration comparisons

All reasoner configurations are evaluated with the same distilled RGB VLP and fixed Verifier. The observation convention and VLP ablation are described in Appendices C.2 and D.3.

Observation context and training data. The current-observation and full-history reasoners share Qwen3.8-27B, rank-16 LoRA, robot decision targets, initialization, optimizer, and training-update count. The history model additionally receives earlier images and executed calls. The expanded-data current-observation model uses 80% robot and 20% general-data updates. This recipe changes robot-data scale, training mixture, and optimization budget together (Appendix C.2).

Full-history robot-only SFT achieves 59.41% success, compared with 56.44% for current-observation robot-only SFT: a difference of $+2.97$ percentage points. Robot-plus-general SFT also achieves 56.44%, giving a paired difference of 0.00 percentage points from current-observation robot-only SFT. The mixed-data comparison evaluates the combined training recipe: data scale, mixture, and optimization budget change together.

Comparison and failure analysis. Table 15 reports complete-task success; Appendix E.4 separates decision, approach, acquisition, and lift outcomes.

Reasoner configuration	Seen (%)	Held-out (%)	Overall (%)
Cosmos3-Nano (pretrained)	0.00	0.00	0.00
+ SFT	26.42	31.25	28.71
RoboBrain2.5-8B-NV (pretrained)	0.00	0.00	0.00
+ SFT	35.85	41.67	38.61
Qwen3.8-27B (pretrained)	5.66	6.25	5.94
Current-observation robot-only SFT	58.49	54.17	56.44
Full-history robot-only SFT	58.49	60.42	59.41
Robot + general SFT	**66.04**	45.83	56.44
+ Rescue distillation	37.74	47.92	42.57
+ Episode-return PPO	56.60	54.17	55.45
+ Counterfactual PPO	62.26	**62.50**	**62.38**

Table 15: Closed-loop success on paired randomized scenes with a shared RGB VLP Executor and fixed Verifier. The pretrained Qwen baseline uses JSON-schema-constrained decoding; SFT and rescue-distilled models use ordinary greedy decoding. Best scores are in bold.

External reasoner baselines. We adapt the autoregressive reasoner of Cosmos3-Nano (NVIDIA, 2026a) and RoboBrain2.5-8B-NV (Tan et al., 2026) with rank-16 LoRA on 24,000 decisions from successful training rollouts, balanced across `xy_move`, `yaw_rotate`, and `grasp_vlp`. Visual components remain frozen. Each model receives eight hours of training on eight GPUs, with effective batch size 32 and learning rate 5×10^{-5}. Evaluation uses four current RGB views, the common primitive-call interface, and the shared frozen VLP and Verifier. These are system-level comparisons: architecture, image processing, training data, and optimization budget differ from Qwen3.8-27B robot-plus-general SFT. Both pretrained models achieve 0.00% complete-task success with the shared primitive interface.

E.4 Execution failure analysis

Each episode receives one terminal outcome. Unsuccessful episodes are grouped into decision-budget exhaustion, preparation or approach failure, failed object acquisition, and loss during lift.

Terminal outcome	Robot-only SFT	Robot + general SFT	+ Rescue distillation
Successful grasp and lift	56.44	56.44	42.57
Decision budget exhausted	0.99	8.91	16.83
Preparation or approach failure	5.94	5.94	7.92
Object not acquired	24.75	11.88	16.83
Object lost during lift	11.88	16.83	15.84

Table 16: Terminal outcome proportions (%) with a shared RGB Executor and fixed Verifier. Both SFT models use current observations. Rescue distillation starts from the mixed-data model. Each episode contributes to one outcome; rounding may prevent columns from summing to exactly 100%.

E.5 General capability evaluation

General-capability evaluation. We compare pretrained Qwen3.8-27B with the current-observation SFT model trained with 80% robot and 20% Vision-Flan optimizer updates. Both receive identical benchmark prompts and images, use greedy decoding with thinking disabled, and have a 4,096-token answer budget. Truncated or unparseable responses count as incorrect. Excluding exact image or question overlaps with the general training corpus leaves 4,317 MMBench English development records (1,161 question groups), 2,612 CV-Bench test records, and 1,899 BLINK validation records.

MMBench circular accuracy requires all supplied answer-order variants of a question to be correct. CV-Bench averages ADE20K and COCO accuracy for its 2D score, then averages this with the Omni3D 3D score. BLINK averages accuracy equally across its 14 tasks.

Model	MMBench ↑	CV-Bench ↑	BLINK ↑
Qwen3.8-27B (pretrained)	85.01	84.60	61.48
+ Robot and general SFT	**85.19**	**87.17**	**67.21**
Change (percentage points)	+0.17	+2.57	+5.72

Table 17: General visual-language performance before and after robot-plus-general SFT. Scores (%) use MMBench circular accuracy, CV-Bench source-weighted accuracy, and BLINK task-macro accuracy on the filtered evaluation sets. Changes subtract the pretrained score from the SFT score before rounding. Best scores are in bold.

The paired change on MMBench is $+0.17$ percentage points. CV-Bench and BLINK increase by 2.57 and 5.72 points, respectively. These results characterize the combined robot-plus-general SFT recipe.

E.6 Primitive-call adaptation

Adaptation of grasp-description models. This diagnostic starts from vision-language models fine-tuned to produce English grasp descriptions from four RGB views and an instruction. Additional LoRA training uses robot primitive-decision examples to adapt them to the next-call JSON interface. Table 18 compares their complete-task success with our counterfactual PPO reasoner. Physical evaluation uses paired scenes, the distilled RGB executor, and a fixed Verifier. Success ranges from 47.52% to 57.43% across the adapted baselines, while Ours achieves 62.38%. Model family, training objective, and training budget vary across this comparison.

Model	Task SR (%)
Qwen3.5-0.8B	54.46
Qwen3.5-2B	54.46
Qwen3.5-4B	57.43
Qwen3.5-9B	55.45
Qwen3-VL-2B	51.49
Qwen3-VL-4B	53.47
Qwen3-VL-8B	47.52
Ours	**62.38**

Table 18: Primitive-call adaptation of grasp-description models compared with counterfactual PPO (Ours; Table 1). Task SR measures complete closed-loop execution.

E.7 Runtime efficiency

We measure model-service latency during simulator evaluation of the robot-plus-general SFT Reasoner and distilled RGB VLP on RTX PRO 5000 72 GB GPUs at batch size one. Mean, median, and 90th-percentile latencies are 6.09/5.33/10.45 s for the Reasoner and 0.99/0.93/0.96 s for the VLP, respectively. Statistics pool calls from both successful and failed episodes and retain first calls.

The Reasoner timer covers input processing, generation, and output decoding; the VLP timer covers input-file loading, RGB encoding, 20-step Heun sampling, and output transfer to CPU. Model loading is outside these timers. These service measurements precede motion planning and physical execution.

F Execution Infrastructure

This section describes the executable interface between the reasoner and the robot. We first specify the robot configuration and the available rule-based and VLP-based primitives, then define primitive-call representation, execution, and termination, and describe the capture, deployment, and serving infrastructure. Verification and reward construction are specified in Appendix C.3.

Table 19: Simulation-physics, robot, and camera parameter domain randomization. All parameters are sampled once at each scene initialization and held fixed throughout a rollout. Camera translation and rotation ranges are symmetric uniform bounds about the calibrated poses.

Group	Parameter	Range
Simulation physics	Object mass and inertia	Mass scale $\mathcal{U}(0.8, 1.2)$; inertia scaled identically
	Contact friction and restitution	Friction scale $\mathcal{U}(0.7, 1.3)$; restitution $\mathcal{U}(0, 0.1)$
Task geometry	Tabletop height	0.78 ± 0.04 m
Robot dynamics	Link mass/inertia, joint stiffness/damping, and actuator gains	Scale $\mathcal{U}(0.95, 1.05)$ relative to nominal values
Camera extrinsics	Head stereo rig	± 2 mm and $\pm 0.5^\circ$ per axis
	Left/right wrist cameras	± 5 mm and $\pm 1.5^\circ$ per axis
Camera intrinsics	Default pinhole model	Fixed; 110° horizontal field of view and centered principal point

F.1 Robot platform

We use the full Galbot G3 robot model. The robot has five leg joints, two head joints, seven joints in each arm, and 22 joints in each Sharpa five-fingered hand, for a total of 65 actuated revolute joints. A single arm–hand configuration is 29-D (7-D arm and 22-D hand). The base can translate and rotate. Figure 12 shows the simulated robot and tabletop manipulation setup. We manually define a standby joint configuration, HOME, comprising all 65 joint angles; Table 20 lists its values.

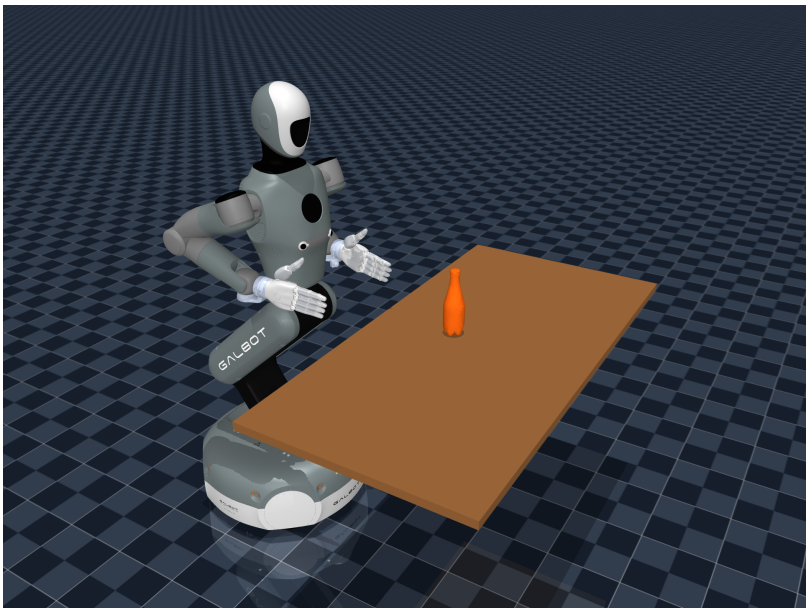


Figure 12: Simulation setup with the Galbot G3 robot, Sharpa dexterous hands, and tabletop workspace.

Table 19 summarizes the domain randomization applied to physical, robot, and camera parameters during MuJoCo physics simulation and Isaac Sim rendering.

Platform and software. Hardware execution uses the Galbot G3 platform described above. Two head and two wrist RGB cameras supply policy observations. One workstation serves online inference. In simulation, reasoner observations and qualitative figures use Isaac Sim rendering, while HOME grasp-VLP observations use MuJoCo rendering. Simulation physics uses MuJoCo (Todorov et al., 2012), planning and inverse kinematics cuRobo (Sundaralingam et al., 2023), and point

Table 20: Actuated joints, ranges, and HOME configuration of the Galbot G3 model. Ranges and HOME values follow joint order within each row, in radians; HOME values are rounded to four decimal places. A zero HOME value for a hand group applies to every joint in that group on both sides. The two hands have identical ranges, and each interval is both the joint limit and actuator `ctrlrange`.

Subsystem	DoF	Joint variable	Joint / command range	HOME
Head	2	`head_joint1:2`	$[-1.5208, 1.5208]$, $[-0.2143, 0.4936]$	$0, 0.3264$
Leg	5	`leg_joint1:5`	$[0, 0.9374]$, $[0, 2.5847]$, $[0, 2.3262]$, $[-1.5906, 1.5906]$, $[-0.1645, 0.1645]$	$0.4679, 1.2063, 0.6719, 0, 0$
Left arm	7	`left_arm_joint1:7`	$[-2.9869, 2.9869]$, $[-1.6953, 0.9972]$, $[-2.9171, 2.9171]$, $[0, 2.4807]$, $[-2.9171, 2.9171]$, $[-0.7354, 0.7354]$, $[-1.5382, 1.5382]$	$1.3584, -1.3683, -0.7771, 2.0104, -0.2580, 0.1606, 0.2607$
Right arm	7	`right_arm_joint1:7`	$[-2.9869, 2.9869]$, $[-0.9972, 1.6953]$, $[-2.9171, 2.9171]$, $[-2.4807, 0]$, $[-2.9171, 2.9171]$, $[-0.7354, 0.7354]$, $[-1.5382, 1.5382]$	$-1.3584, 1.3683, 0.7771, -2.0104, 0.2580, -0.1606, -0.2607$
Each hand, thumb	5	`CMC FE`, `CMC AA`, `MCP FE`, `MCP AA`, `IP`	$[-0.1745, 1.9199]$, $[-0.3491, 0.3491]$, $[-0.5236, 1.3963]$, $[-0.3491, 0.3491]$, $[0, 1.7453]$	0 (all 5)
Each hand, index/middle/ring	$3 \times 4 = 12$	`MCP FE`, `MCP AA`, `PIP`, `DIP`	per finger: $[-0.1745, 1.5708]$, $[-0.3491, 0.3491]$, $[0, 1.7453]$, $[0, 1.3963]$	0 (all 12)
Each hand, pinky	5	`CMC`, `MCP FE`, `MCP AA`, `PIP`, `DIP`	$[0, 0.2618]$, $[-0.1745, 1.5708]$, $[-0.3491, 0.3491]$, $[0, 1.7453]$, $[0, 1.3963]$	0 (all 5)

processing Open3D (Zhou et al., 2018). The reasoner and the VLP services are called over gRPC (Biswas et al., 2018) (Appendix F.6).

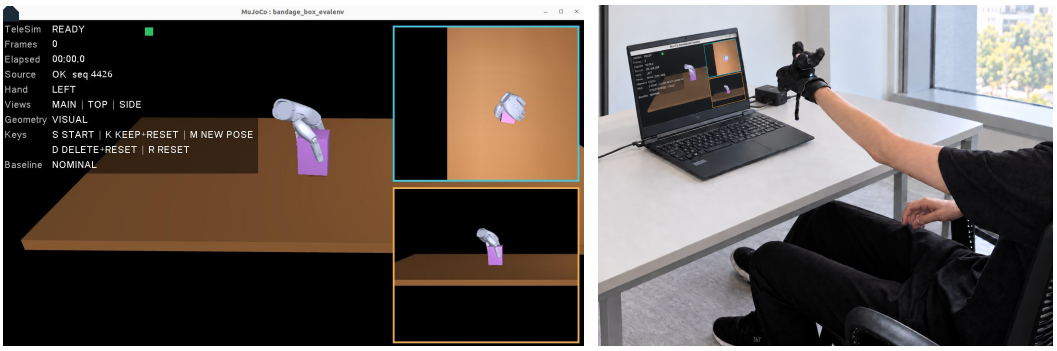
Pinning and seeds. Package versions and the driver, CUDA, and simulator stack are fixed for each experiment. Point-cloud reduction is seeded from the asset identifier and the mesh hash (Appendix B.1), and every saved call stores the simulator and generator state needed to reproduce the recorded verifier fields.

F.2 TELEOPERATION SETUP

Hardware. Each hand uses one MANUS Metagloves Pro glove (MANUS, 2026) and one VIVE Tracker. The glove provides a finger skeleton relative to its parent nodes, while the tracker provides the six-DoF wrist pose. SteamVR establishes the spatial reference frame, and the MANUS Integrated CoreLite SDK streams glove observations. Either or both hands can drive the corresponding Sharpa F100/WAVE models in MuJoCo (Todorov et al., 2012). Table 21 lists the components of the capture system; Figure 13 shows the simulation view and wearable setup.

Table 21: Components of the teleoperation data-capture system.

Component	Quantity	Role
MANUS Metagloves Pro	2	Capture finger skeletons
VIVE Tracker and Watchman Dongle	2 each	Track the left and right wrists
SteamVR Base Station 2.0	1 set	Establish the spatial reference frame
MANUS Sensor Dongle and Wireless Transceiver	1 each	Receive glove data
Sharpa F100/WAVE models	Left/right hands	Simulated 22-DoF hands



(a) Simulation interface (b) Human teleoperator

Figure 13: Teleoperation setup. (a) Simulation interface with main and auxiliary views. (b) Illustrative composite based on an operator photograph, showing the glove, wrist tracker, and workstation.

Calibration and synchronization. MANUS calibration adapts the finger measurements to the operator. Tracker zeroing maps the SteamVR room frame to a normalized operator frame; a four-pose calibration then maps this frame into the MuJoCo workspace and estimates the fixed tracker-to-palm rotation. Bimanual capture shares one workspace transform while retaining a separate mounting rotation for each hand. At 100 Hz, the gateway pairs each glove observation with the temporally nearest tracker sample. After validating device identity, calibration state, tracking state, and the hand skeleton, it emits a normalized data frame.

Retargeting and simulation. The system resolves node identifiers using the semantic hierarchy reported by MANUS. Recursive composition converts 25 parent-relative transforms into root-relative position and orientation keypoints. After a fixed coordinate transform, the official V4 `HandRetargetingOptimizer` maps these keypoints to 22 Sharpa joint targets, using the preceding frame and its solution to maintain temporal continuity. The two hands use independent optimizers, and the calibrated tracker poses provide the hand-root targets.

The Sharpa hands are simulated as dynamic bodies: a compliant weld tracks each hand-root mocap target, while joint actuators track the retargeted finger angles. MuJoCo advances the coupled hand–object dynamics at 500 Hz and records data at 100 Hz. Each frame retains the tracker and glove inputs, root and joint commands, realized hand and object states, and simulated contacts. Each episode also stores its calibration, scene and asset identifiers, and a quality summary.

F.3 Primitive interface

The primitive library defines 16 interfaces. The randomized tabletop benchmark uses `xy_move`, `yaw_rotate`, and `grasp_vlp`; the broader inventory in Table 10 specifies the available action and verification interfaces. We divide primitives into rule-based and VLP-based families according to how they are executed. Rule-based primitives invoke no learned model and are executed by predefined kinematic, motion-planning, or physical procedures. They comprise `xy_move` for base translation, `yaw_rotate` for base rotation, `z_move` for torso-height adjustment, `hand_move` for translating the selected hand, and `release`, which constructs and executes a release target through physics. The rule-based family also includes `reset`, which returns both arms and hands to HOME without moving the base, legs, or head. Table 23 lists the arguments that the reasoner must provide when invoking the rule-based primitives, their valid ranges, and their execution procedures. VLP-based primitives instead infer two target keyframes with their corresponding VLPs. The leg joints remain fixed during VLP-based primitives, and the head joints remain fixed throughout execution to stabilize the head-camera viewpoints.

Primitive invocation. When a VLP-based primitive is invoked, its VLP returns a 58-D arm–hand target comprising a 29-D pre-contact configuration and a 29-D goal configuration. A deterministic motion procedure executes the two configurations in sequence within that primitive call. Rule-based primitives are executed entirely by deterministic procedures. The reasoner returns one next-primitive

call at a time. Each call is a schema-constrained structured record whose fields are given in Table 22. Arguments inconsistent with the primitive schema are rejected.

Table 22: Primitive-call data structure. Each call selects one primitive by its identifier and must carry a set of arguments matching that primitive.

Field	Meaning
`primitive_id`	Unique positive integer identifier of the primitive
`args`	Arguments matching the primitive

Table 23: Input arguments, valid ranges, and execution procedures of the rule-based primitives.

Primitive	Reasoner arguments	Valid range	Execution
`xy_move`	`dx_m`, `dy_m`	$\|(\Delta x, \Delta y)\|_2 \leq 0.50\,\mathrm{m}$	Interpolate base positions along a straight line.
`z_move`	`dz_m`	$\|\Delta z\| \leq 0.20\,\mathrm{m}$; feasible under leg IK and joint limits	Solve leg IK along vertical waypoints while maintaining torso planar pose and orientation.
`yaw_rotate`	`yaw_rad`	$\|\Delta\psi\| \leq \pi$	Interpolate the base heading.
`hand_move`	`delta_m[3]`, `side`	$\|\Delta\mathbf{p}\|_2 \leq 0.30\,\mathrm{m}$; `side`$\in$ {left, right}	Plan a collision-checked arm path while maintaining hand orientation and finger configuration.
`release`	`side`	`side`$\in$ {left, right}	Construct a release target using an expanded object margin and interpolate from the current command.
`reset`	None	—	Use cuRobo to plan collision-checked return trajectories from the current configurations to the HOME targets for both arms and hands; keep the base, legs, and head fixed.

F.4 PRIMITIVE EXECUTION

All 65 joints use position actuators. The client receives joint-position commands at 50 Hz, linearly interpolates them, and sends targets at 250 Hz. Gravity compensation is enabled for the robot subtree in simulation. Both arms use PD control with $K_p = 100$ and $K_d = 40$. Each actuator's `ctrlrange` equals its joint limit, as listed in Table 20.

The system makes closed-loop decisions one step at a time: at each decision step, the client synchronously captures four images of the current state and asks the reasoner for one next primitive. After a successful nonterminal call, the client captures four new images and queries the reasoner again.

Primitive execution. When a VLP-based primitive is invoked, the robot first reaches HOME to capture the observation. The corresponding VLP then predicts two arm–hand keyframe targets, $\mathbf{y}^{\text{pre-contact}}$ and $\mathbf{y}^{\text{goal}}$. Motion planning and fixed controllers connect and execute the two keyframes according to the primitive's execution procedure. After the call, the fixed Verifier determines the outcome using the corresponding completion criterion in Appendix C.3. Table 23 specifies the execution of rule-based primitives.

Failure determination and termination. Invalid arguments, planning or IK failures, collision-check failures, unresolved remote-inference errors, and exceptions raised during execution terminate the rollout and are logged with their failure reason. A call rejected by its physical acceptance criterion also terminates the rollout. Table 10 gives the completion criteria of each primitive. Whenever a primitive completes successfully, the client invokes the scene-level success rule to determine from the MuJoCo state whether the complete task has succeeded.

The trained reasoner and distilled RGB grasp VLP remain frozen during hardware deployment. The reasoner receives four fresh RGB views and task context and emits one next primitive. For grasping, the VLP maps calibrated images, a grasp description, and the active-hand identifier to pregrasp and grasp arm–hand targets. Motion planning and rule-based controllers execute and connect calls; scene

geometry supports collision checking. Subsequent decisions use fresh observations. Privileged simulator records used for training are unavailable on hardware.

F.5 Distributed rollout environment

Training is distributed: the trainer holds the weights and the optimizer state, while rollout workers run the execution clients with the VLP server, Isaac Sim rendering, MuJoCo physics (Todorov et al., 2012) and the verifiers. Each optimization round collects reference rollouts on parallel environments, and counterfactual siblings are charged separately (Appendix C.6); PPO uses the settings of Appendix C.6.

Components, isolation and records. A process that can see oracle state never talks to the reasoner, and a process that talks to the reasoner never sees oracle state. State is keyed by `(worker_id, env_id, episode_id)` and nothing mutable is shared, so a worker restart loses only the episodes in flight, which are re-run from their initial configuration. Snapshots are immutable, file-backed and owned by one group; the trainer synchronizes at round boundaries, and a phase pins one configuration identifier throughout. That configuration is an immutable versioned artifact holding the weights, the prompt, the VLP checkpoint identifiers, the reasoner configuration, the verifier rule set and the training environment, published atomically, so a failed validation leaves the previous one in force. Three stores are written per call: the observation record, holding only the reasoner-visible projection o_k of Appendix C.3; the verification record, holding the protocol configuration, the executed lift and the success block of Table 11; and the evaluation record, holding the semantic block. The execution client reads only the first, the counterfactual-data and PPO loaders are allowlisted to traces, calls, returns, diagnostics, spans and behavior log probabilities, and the semantic block is read only by evaluation code.

Requests, responses and errors. `Reset` takes a scene identifier and a seed and returns the initial observation, `Step` submits one schema-validated call and returns the executor status with the next observation, and `Snapshot` and `Restore` implement the counterfactual machinery of Appendix C.4 for training workers only. Errors are either retryable transport faults (connection loss, queue overflow, batch allocation failure) or terminal outcomes (the execution failures of Appendix F.4, a target that fails its success rule, a timeout or a safety stop); only the first class is retried, and only before motion has started. A terminal outcome is written into the verifier record and handled by the execution client, and a call that exhausts its retries terminates the episode with that status. Every response carries the configuration identifier that produced it.

F.6 Model serving

Calls and statuses travel over one persistent bidirectional gRPC stream per worker; deployment views are raw `uint8` arrays in a shared-memory ring that reasoner and VLP read without a copy, the control message carrying only `(frame_id, timestamp, shape)`. Model calls are batched per family, the reasoner batches by token budget with power-of-two length bucketing, and each VLP batches fixed-shape inputs. Within each rollout, primitive selection, grounding, and execution follow the sequential loop in Appendix F.4.

Queues and measurement. The batching unit is one model request, one reasoner decision or one VLP grounding; queues are per model family, so a burst of grasp groundings cannot delay a press grounding, and an overflowing queue returns a retryable error. The reasoner queue and the VLP queues are scheduled independently; online hardware requests use a separate queue prioritized over training rollouts. Training reports calls and simulated steps per second. Model-service latency is reported in Appendix E.7.