DEXHOLDEM: PLAYING TEXAS HOLD'EM WITH DEXTEROUS EMBODIED SYSTEM

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating embodied systems with real dexterous hardware requires more than isolated motor-skill tests: an agent must perceive a changing scene (e.g. a tabletop), choose a context-appropriate action, execute it with a dexterous hand, and leave the scene usable for later decisions. We introduce **DexHoldem**, a comprehensive real-world benchmark evaluating Texas Hold'em related dexterous manipulations with a ShadowHand. DexHoldem provides 1,470 teleoperated demonstrations across 14 Texas Hold'em manipulation primitives, a standardized physical policy benchmark, and an agentic perception benchmark that tests whether agents can recover the structured game state needed for embodied decision making. On primitive execution, $\pi_{0.5}$ obtains the highest task completion rate (61.2%), while $\pi_{0.5}$ and π_0 tie on scene-preserving success rate (47.5%). On agentic perception, Opus 5.5 narrowly leads on both strict problem-level accuracy (49.1%) and average field-wise accuracy (80.6%); the gap between the two exposes the distance between isolated visual sub-capabilities and complete routing-relevant state recovery. Finally, we instantiate the full embodied-agent loop with one agent–policy pairing over 33 closed-loop hand-level rollouts, in which only 12.1% of hands complete; retries restore the failed primitive in 12 of 34 dispatches and resolve prolonged execution stalls in three of the four completed hands, which would otherwise have required manual termination. Only one hand completes with neither a retry nor a human-help request. DexHoldem therefore evaluates dexterous tabletop execution, agentic perception, and embodied decision routing in a shared physical setting. Anonymous Website at DexHoldem.

1 INTRODUCTION

Recent advances in robotics and embodied agents have expanded the range of behaviors that can be learned and evaluated, including instruction following and long-horizon task composition (Zitkovich et al., 2023; Chi et al., 2023; Intelligence et al., 2025; Driess et al., 2023; Huang et al., 2022; 2023; Liang et al., 2023; Liu et al., 2023; Chen et al., 2025; Zhang et al., 2024b; Mees et al., 2022). Yet evaluating these systems in realistic physical environments remains difficult. Existing embodied-agent benchmarks (Srivastava et al., 2021; James et al., 2019; Yu et al., 2020; Nasiriany et al., 2024) have advanced evaluation of language grounding (Liu et al., 2023; Mu et al., 2025; Chen et al., 2025) and planning (Zhang et al., 2024b; Mees et al., 2022; Mu et al., 2025), but many still rely on simulation, coarse action spaces, or gripper-centric manipulation (Chen et al., 2025; Liu et al., 2023; Mu et al., 2025; 2021). Consequently, these scores provide limited evidence for grounding instructions in physical scenes while executing precise, real-world multi-finger manipulation.

Benchmarks for dexterous manipulation address a complementary aspect of this problem by advancing contact-rich manipulation (Chen et al., 2022; Kumar et al., 2023; Fu et al., 2021; Bao et al., 2023; Tao et al., 2025), grasping (Chen et al., 2024; Wang et al., 2025b; Turpin et al., 2023; Ye et al., 2025; Zhang et al., 2024a), and in-hand manipulation (Charlesworth & Montana, 2021; Cruciani et al., 2020). However, these benchmarks typically evaluate motor competence through isolated low-level skills rather than instruction-conditioned tasks that also require visual grounding, sequential state awareness, and progress verification (Ye et al., 2025; Chen et al., 2024; Wang et al., 2025b). Consequently, existing evaluation paradigms remain incomplete in complementary ways: embodied-agent benchmarks (Yang et al., 2025; Ju et al., 2026; Wang et al., 2025a) often under-emphasize real-world dexterous execution, whereas dexterous manipulation benchmarks often lack the task structure needed to assess instruction-driven embodied behavior.

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.

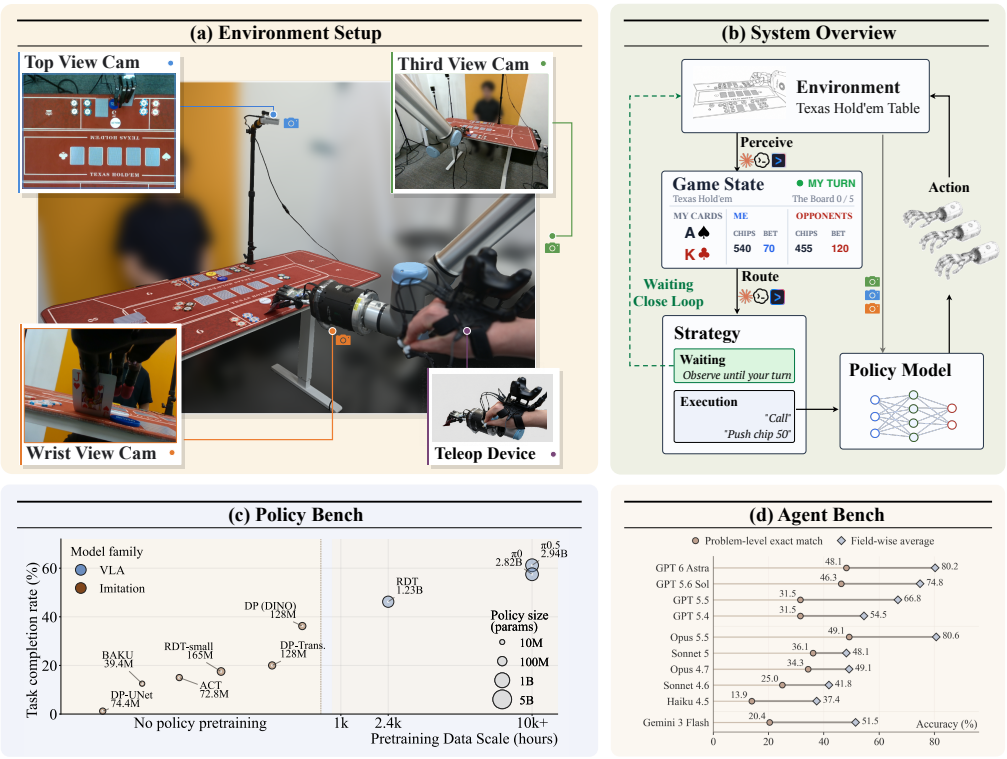


Figure 1: **Overview of DexHoldem**, a real-world Texas Hold'em benchmark for dexterous manipulation. *(a)* The policy uses top-down, third-person, and wrist-mounted views; a separate dedicated camera supplies the agent view. *(b)* The system closes the loop by parsing observations into game state, routing instructions, and executing policies. *(c,d)* Policy and agent benchmarks (the latter for a subset of perceivers) show that current models still struggle with contact-rich manipulation and fine-grained visual-state grounding.

To evaluate this coupled setting, we use Texas Hold'em tabletop interaction, where semantic grounding, state tracking, and dexterous control are jointly necessary. Cards and chips define structured targets: a policy must identify a specific card or chip denomination and manipulate it at a designated location. Thin cards (~ 0.3 mm) and chips require contact-rich control under friction and disturbance uncertainty. Each action changes the scene, allowing perception, routing, execution, and recovery errors to compound. The domain therefore provides a controlled test of instruction-conditioned dexterous manipulation rather than general poker intelligence.

Based on this rationale, we introduce **DexHoldem**, a real-world ShadowHand (Shadow Robot Company, 2025a) benchmark for Texas Hold'em tabletop manipulation. As summarized in Figure 1, it provides 1,470 demonstrations across 14 card and chip primitives, a shared physical policy evaluation, a contextual state-parsing benchmark, and closed-loop system evaluation. The policy benchmark measures execution conditioned on a fixed task vocabulary; it does not independently establish generalization to unseen language or disentangle visual grounding from learned spatial priors.

The physical evaluation separates completion of the requested primitive from preservation of the surrounding scene. Its four outcome categories distinguish successful execution with or without disruption from failed execution with or without disruption. The scoring rules in Section A.2 specify playable-region, unintended-overlap, and non-target-displacement criteria. This separation reveals failures that task completion alone would conceal, while the perception and system evaluations examine state parsing and the composition of primitives in the same environment. The shared interfaces and scoring protocols allow future policies and perceivers to be compared at the component level and then evaluated together in closed-loop interaction.

We evaluate nine policies on an 80-trial schedule each, 720 trials in total. The highest observed TCR is 61.2%, whereas the best SPSR is 47.5%. On 36 perception problems, the best exact match over applicable scored fields is 49.1% and the best field-wise average is 80.6%; community-card

accuracy peaks at 61.5%. Over 33 closed-loop rollouts pairing GPT 5.5 with π_0, only 12.1% of hands complete; retries restore the failed primitive in 12 of 34 dispatches. In three of the four completed hands, closed-loop retries resolve prolonged stalls that would otherwise require manual termination; only one hand completes with neither a retry nor a human-help request.

In summary, DexHoldem makes the following contributions:

1. We collect a real-world Texas Hold'em dexterous manipulation dataset with 1,470 real-world teleoperated demonstrations, covering 14 Texas Hold'em manipulation primitives.
2. We introduce a real-world dexterous hand policy benchmark that trains and evaluates policy models on these demonstrations under a shared multi-view observation–action interface and a scene-preservation-aware physical scoring rubric.
3. We introduce an agentic perception benchmark that evaluates whether embodied agents can visually parse structured tabletop game state for downstream decision routing.
4. We provide a system-level evaluation over 33 closed-loop hand-level rollouts, with three case studies, and an empirical analysis of RDT fine-tuning dynamics, exposing failure modes in scene-preserving execution, chip-state perception, and long-horizon reliability.

2 Related Work

Dexterous Robotic Manipulation. Dexterous manipulation studies how multi-fingered robot hands can perform contact-rich behaviors that are difficult for parallel grippers, including grasping, in-hand reorientation, articulated-object operation, and bimanual coordination. Early large-scale learning systems and real-robot platforms showed that dexterous control can be learned from demonstrations, reinforcement learning, or large offline datasets (Rajeswaran et al., 2017; Andrychowicz et al., 2020; Ahn et al., 2020; Fu et al., 2021). More broadly, general robot policy learning has shown that transformer-based and multi-task visuomotor policies can scale manipulation across language instructions and visual observations (Brohan et al., 2023; Zitkovich et al., 2023; Octo Model Team et al., 2024), while diffusion policies and data-generation systems provide expressive action distributions and scalable supervision for imitation learning (Chi et al., 2023; Jiang et al., 2023; Mandlekar et al., 2023; Jiang et al., 2025). Subsequent dexterous work has expanded the range of multi-finger behaviors, including bimanual hand control (Chen et al., 2022), articulated object manipulation (Bao et al., 2023), sim-to-real point-cloud policies (Qin et al., 2023), object reorientation (Chen et al., 2023), in-hand manipulation protocols (Cruciani et al., 2020), and challenging simulated manipulation tasks solved with trajectory optimization and reinforcement learning (Charlesworth & Montana, 2021). More recent work targets scalable dexterous grasp synthesis, dynamic handover, differentiable grasp generation, and large-scale dexterous demonstration data (Chen et al., 2024; Wang et al., 2025b; Turpin et al., 2023; Zhang et al., 2024a; Ye et al., 2025). These methods substantially advance robot policy learning and low-level dexterous skill learning, but they often evaluate motor competence in isolation rather than within a complete language-conditioned perception-decision-action loop.

Robot Manipulation Benchmarks and Dexterous Evaluation. Benchmark design has been central to progress in robot learning. RLBench and Meta-World provide diverse manipulation tasks for evaluating generalization, multi-task learning, and meta-learning (James et al., 2019; Yu et al., 2020), while CALVIN, LIBERO, VLABench, RoboCasa, RoboTwin, and RoboTwin 2.0 focus on language-conditioned long-horizon manipulation, lifelong transfer, household tasks, and bimanual coordination (Mees et al., 2022; Liu et al., 2023; Zhang et al., 2024b; Nasiriany et al., 2024; Mu et al., 2025; Chen et al., 2025). Simulation frameworks such as ManiSkill, ManiSkill2, and ManiSkill3 improve scalability and standardized evaluation for manipulation learning (Mu et al., 2021; Gu et al., 2023; Tao et al., 2025). Dexterous manipulation benchmarks and datasets, including Adroit, ROBEL, RoboHive, D4RL, Bi-DexHands, DexArt, BODex, DexH2R, DexGraspNet 2.0, and Dex1B, further provide important testbeds for multi-finger control, grasping, articulation, handover, and large-scale dexterous learning (Rajeswaran et al., 2017; Ahn et al., 2020; Kumar et al., 2023; Fu et al., 2021; Chen et al., 2022; Bao et al., 2023; Chen et al., 2024; Wang et al., 2025b; Zhang et al., 2024a; Ye et al., 2025). However, most dexterous benchmarks emphasize isolated motor skills, while many language-conditioned embodied benchmarks rely on simulation, simple grippers, or arm-centric manipulation. DexHoldem connects these lines with a ShadowHand setup (Shadow Robot Company, 2025a) for instruction-conditioned manipulation requiring semantic grounding, state tracking, and precise contact-rich execution.

Embodied Agents. Embodied agents use multimodal foundation models for perception, reasoning, and high-level action selection in simulated or real environments. PaLM-E studies embodied multimodal reasoning across heterogeneous observations and embodiments (Driess et al., 2023), while recent vision-language-action and flow-based models such as OpenVLA, π_0, and $\pi_{0.5}$ explore open-world generalization, continuous action generation, and cross-embodiment transfer (Kim et al., 2024; Black et al., 2024; Intelligence et al., 2025). Embodied-AI environments and instruction-following benchmarks such as AI2-THOR, Habitat, VirtualHome, ALFRED, and BEHAVIOR established evaluation settings for visual navigation, household interaction, and compositional language grounding (Kolve et al., 2017; Savva et al., 2019; Puig et al., 2018; Shridhar et al., 2020; Srivastava et al., 2021). Embodied agents extend this direction by grounding language in affordances, using feedback for closed-loop reasoning, generating executable policy code, or composing 3D value maps for manipulation (Ahn et al., 2022; Huang et al., 2022; Liang et al., 2023; Huang et al., 2023), and recent suites such as EmbodiedBench, MomaGraph, and ENACT evaluate multimodal perception, spatial understanding, dynamic state tracking, world modeling, and long-horizon planning (Yang et al., 2025; Ju et al., 2026; Wang et al., 2025a). Together, these works make it increasingly important to test whether embodied systems can close the loop from instruction understanding and visual reasoning to real-world dexterous execution. DexHoldem complements this literature by making the final action step physically demanding: the agent must not only identify what should be done, but also execute fine-grained multi-finger manipulation of thin cards and chips without disturbing the tabletop state.

3 DexHoldem System Design

DexHoldem is designed to evaluate dexterous manipulation policies and embodied agents in a human-robot Texas Hold'em tabletop setting. An overview of the system is shown in Figure 2. The system has two coupled layers: an embodied agent captures observations, maintains a structured game-state memory, and chooses the next activity stage, while a multi-task policy executes the corresponding primitive from visual observations, proprioceptive states, and a task condition. The loop supports waiting, perception, reasoning, action execution, re-execution after recoverable failures, and human intervention when the tabletop state cannot be safely continued. We provide details below on how we benchmark atomic policy tasks, agentic perception, and full-system evaluation.

3.1 Dexterous Hand Policy Bench

The policy benchmark isolates atomic dexterous execution from game-level decision making. It consists of a standardized suite of 14 language-instructed primitives on the Texas Hold'em tabletop, spanning card pickup, card placement, card revealing, and single-chip pushing or pulling across multiple chip denominations. For each primitive, DexHoldem provides 105 teleoperated demonstrations, yielding 1,470 demonstrations in total. We use a fixed split of 100 training trajectories and 5 validation trajectories per primitive, so every policy uses the same multi-task demonstration budget and is evaluated against the same primitive specification in Table 8.

All policies use a shared observation-action interface for the ShadowHand–UR platform. At each rollout step, a policy receives synchronized visual observations from top-down, third-person, and wrist-mounted cameras, the current arm and hand proprioceptive state, and a task condition specifying the requested primitive. It outputs a short-horizon sequence of joint-position targets in the shared 30-dimensional action space, with 6 dimensions for the arm and 24 for the dexterous hand. This interface makes the benchmark model-agnostic: task-trained and pretrained policies are compared without changing the task, state representation, or rollout protocol.

We score each rollout along two independent axes: primitive completion and terminal scene preservation. Pickup requires the selected card to remain held clear of the tabletop at its endpoint; put-down requires release and settled face-down placement at the requested side. A failed pickup automatically makes its paired put-down trial a failure, retained in the denominator. The resulting categories are **scene-preserving success** (SP: completed, preserved), **disruptive completion** (DC: completed, disrupted), **task failure** (TF: not completed, preserved), and **disruptive failure** (DF: not completed, disrupted). These are outcome categories, not an ordinal ranking. SPSR counts SP; TCR counts SP and DC. Disruption is assessed after settling using playable-region, unintended-overlap, and non-target-displacement rules in Section A.2; Figure 4 illustrates the distinction.

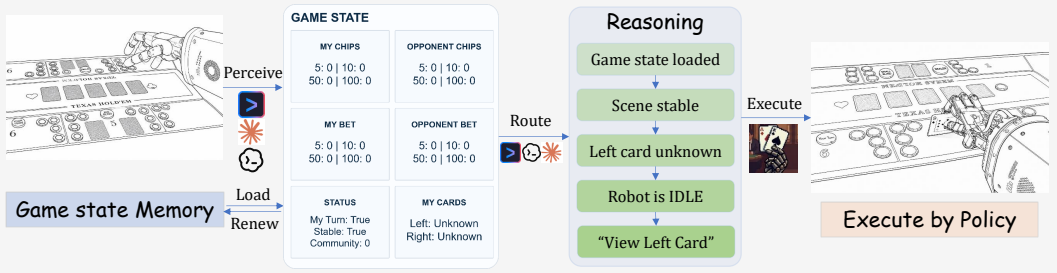


Figure 2: **One decision step of the DexHoldem embodied agent**. The agent perceives the tabletop, loads and renews structured game-state memory, routes the state through reasoning checks, and dispatches a dexterous policy when the scene is stable and an executable primitive is needed. In the illustrated step, an unknown left card with the robot idle routes to the agent primitive `view_card(L)`, which translates to the dexterous-policy sequence `pick_up_left` → perceive → `put_down_left`.

3.2 Agentic Perception Bench

DexHoldem also includes an agentic perception benchmark that isolates visual state parsing from downstream routing, poker-action selection, and physical execution. Each problem corresponds to one tabletop state sampled from a real game trajectory, presented with the available predecessor-state context: agent-view captures and pre-labeled structured states. Twelve problems, including all five turn-gate and five of the seven outcome-judge problems, are presented without predecessor context. The agent-view capture of the sampled state itself is the only frame the perceiver must parse from raw pixels. Following the system visual guidelines, the perceiver parses the current state into a structured game state decomposed into eight perception challenges, each scored as a separate evaluator column: **loop stage** (LS), **turn ownership** (TO), **blind information** (BI), **community cards** (CC), **current bet chips** (CB), **robot chip inventory** (RCI), **opponent chip inventory** (OCI), and **showdown outcome** (SO). Because the latter five challenges apply only to a subset of states—for example, SO is scored only on showdown problems and CC only when community cards are visible—we define overall success on a problem as exact match over the scored challenges applicable to that problem. This measures current-state parsing conditioned on annotated history, not autonomous tracking of a model's own past predictions.

Problems fall into six scene types that determine which challenges are scored: *turn gate*, *robot progress*, *held-card read*, and *recovery/safety* states are scored on LS, TO, and BI; *table decision* states add CB, RCI, OCI, and, when community cards are visible, CC; and *outcome judge* states require all eight. For a state in which the robot is executing a primitive, the scored question is thus the current loop stage rather than a re-read of community cards already recorded in the predecessor states; for a state in which both players have just revealed their hole cards, it is the showdown outcome. The problem inventory, ground-truth schema, prompt and harness specification, and deterministic evaluator are documented in Section A.3.

3.3 System-Level Evaluation

DexHoldem evaluates closed-loop embodied execution by composing the dexterous-policy and agentic-perception interfaces in real two-player Texas Hold'em tabletop rollouts. Each system-level instantiation pairs a pre-configured embodied agent with one dexterous-policy model from Section 3.1. At each loop step, the agent captures an agent-view image, parses it into the structured state defined in Section 3.2, routes the state through deterministic workflow gates, and dispatches a dexterous-policy primitive from Table 8 whenever physical motion is required. The main agent is not invoked at every captured state: the router handles waiting, verification, completion, continuation of pending multi-atom translations, and retryable recovery, while the main agent is queried only at decision states where multiple high-level agent primitives are legal.

We probe system-level trajectory quality with per-trajectory operational counters. As reported in Table 3, these counters are **captured states** (States); **dispatched agent primitives** (AP), every primitive chosen by the main agent, including `request_human`, `reset_to_init`, and `stop`, but excluding `wait` and router-level gates; **dispatched dexterous-policy primitives** (DPP); and

wait-branch events (WA), **human-help requests** (HL), and **recovery retries** (RC), counted when the router prepares a retry of a primitive that has remained stuck for a prolonged period without disrupting the scene. These quantities describe where the composed trace spends its steps: AP and DPP measure the length of the composed decision-execution trace, HL is the subset of AP corresponding to human-help escalation, WA captures repeated waiting for scene stability, robot progress, or turn changes, and RC records retries of harmless failures; disrupted scenes are routed to `request_human` instead. Section A.1 provides the loop-stage labels, agent-to-policy primitive translation, rollout protocol, verification and recovery logic, and termination criteria.

4 Experiments

4.1 Experimental Setup

We use the policy-bench protocol in Section 3.1 to test whether current visuomotor policies can execute the 14 atomic primitives in Table 8 under a shared demonstration split, robot interface, and physical scoring protocol. Each multi-task policy uses the fixed 100/5 train–validation split per primitive and maps three camera views and proprioception to 30-dimensional joint-position targets, executed in 32-step chunks at 10 Hz without action filtering. We reset the hand and task-relevant objects and randomize the tabletop configuration within the benchmark layout before each independently initialized trial; paired put-down trials continue from the preceding pickup. We score rollouts using the four outcome categories defined in Section 3.1 and report scene-preserving success rate (SPSR), which counts only scene-preserving successes, and task completion rate (TCR), which also counts disruptive completions. The detailed rollout randomization schedule and primitive-group breakdown are provided in Section A.2.

We compare two broad policy families under this interface. The first family contains pretrained robot policies and vision-language-action models adapted to DexHoldem, including $\pi_{0.5}$, π_0, and RDT (Intelligence et al., 2025; Black et al., 2024; Liu et al., 2025), all conditioned on natural-language task text. The second family contains task-specific imitation baselines trained on DexHoldem demonstrations, including diffusion-policy variants (Chi et al., 2023), ACT (Zhao et al., 2023), and BAKU (Haldar et al., 2024), which are conditioned on discrete instruction IDs. DP (DINO) uses a DINOv2 visual representation but is trained only as a task-specific policy rather than as a pretrained foundation policy model (Oquab et al., 2023), and DP-Transformer is trained from scratch as an instruction-ID-conditioned diffusion-policy baseline. RDT-small uses the RDT architecture family with a randomly initialized policy backbone. All policies follow the same physical trial protocol and scoring rule; model-native input encoders and action adapters differ (Table 7).

4.2 Policy Model Results

Table 1 summarizes aggregate physical evaluation for each policy over the 80-trial schedule per policy (720 trials in total), covering all 14 primitives in Table 8. Although individual primitives use different trial counts, the schedule forms four balanced 20-trial primitive groups; Table 11 reports the corresponding pickup, chip-push, chip-pull, and put-down/show breakdown. By task completion rate, $\pi_{0.5}$ obtains the highest observed aggregate result at 61.2%. By the stricter scene-preserving success rate, however, $\pi_{0.5}$ and π_0 tie at 47.5%; π_0 has a lower task completion rate because it produces fewer disruptive completions. All other policies trail these two models by a substantial margin, showing that DexHoldem remains difficult even when evaluation is restricted to atomic skill execution rather than full game-level routing. Figure 7 relates pretraining scale, policy size, and task completion rate.

Task groups reveal a more specific difficulty than aggregate ranking alone. Both π policies achieve 100% SPSR on pickup, but only 25% on chip push and 15% on chip pull. On put-down/show, their 80% TCR falls to 50% SPSR (Table 11). RDT reaches 30.0% overall SPSR, and DP (DINO) reaches 26.2%. These are observed differences under the evaluated configurations, not isolated effects of architecture or pretraining. In particular, the two π policies tie at 38/80 SP outcomes and differ by only three completions (49/80 versus 46/80). The aggregate is trial-weighted: each of the four 20-trial groups contributes equally.

The SPSR–TCR gap identifies completed primitives whose terminal scenes violate the preservation rules: 11 trials for $\pi_{0.5}$ and 13 for RDT. Thus, task completion alone hides a substantial number of

Table 1: Aggregate policy-model results over the 80-trial real-world evaluation schedule per policy; put-down trials whose preceding pickup failed count as failures (Section A.2). Params reports policy-only parameter count, excluding visual encoders. Trial outcomes are abbreviated as SP (scene-preserving success), DC (disruptive completion), TF (task failure), and DF (disruptive failure). SPSR counts SP; TCR counts SP and DC.

		Trial outcomes					Rates	
Policy	**Params**	**SP**	**DC**	**TF**	**DF**	N	**SPSR**	**TCR**
$\pi_{0.5}$	2.94B	38	11	31	0	80	47.5%	61.2%
π_0	2.82B	38	8	33	1	80	47.5%	57.5%
RDT	1.23B	24	13	40	3	80	30.0%	46.2%
DP (DINO)	128M	21	8	48	3	80	26.2%	36.2%
DP-Transformer	128M	11	5	46	18	80	13.8%	20.0%
RDT-small	165M	11	3	59	7	80	13.8%	17.5%
ACT	72.8M	8	4	67	1	80	10.0%	15.0%
BAKU	39.4M	5	5	67	3	80	6.2%	12.5%
DP-UNet	74.4M	1	0	79	0	80	1.2%	1.2%

disruptive outcomes. These labels characterize the terminal scene; beyond the paired put-down trials, they do not estimate the success probability of a subsequent task.

4.3 RDT Fine-Tuning Data Scaling Study

We use RDT as a representative policy instantiation to probe how much DexHoldem-specific dexterous-hand data is needed to reliably fit the target action distribution, using held-out action-prediction loss as an offline diagnostic. Architecture, objective, validation split, and evaluation protocol are fixed; only initialization and data amount vary. We compare random initialization against a gripper-pretrained RDT checkpoint (Liu et al., 2025) using paired random seeds. The 10%, 20%, 50%, and 100% data ratios correspond to 10, 20, 50, and 100 training trajectories per primitive, sampled from each primitive's 100-trajectory training split. All ratios use the same five held-out validation trajectories per primitive. We report the full train-time validation-loss curves in Figure 8 and summarize final validation loss in Figure 3.

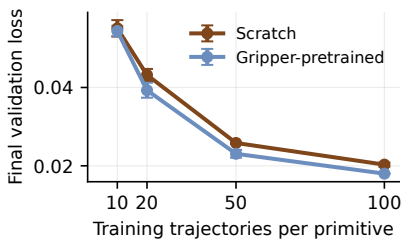


Figure 3: Final validation loss for the RDT fine-tuning data-scaling probe. Random and pretrained initializations follow similar data-scaling trends. Error bars denote one standard deviation over three completed paired seeds.

Under this held-out action-prediction-loss criterion, pretrained initialization reduces final loss by 1.2%, 9.0%, 10.7%, and 11.3% at the four increasing data fractions. The two initializations follow similar trends, with a small difference at 10% data and larger offsets at higher fractions. This is an offline diagnostic of the evaluated RDT configuration: it does not establish physical few-shot transfer, scene preservation, or a general reduction in dexterous demonstration requirements. "Random initialization" refers to the policy initialization comparison, not a claim that all visual and language representations lack pretraining.

4.4 Benchmarking Perception Modules of Agents

We evaluate each perceiver in the isolated agentic-perception setting defined in Section 3.2. Each problem supplies the current image, permitted predecessor context, and field-specific visual guidelines. The prompt requires visual evidence and a structured summary, and prohibits robot actions, helper scripts, and poker-action selection. GPT, Claude, and Gemini models run through Codex, Claude Code, and Gemini CLI, respectively. This compares model–harness configurations; reasoning labels do not equalize compute or tool budgets. The GPT configurations use `medium` reasoning and the `fast` service tier. The Gemini runner supplies no reasoning override. Section A.3 gives model identifiers, the full prompt, tool and budget settings, and configuration provenance.

Table 2: **Per-perceiver accuracy on perception bench.** Each row is the average of three evaluation runs. **Overall** is strict problem-level exact match: a problem counts as correct iff every applicable scored field is correct. **LS** (loop stage), **TO** (turn ownership), **BI** (blind info), **CC** (community cards), **CB** (current bet chips), **RCI** (robot chip inventory), **OCI** (opponent chip inventory), and **SO** (showdown outcome) are field-wise accuracies on their applicable problem subsets, defined in Section A.3 and tabulated in Table 13. **Avg** is the unweighted mean of these eight columns, excluding Overall.

Harness	Perceiver	Overall	LS	TO	BI	CC	CB	RCI	OCI	SO	Avg
	GPT 6 Astra	48.1	69.4	**97.2**	**100.0**	20.5	**93.8**	**93.8**	**100.0**	66.7	80.2
	GPT 6 Sol	46.3	66.7	96.3	98.1	43.6	79.2	91.7	83.3	76.2	79.4
	GPT 5.6 Sol	46.3	**78.7**	95.4	**100.0**	38.5	68.8	75.0	56.2	**85.7**	74.8
	GPT 5.6 Luna	27.8	62.0	79.6	97.2	43.6	41.7	47.9	50.0	57.1	59.9
	GPT 5.5	31.5	72.2	80.6	**100.0**	**61.5**	45.8	62.5	35.4	76.2	66.8
	GPT 5.4	31.5	65.7	93.5	**100.0**	23.1	31.2	56.2	18.8	47.6	54.5
	GPT 5.4 mini	25.9	56.5	94.4	99.1	33.3	14.6	29.2	18.8	47.6	49.2
	Opus 5.5	**49.1**	76.9	**97.2**	**100.0**	20.5	81.2	91.7	91.7	**85.7**	**80.6**
	Sonnet 5	36.1	61.1	94.4	**100.0**	5.1	31.2	33.3	35.4	23.8	48.1
	Opus 4.7	34.3	43.5	93.5	**100.0**	43.6	31.2	37.5	43.8	0.0	49.1
	Sonnet 4.6	25.0	46.3	88.0	**100.0**	23.1	10.4	29.2	22.9	14.3	41.8
	Haiku 4.5	13.9	47.2	68.5	91.7	35.9	12.5	25.0	18.8	0.0	37.4
	Gemini 3 Flash	20.4	63.9	77.8	**100.0**	28.2	18.8	29.2	22.9	71.4	51.5
	Gemini 3.1 Flash L.	10.2	27.8	73.1	94.4	28.2	12.5	22.9	14.6	0.0	34.2

Table 2 reports the resulting semantic perception accuracy over the 36-problem benchmark inventory in Section A.3. Overall is strict problem-level exact-match accuracy: a problem is counted as correct only when every applicable scored field is correct. The remaining columns report field-wise accuracy on their applicable subsets, and Avg is the unweighted mean over the eight sub-capability columns, excluding Overall. Neither measures every runtime state field or downstream routing correctness.

Opus 5.5 in the Claude Code harness achieves both the best strict Overall (49.1%) and the best Avg (80.6%), narrowly ahead of GPT 6 Astra in the Codex harness (by one of 108 attempts and 0.4 points). Even the best Overall leaves over half of the attempts incorrect: one wrong field fails a problem, and table-decision and outcome-judge states require many fields at once.

The easiest fields are those tied to explicit tabletop markers. Blind information (BI) is near-saturated, with nine of fourteen perceivers reaching 100.0%, and turn ownership (TO) reaches 97.2% for GPT 6 Astra and Opus 5.5. Chip-state accuracy varies substantially across configurations: GPT 6 Astra reaches 93.8% on current bet chips (CB) and 100.0% on opponent chip inventory (OCI), both requiring exact denomination-level chip dictionaries. Community-card recovery remains difficult: CC peaks at 61.5% for GPT 5.5, while the two strongest aggregate configurations, Opus 5.5 and GPT 6 Astra, score only 20.5%, showing that strong aggregates can still conceal a weak routing-relevant field.

A no-vision predecessor-copy baseline achieves 8/36 Overall (22.2%), by copying the most recent provided structured state and counting the 12 problems without history as failures. Table 14 reports the breakdown by problem type. This diagnostic shows that history reuse alone can satisfy some scored targets; it does not quantify the value of history to each evaluated model. The three model runs reuse the same 36 problems and are not 108 independent scenes. Held-card identity, scene stability, and uncertainty flags are not directly scored by the reported columns (Section A.3).

4.5 System-Level Evaluation

We instantiate the system-level protocol of Section 3.3 with the Codex harness, using GPT 5.5 as both the perceiver and the main agent and π_0 as the dexterous policy, over 33 hand-level rollouts (Table 3); five further rollouts invalidated by operator error during setup are excluded. Each rollout plays one heads-up hand in closed loop until the hand completes and the agent stops, or the experiment conductor terminates it once the robot can no longer make progress (unrecognized disruption, indefinite waiting,

Table 3: **System-level evaluation** (Codex GPT 5.5 agent, π_0 policy): three case-study rollouts and the aggregate over 33 hand-level rollouts under the same protocol; (ii) and (iii) belong to the 33, whereas (i) is an earlier rollout. Counters follow Section 3.3; RC counts retries prepared by the runtime, of which 34 were dispatched and 12 verified complete before any further retry, escalation, or termination. A completion is assisted if its rollout contains a `request_human` dispatch.

Rollout	States	AP	DPP	WA	HL	RC	Outcome
(i)	22	8	7	7	2	1	terminated during `call`
(ii)	54	13	22	26	0	1	won at showdown
(iii)	23	8	10	7	0	1	lost at showdown
Mean over 33 rollouts	18.1	4.6	5.1	9.4	0.5	1.1	completion 4/33 (12.1%)
Median [min, max]	12 [2, 54]	3 [1, 18]	4 [1, 22]	5 [0, 34]	0 [0, 4]	1 [0, 4]	retries verified 12/34 (35.3%)
Completed hands	neither retry nor help 1 · retry only 2 · retry and help 1 · no `request_human` 3/33 (9.1%)						

or a scene that cannot be restored after a human-help request). A `request_human` dispatch pauses the loop for conductor intervention (Section A.1).

The evaluated system rarely sustains a full hand. Only 4 of 33 rollouts (12.1%) play a hand to its outcome (two won, two lost). A typical rollout spans 18.1 captured states (median 12) but dispatches only 4.6 agent primitives and 5.1 dexterous-policy primitives before it completes or is terminated: about five physical actions per hand. Wait-branch events account for 51.8% of all captured states, and human help is requested in 13 rollouts.

Closed-loop retries restore progress after prolonged stalls. A retry is triggered when an atomic primitive remains stuck for a prolonged period with the scene intact (`to_recover`); disrupted scenes route to `request_human`. At such stalled execution points, waiting alone does not restore progress, so without a retry the conductor would have to terminate the rollout. Across the 33 rollouts the runtime prepared 36 retries and dispatched 34: 12 completed the retried primitive, 8 were retried again, 7 escalated to human help, and 7 were cut before verification. Retries concentrate on the hole-card cycle (12 pickups, 15 put-downs). In three of the four completed hands, retries resolved such stalls in a card pickup for the showdown reveal and two chip pushes during bets, enabling the hands to continue to completion. Only one hand completes with neither (Table 3). Beyond completed hands, 28.5% of all captured states and 35 of 167 dispatches occur after a rollout's first verified recovery, and 9 of the 10 rollouts with one advance at least one hand stage (hole cards read, pre-flop bet, community-card street, showdown, outcome) afterwards. Human help also lets hands continue: 13 of 18 requests were resolved and the hand continued, and 11 of the 18 report a hole card returned face-up after put-down, which the endpoint criterion in Section A.2 counts as an incomplete put-down. Trajectory (iii) (panels in Section B.3) shows a full hand played to showdown with one verified recovery.

Takeaway. The evaluated system struggles to sustain interaction despite moderate isolated policy (Table 1) and perception (Table 2) scores. Retry traces illustrate recovery from prolonged stalls. Controlled comparisons are needed to quantify recovery gains and separate component effects. The shared interfaces provide a basis for varying the perceiver, policy, or recovery mechanism while holding other components fixed and measuring hand completion and human-help requests.

5 Conclusion

We introduced DexHoldem, a real-world benchmark combining dexterous policy evaluation, agentic perception, and closed-loop interaction in an environment for a Texas Hold'em game. Our evaluation results expose three complementary challenges: primitive completion can disrupt the scene, strong field-wise perception can coexist with incorrect joint state parsing, and the evaluated system completes only 4 of 33 hands. Retries resolve prolonged stalls in three completed hands, illustrating the role of recovery in sustaining interaction. The shared tasks, interfaces, and scoring protocols let future studies test whether better policies and perceivers improve hand completion and reduce human-help requests. The present study covers one ShadowHand–UR platform and one system-level agent–policy pairing. Within this setting, DexHoldem supports continued research on scene-preserving manipulation, structured state perception, and recovery from execution failures.

AI USE STATEMENT

The benchmark evaluates large language and multimodal models as components of the embodied-agent system. The GPT, Claude, and Gemini model configurations, agent interfaces, and evaluation protocols are described in Sections A.3, 3.2 and 4.4. OpenAI Codex was also used to assist with conversion to the ICLR submission format, manuscript revision, figure-export font settings, and checks of LaTeX compilation and layout.

REFERENCES

Michael Ahn, Henry Zhu, Kristian Hartikainen, Hugo Ponte, Abhishek Gupta, Sergey Levine, and Vikash Kumar. Robel: Robotics benchmarks for learning with low-cost robots. In *Conference on robot learning*, pp. 1300–1313. PMLR, 2020.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Chen Bao, Helin Xu, Yuzhe Qin, and Xiaolong Wang. Dexart: Benchmarking generalizable dexterous manipulation with articulated objects, 2023. URL https://arxiv.org/abs/2305.05706.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. URL https://arxiv.org/abs/2410.24164.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale, 2023.

Henry Charlesworth and Giovanni Montana. Solving challenging dexterous manipulation tasks with trajectory optimisation and reinforcement learning, 2021. URL https://arxiv.org/abs/2009.05104.

Jiayi Chen, Yubin Ke, and He Wang. Bodex: Scalable and efficient robotic dexterous grasp synthesis using bilevel optimization. *arXiv preprint arXiv:2412.16490*, 2024.

Tao Chen, Megha Tippur, Siyang Wu, Vikash Kumar, Edward Adelson, and Pulkit Agrawal. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8(84): eadc9244, 2023.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation, 2025. URL https://arxiv.org/abs/2506.18088.

Yuanpei Chen, Yaodong Yang, Tianhao Wu, Shengjie Wang, Xidong Feng, Jiechuan Jiang, Zongqing Lu, Stephen Marcus McAleer, Hao Dong, and Song-Chun Zhu. Towards human-level bimanual dexterous manipulation with reinforcement learning. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=D29JbExncTP.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

Silvia Cruciani, Balakumar Sundaralingam, Kaiyu Hang, Vikash Kumar, Tucker Hermans, and Danica Kragic. Benchmarking in-hand manipulation. *IEEE Robotics and Automation Letters*, 5(2):588–595, April 2020. ISSN 2377-3774. doi: 10.1109/lra.2020.2964160. URL http://dx.doi.org/10.1109/LRA.2020.2964160.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model, 2023.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021. URL https://arxiv.org/abs/2004.07219.

Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. Maniskill2: A unified benchmark for generalizable manipulation skills, 2023. URL https://arxiv.org/abs/2302.04659.

Siddhant Haldar, Zhuoran Peng, and Lerrel Pinto. Baku: An efficient transformer for multi-task policy learning, 2024. URL https://arxiv.org/abs/2406.07539.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022.

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. *arXiv preprint arXiv:2307.05973*, 2023.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment, 2019. URL https://arxiv.org/abs/1909.12271.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. In *Fortieth International Conference on Machine Learning*, 2023.

Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Jim Fan, and Yuke Zhu. Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16923–16930. IEEE, 2025.

Yuanchen Ju, Yongyuan Liang, Yen-Jen Wang, Nandiraju Gireesh, Yuanliang Ju, Seungjae Lee, Qiao Gu, Elvis Hsieh, Furong Huang, and Koushil Sreenath. Momagraph: State-aware unified scene graphs with vision-language model for embodied task planning. *International Conference on Learning Representations (ICLR) Oral*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

Vikash Kumar, Rutav Shah, Gaoyue Zhou, Vincent Moens, Vittorio Caggiano, Jay Vakil, Abhishek Gupta, and Aravind Rajeswaran. Robohive – a unified framework for robot learning. In *NeurIPS: Conference on Neural Information Processing Systems*, 2023. URL https://sites.google.com/view/robohive.

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, 2023. URL https://arxiv.org/abs/2209.07753.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023. URL https://arxiv.org/abs/2306.03310.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation, 2025. URL https://arxiv.org/abs/2410.07864.

Yumeng Liu, Yaxun Yang, Youzhuo Wang, Xiaofei Wu, Jiamin Wang, Yichen Yao, Sören Schwertfeger, Sibei Yang, Wenping Wang, Jingyi Yu, and Yuexin Ma. Realdex: Towards human-like grasping for robotic dexterous hand. *arXiv preprint arXiv:2402.13853*, 2024. URL https://arxiv.org/abs/2402.13853.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. *arXiv preprint arXiv:2310.17596*, 2023.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks, 2022. URL https://arxiv.org/abs/2112.03227.

Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations, 2021.

Yao Mu, Tianxing Chen, Shijia Peng, Zanxin Chen, Zeyu Gao, Yude Zou, Lunkai Lin, Zhiqiang Xie, and Ping Luo. Robotwin: Dual-arm robot benchmark with generative digital twins (early version), 2025. URL https://arxiv.org/abs/2409.02920.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots, 2024. URL https://arxiv.org/abs/2406.02523.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8494–8502, 2018.

Yuzhe Qin, Binghao Huang, Zhao-Heng Yin, Hao Su, and Xiaolong Wang. Dexpoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation. In *Conference on Robot Learning*, pp. 594–605. PMLR, 2023.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9339–9347, 2019.

Shadow Robot Company. *Shadow Dexterous Hand - Technical Specification*. Shadow Robot Company, 2025a. URL https://shadowrobot.com/wp-content/uploads/2025/09/shadow_dexterous_hand_e_technical_specification.pdf.

Shadow Robot Company. Shadow teleoperation system: Technical specification, September 2025b. URL https://shadowrobot.com/wp-content/uploads/2025/09/shadow_teleop_technical_specification.pdf. Technical specification.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10740–10749, 2020.

Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, C. Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments, 2021.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Viswesh Nagaswamy Rajesh, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai, 2025. URL https://arxiv.org/abs/2410.00425.

Dylan Turpin, Tao Zhong, Shutong Zhang, Guanglei Zhu, Jingzhou Liu, Ritvik Singh, Eric Heiden, Miles Macklin, Stavros Tsogkas, Sven Dickinson, and Animesh Garg. Fast-grasp'd: Dexterous multi-finger grasp generation through differentiable simulation, 2023. URL https://arxiv.org/abs/2306.08132.

Qineng Wang, Wenlong Huang, Yu Zhou, Hang Yin, Tianwei Bao, Jianwen Lyu, Weiyu Liu, Ruohan Zhang, Jiajun Wu, Fei-Fei Li, and Manling Li. Enact: Evaluating embodied cognition with world modeling of egocentric interaction. *arXiv preprint arXiv:2511.20937*, 2025a.

Youzhuo Wang, Jiayi Ye, Chuyang Xiao, Yiming Zhong, Heng Tao, Hang Yu, Yumeng Liu, Jingyi Yu, and Yuexin Ma. Dexh2r: A benchmark for dynamic dexterous grasping in human-to-robot handover, 2025b. URL https://arxiv.org/abs/2506.23152.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*, 2025.

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, 2025. URL https://arxiv.org/abs/2506.17198.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pp. 1094–1100. PMLR, 2020.

Jialiang Zhang, Haoran Liu, Danshi Li, Xinqiang Yu, Haoran Geng, Yufei Ding, Jiayi Chen, and He Wang. Dexgraspnet 2.0: Learning generative dexterous grasping in large-scale synthetic cluttered scenes, 2024a. URL https://arxiv.org/abs/2410.23004.

Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, and Xipeng Qiu. Vlabench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks, 2024b. URL https://arxiv.org/abs/2412.18194.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183. PMLR, 2023.

A BENCHMARK DOCUMENTATION

Dataset and Code Availability. Public dataset and code links are omitted for double-blind review. The data-collection setup and dataset contents are summarized in Section A.2.

Reproduction Workflow. The benchmark package includes instructions for environment setup, workspace bootstrapping, and launch. The bootstrapper writes a per-file SHA-256 manifest of the installed benchmark components, and each evaluation run records its harness version and command provenance. These records identify the installed files and invocation used for a run. The perception package includes the problems, ground-truth annotations, prompts, and deterministic scorer for offline evaluation.

DexHoldem specifies tasks at two levels. The primitive level defines the callable dexterous skills used for data collection, policy training, and physical rollouts. The agent level defines the perception, routing, verification, and recovery problems that arise when these primitives are composed into a Texas Hold'em tabletop interaction. This separation keeps the benchmark explicit about which results measure low-level manipulation and which results measure closed-loop embodied-agent behavior.

A.1 EMBODIED SYSTEM DESIGN DETAILS

Agent Design and Tasks. The DexHoldem embodied agent runs the closed-loop *capture→perceive→route→execute* workflow illustrated in Figure 2, composing the dexterous-policy primitives in Table 8 into hand-level Texas Hold'em interactions. Each loop iteration begins with a single agent-view capture from a dedicated tabletop camera, distinct from the three policy-side cameras in Section 3.1, parses the captured image into the structured game-state memory defined in Section 3.2 with the eight perception challenges *loop stage*, *turn ownership*, *blind information*, *community cards*, *current bet chips*, *robot chip inventory*, *opponent chip inventory*, and *showdown outcome*, routes the parsed state through gating logic that decides among waiting, perception repair, primitive verification, recovery, and selection of the next high-level decision, and dispatches and verifies a dexterous primitive whenever the chosen route requires physical motion. The `loop_stage` field takes one of eight values that summarize whether the robot is in motion (`acting`), settled between atoms of a multi-step sequence (`atom_idle`), ready for a new poker decision (`idle`), at a showdown that requires revealing the robot's cards (`show_hand`), in a settled showdown outcome (`win`, `lose`), eligible to retry a harmless failure (`to_recover`), or unsafe to continue without human intervention (`down`); these stages drive the routing branches in Section 3.3.

The agent acts through a fixed set of 13 *agent primitives*—the high-level actions available to the main agent at legal decision states—which together cover the routing branches in Section 3.3. Each agent primitive is translated either into a sequence of dexterous-policy primitives from Table 8 or into a non-robot operation such as an audio cue, a state-machine transition, or a request for human help. Because the dispatch branch executes one dexterous-policy primitive at a time, the recovery branch retries individual failed primitives rather than the entire agent primitive. Table 4 enumerates the agent primitives and their translations; for chip-betting primitives, the agent splits the target chip count using a min-count rule that prefers larger denominations and dispatches one push or pull primitive per chip in $100 \rightarrow 50 \rightarrow 10 \rightarrow 5$ order. Within each call, the single target chip is selected according to Section A.2. Section A.3 specifies the released perception interface and evaluator used to measure the parsing component of this loop in isolation.

Pickup-to-Put-Down Transition. Within `view_card`, physical pickup completion and held-card recognition are separate checks. The router waits while the robot is acting or the scene is unstable. Once the robot has settled in `atom_idle`, the dispatched pickup is visually verified before the sequence advances to reading the held card. Put-down is dispatched only after the card's rank and suit have been recorded in the memory slot corresponding to the selected left or right card. A missing card read routes to recognition or human help before put-down. A settled missed grasp with an intact scene is eligible for recovery; a dropped or misplaced card requires human help. After put-down, the released card's face-down placement is visually verified before the sequence is marked complete. These runtime checks compose the physical endpoint criteria in Section A.2; reading the card is not part of the standalone pickup completion criterion.

Table 4: Agent-primitive to dexterous-policy-primitive mapping. Names and numeric IDs refer to the dexterous-policy primitives defined in Table 8 (0–1 pickup, 2–5 push, 6–9 pull, 10–13 put-down/show). Control and audio primitives do not dispatch any dexterous-policy primitive. For chip-betting primitives, Δ denotes the chip target inferred from the parsed table state, and the agent dispatches one push or pull primitive per chip in $100 \rightarrow 50 \rightarrow 10 \rightarrow 5$ order so that a single failed primitive can be retried in isolation. Dotted rules box the argument-variant groups of `view_card`, `show_card`, and `put_down_card`.

Agent primitive	Dexterous-policy primitive sequence	Type
`wait`	state-machine sleep	
`fold`	recognized by scene	Control
`stop`	terminate route	
`reset_to_init`	reset hand to home pose	Reset
`view_card`	(L) `pick_up_left` (0) $\rightarrow$ perceive $\rightarrow$ `put_down_left` (10) (R) `pick_up_right` (1) $\rightarrow$ perceive $\rightarrow$ `put_down_right` (11)	View
`show_card`	(L) `pick_up_left` (0) $\rightarrow$ `show_left` (12) (R) `pick_up_right` (1) $\rightarrow$ `show_right` (13)	Show
`put_down_card`	(L, down) `put_down_left` (10) (R, down) `put_down_right` (11) (L, up) `show_left` (12) (R, up) `show_right` (13)	Put-down
`check`	audio cue "Check"	Audio
`call`	push primitives 5/4/3/2 over Δ = opponent_bet $-$ my_bet	
`raise` (amount A)	push primitives 5/4/3/2 over $\Delta = A-$my_bet	Chip push
`all_in`	push primitives 5/4/3/2 over the full robot-side chip stack	
`collect_winnings`	pull primitives 9/8/7/6 across both bet zones	Chip pull
`request_human` (reason)	audio cue, then `loop_stage=down`	Help

Agent Sandbox. At runtime, the agent operates inside a small sandbox that bundles the workflow document, the perception guidelines, and a fixed set of deterministic Python helpers that the main agent invokes between reasoning steps. The workflow document defines the loop, the agent action space, and per-state routing rules; the perception guidelines define how each parsed-state field is read from the captured image; and the helpers handle capture, state-folder management, rule-based routing, agent-to-policy primitive translation, robot dispatch, and audio or remote-control side effects. The router is rule-based and encodes the hard workflow constraints that do not require agent reasoning—for example, the first agent primitive of a fresh game is always routed to `view_card`, and once a chip-bet sequence such as `raise` has been pre-translated, the router advances directly to the next pending robot atom in that sequence without re-prompting the main agent. The main agent is therefore only invoked in states where multiple branches are legal, such as the `idle` loop stage where a new poker action must be selected. Table 5 enumerates the contents of this sandbox.

Policy Network Interface. All policy implementations use the same benchmark-facing robot interface. The physical platform consists of a Universal Robots UR10e arm with 6 controllable joints, a Shadow Dexterous Hand with 24 controllable joints, and three Intel RealSense D435i RGB-D cameras covering top-down, third-person, and wrist-mounted views. The action and proprioceptive state are represented in a shared 30-dimensional joint-position space, with the first 6 dimensions assigned to the arm and the remaining 24 dimensions assigned to the hand. Each low-level policy is conditioned on one of the 14 task instructions in Table 8.

Data and Observation Pipeline. Raw demonstrations are stored as numbered trajectory files under the 14 primitive folders. Each trajectory records synchronized multi-view RGB-D observations and robot joint measurements. The data organizer maps each primitive folder to its instruction ID, constructs 100 training trajectories and 5 validation trajectories per primitive, and converts dict-valued joint records into the canonical 30-dimensional joint order. The task-specific loaders use per-episode `.npy` arrays containing RGB/depth observations, optional precomputed RGB features, joint-position

Table 5: Sandbox setup. The agent runtime bundles a workflow document, ten perception guidelines, and a set of deterministic Python helpers that the main agent invokes between reasoning steps.

Component	Role	Class
`SKILL.md`	Workflow document: loop, action space, and routing rules	Doc
`visual_guidelines/`	Ten Markdown modules used during the *perceive* stage	Doc
`preflight.py`	Backend validation and experiment-folder initialization	Setup
`capture.py`	Single-frame capture from the agent camera	Perception I/O
`state.py`	State-folder manager and parsed-state writer	State
`router.py`	Rule-based per-state router; emits the next gate as JSON	Routing
`action_translator.py`	Translates an agent primitive into a robot-primitive sequence	Translation
`executor.py`	Dispatches robot commands and records execution progress	Execution
`text_to_sound.py`	Plays audio cues for non-robot primitives	Audio
`remote_exec.py`	Sends commands to the dexterous-hand control terminal	Control
`utils.py`	Shared config, file I/O, and state helpers	Helpers

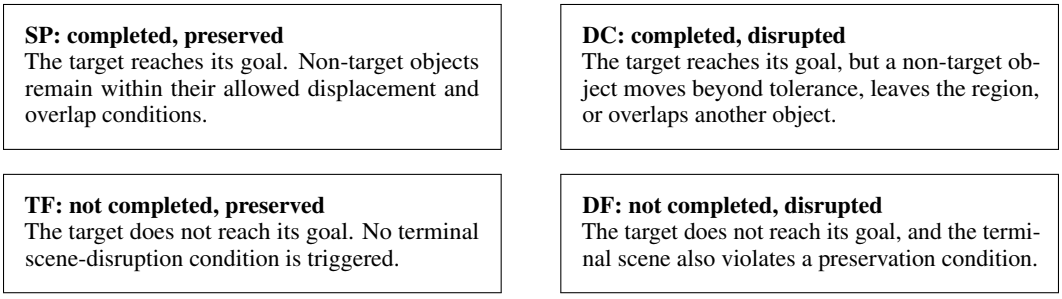


Figure 4: Two independent scoring axes produce four outcome categories. Panels are schematic adjudication examples; exact disturbance rules are given in Section A.2.

proprioception, the instruction ID, and action targets. For π_0 and $\pi_{0.5}$, the same demonstration split is converted to LeRobot format with RGB observations, joint positions, task text, and absolute joint-position action targets. Proprioception and actions use model-specific normalization with training-set statistics saved alongside the checkpoint and reused during deployment. Unless otherwise specified, policies use an observation horizon of 1 and predict a 64-step action sequence; both π_0 and $\pi_{0.5}$ instead predict 50 steps from a single observation. The prediction horizon is distinct from the executed action chunk: all evaluated policies execute 32 steps per query.

Training and Checkpoint Selection. For the physical policy comparison, models with randomly initialized policy backbones are trained for 300 epochs at a learning rate of 10^{-5}, and RDT is fine-tuned for 20 epochs at the same learning rate. Their mini-batch sizes and gradient accumulation settings are listed in Table 7; the OpenPI input and optimization settings for π_0 and $\pi_{0.5}$ are reported separately in Table 6. For each model, we select the checkpoint with the lowest loss on the fixed validation split for real-robot evaluation.

Observation and Instruction Encoding. The task-trained policies share an observation-encoder interface that can process raw RGB, depth, precomputed visual features, and proprioceptive vectors. Lightweight baselines use trainable ResNet encoders, while larger policies use frozen visual backbones with offline feature precomputation: DinoV2 CLS features for standard diffusion-policy variants and SigLIP-SO400M patch tokens for RDT implementations. Instructions are represented either as discrete task IDs passed through a learned embedding/projection, as cached text embeddings, or as model-specific language tokens, depending on the policy family.

DP (DINO) and DP-Transformer. DP (DINO) is the high-capacity diffusion-policy baseline for the shared DexHoldem interface. It uses frozen DinoV2 visual features as observation context and a Transformer denoiser to predict 30-dimensional joint-position chunks. DP-Transformer uses the same instruction-ID-conditioned diffusion-policy objective and 30-dimensional action interface, but

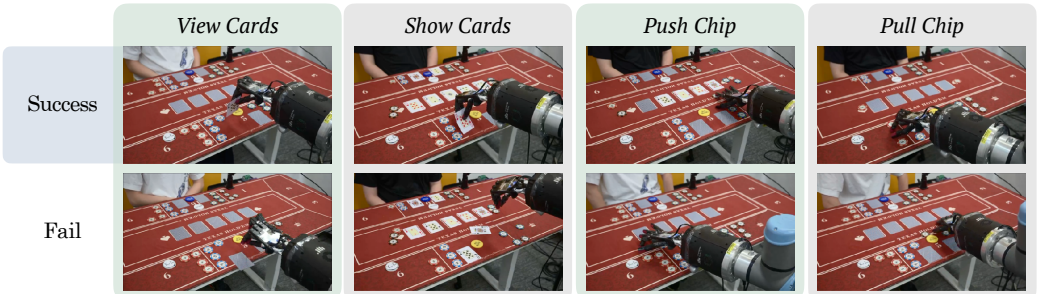


Figure 5: Representative physical policy executions, with task success/failure labels. Scene preservation is a separate scoring axis, as illustrated in Figure 4.

is trained from scratch as the task-policy Transformer baseline reported in the main comparison. During deployment, both variants use the checkpoint's normalization statistics and EMA weights to produce executable joint targets.

DP-UNet. DP-UNet keeps the same diffusion-policy action interface but replaces the frozen-feature Transformer configuration with trainable ResNet visual encoders and a 1D UNet denoiser. This variant provides a lighter task-trained baseline whose inputs, action horizon, normalization, and instruction conditioning remain compatible with the other low-level policies.

ACT. ACT implements a CVAE Transformer over the benchmark observation and action sequence. At training time, the encoder receives observation tokens and ground-truth action chunks to infer a latent action variable; at inference time, the policy uses the prior mean and decodes a deterministic 30-dimensional action chunk conditioned on the current observation and instruction.

BAKU. BAKU is adapted as a deterministic action-token policy under the same canonical batch format. The model builds observation tokens from the available visual and proprioceptive streams, appends learned action tokens, applies a causal Transformer, and decodes the predicted action-token states into a full joint-position chunk for the robot.

RDT. RDT adapts an RDT-1B-style robotics diffusion Transformer from a gripper-centric interface to the ShadowHand–UR 30-dimensional joint space. The implementation retains SigLIP visual patch tokens, cached T5 instruction tokens, and alternating language/image conditioning, while replacing the state and action representation with the benchmark joint-position interface. Training predicts clean action samples under DDPM, and deployment uses fast DPMSolver inference.

RDT-small. RDT-small is the reduced-capacity RDT variant reported in the main policy comparison. It uses the same DexHoldem observation adapter, T5 instruction-token interface, 30-dimensional state-action representation, diffusion training objective, and DPMSolver deployment path as RDT, but replaces the full RDT-1B-style backbone with the smaller configuration used in our implementation. Its policy backbone does not load pretrained policy weights and is trained from random initialization on DexHoldem demonstrations. This distinction concerns policy pretraining; it does not imply that the upstream visual or text representations are unpretrained.

π_0. We fine-tune the official `pi0_base` checkpoint using the OpenPI JAX training pipeline and a DexHoldem data adapter. The adapter maps camera RGB streams into OpenPI image fields, passes the task prompt as language context, and adapts model outputs to the 30-dimensional robot command space. Our π_0 configuration predicts absolute joint-position targets. Joint-position proprioception and action targets are standardized using their training-set means and standard deviations.

$\pi_{0.5}$. We fine-tune the official `pi05_base` checkpoint using the same OpenPI JAX pipeline and DexHoldem data adapter. Like π_0, this configuration predicts absolute joint-position targets. Its proprioception and actions use quantile normalization: the training-set 1st and 99th percentiles define the affine mapping to $[-1, 1]$, with no clipping in the normalization transform. Values outside these

Table 6: Shared input and optimization settings for the OpenPI π_0 and $\pi_{0.5}$ configurations. Normalization differs as described in the model-specific paragraphs.

Setting	Configuration
Demonstrations	1,400 training and 70 validation trajectories across 14 primitives
Visual input	RGB only, resized to 224×224 per view; no depth input
State/action dimensions	30 robot joints, zero-padded to 32 model dimensions; outputs retain the first 30 dimensions
Observation horizon	1
Prediction horizon	50 action steps
Execution length	32 action steps per query at 10 Hz, without action filtering
Action representation	Absolute joint-position targets for both models
Fine-tuning	All model parameters; exponential moving average decay 0.999
Optimizer	AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 10^{-8}$
Weight decay	10^{-10}
Gradient clipping	Global gradient norm clipped to 1.0

percentiles can therefore fall outside this interval. Both models consume RGB images without depth; their shared input and optimization settings are summarized in Table 6.

BeingH. BeingH wraps the pretrained Being-H 0.5-2B vision-language-action model for the DexHoldem embodiment. Because Being-H operates in a 200-dimensional unified action space, the wrapper zero-pads the 30-dimensional robot state and action into that space, applies a validity mask so the loss is computed only on robot dimensions, and slices the robot-valid dimensions after flow-matching inference. Task-specific instruction tokens are inserted into the packed multimodal sequence so the pretrained model remains language conditioned.

BeingH Deployment Note. Although we completed finetuning of Being-H 0.5-2B on the DexHoldem data, we do not include this model in the main benchmarked-policy comparison. In preliminary real-robot deployment, the finetuned policy exhibited high-frequency joint oscillations and unstable motion. Based on discussion with the Being-H authors, deployment on a specific physical platform requires robot-specific action filtering to suppress such instability. This filtering would introduce an additional platform-dependent controller outside the standardized DexHoldem policy interface, and it could not be implemented consistently within the current benchmark protocol. We therefore treat BeingH as an implemented adaptation, but exclude it from the set of benchmarked real-robot models reported in the main text.

System-Level Evaluation Protocol. System-level evaluation composes perception, routing, primitive execution, verification, and recovery. A rollout starts from an initial tabletop state and repeatedly captures the table, parses the current state, applies router gates for waiting, verification, completion, continuation, recovery, or human-help escalation, invokes the main agent only when a new high-level decision is required, translates the selected agent primitive from Table 4 into zero or more dexterous-policy primitives from Table 8, executes the selected policy when physical motion is required, and verifies the post-condition from the next observation. A `request_human` dispatch does not terminate the rollout: it sets `loop_stage` to `down` and pauses the loop until the conductor restores the scene and acknowledges the request, which clears the consecutive-wait and recovery counters and re-enters the loop at `to_recover`. Exhausting the per-step retry budget (2), the total recovery budget (8), or the consecutive-wait budget (20) likewise routes to `request_human` rather than ending the rollout. A rollout ends when the hand reaches a terminal table outcome and the agent stops, or when the experiment conductor stops it because the robot can no longer make progress (an unrecognized disruption, indefinite waiting, or a scene that cannot be restored after a human-help request). A retry is triggered after the current atomic primitive has remained stuck for a prolonged period with the scene intact. Without retrying at such a stalled execution point, the conductor would have to terminate the rollout because continued waiting cannot restore progress. We report hand-level completion, retry outcomes, and human-help reasons (Section 4.5), and classify each completed hand by whether it contains a verified retry, a `request_human` dispatch, both, or neither; a completion is unassisted if its rollout contains no `request_human` dispatch.

Table 7: Shared evaluation conditions and model-native differences in representation and training.

Aspect	Configuration
Data and scoring	100/5 train/validation trajectories per primitive; 80 scheduled trials per policy; common SP/DC/TF/DF categories
Robot interface	Three policy camera views, proprioception, 30-dimensional joint targets; independent agent camera
Task condition	Natural-language tokens for π and RDT families; discrete IDs for task-specific imitation baselines
Representation	Frozen DINOv2 for DP (DINO); SigLIP/T5 representations for RDT; trainable ResNet encoders for DP-UNet
Policy initialization	Pretrained policy initialization for π_0, $\pi_{0.5}$, and RDT; no pretrained policy backbone for RDT-small or imitation baselines
Training budget	Learning rate 10^{-5}; 300 epochs from scratch, 20 epochs for RDT fine-tuning; OpenPI settings in Table 6
Mini-batch size	DP (DINO), DP-Transformer: 128; DP-UNet, BAKU: 256; ACT: 512; RDT-small: 64; RDT: 32
Gradient accumulation	RDT-small: 2 mini-batches; RDT: 4 mini-batches
Checkpoint selection	Lowest loss on the fixed validation split
Action adaptation	π_0 and $\pi_{0.5}$: absolute joint-position targets with a 50-step prediction horizon; other horizons are model-dependent; returned chunks execute open loop between updates
Control execution	All policies execute 32-step chunks at 10 Hz, open loop between policy queries; no action filtering

Deployment Runtime. Real-robot evaluation uses a client–server deployment architecture shared across the implemented policy families. A robot-side client maintains the hardware connection, captures observations from the manipulator and cameras, and sends policy requests to a desktop server equipped with 4× NVIDIA RTX 4090 GPUs. The policy server communicates through ZeroMQ, with port `13579` used as the default endpoint, and returns executable robot actions rather than model-internal predictions. This separation keeps GPU inference, model-specific preprocessing, and checkpoint loading off the robot-control process while preserving a common runtime interface for all benchmarked policies.

Live Observation Conversion. At each policy query, the robot client packages synchronized observations from the three RealSense RGB-D cameras together with the current robot joint state and the selected task instruction. Each model adapter selects the visual channels used during training, orders proprioceptive joint positions in the canonical 30-dimensional robot order, and supplies either a discrete instruction ID or a language prompt. For π_0 and $\pi_{0.5}$, the visual inputs contain only RGB images resized to 224×224; depth is not passed to the policy. The server applies each model's normalization transform using the statistics saved with its checkpoint. This online conversion mirrors the corresponding training-time data layout while preserving the common robot interface.

Chunked Control Execution. The deployment server unnormalizes model predictions and maps them to executable 30-dimensional joint-position targets before returning a 32-step action chunk to the robot client. All evaluated policies use this execution length and a control frequency of 10 Hz, with no action filtering. Each chunk is executed open loop before the next observation-conditioned policy query. The prediction horizon and executed chunk length are separate settings: for π_0 and $\pi_{0.5}$, the OpenPI server returns the first 32 actions of the 50-step prediction. Model-specific horizons and internal action dimensions are handled inside the adapter, while the robot receives the same 30-dimensional command format.

π-Series and OpenPI Adaptation. The π_0 and $\pi_{0.5}$ implementations use the OpenPI policy loader with a DexHoldem observation-action adapter during deployment. The adapter maps the top-down, third-person, and wrist-mounted RGB streams into OpenPI image fields, forwards the task prompt as language context, and zero-pads the 30-dimensional joint state to the model's 32-dimensional input. Both policies predict absolute joint-position actions. After applying the corresponding inverse normalization, the adapter retains the first 30 action dimensions, and the server returns the configured execution prefix to the robot client.

A.2 DEXTEROUS HAND POLICY BENCH DETAILS

Primitive-Level Skill Tasks. Each primitive is indexed by an instruction ID, paired with the natural-language instruction used by language-conditioned policies, and evaluated by a physical post-condition. All directional terms use a single ShadowHand robot-facing tabletop frame: robot-left and robot-right are the left and right sides from the robot player's perspective, push moves a target chip away from the robot into the forward betting region, and pull moves it back toward the robot-side region. Camera, viewer, and rendered-figure orientations do not redefine these task labels. A scene-preserving success requires both satisfying the requested primitive and leaving non-target cards and chips in positions that allow subsequent tasks to continue.

Table 8: Primitive-level task definitions. Each primitive provides 100 training and 5 validation teleoperated trajectories under a uniform split. Real-world evaluation uses 10 rollouts per pickup primitive (which also seed downstream card-placement and card-revealing trials) and 5 scheduled trials per other primitive, for 80 primitive-level trials per policy. Policy instructions are taken from the benchmark instruction file, with all left/right and push/pull directions interpreted in the robot-facing frame defined above.

ID	Primitive	Policy instruction
0	`pick_up_left`	Pick up the card on the left side.
1	`pick_up_right`	Pick up the card on the right side.
2	`push_5`	Push forward the chips worth 5.
3	`push_10`	Push forward the chips worth 10.
4	`push_50`	Push forward the chips worth 50.
5	`push_100`	Push forward the chips worth 100.
6	`pull_5`	Pull back the chips worth 5.
7	`pull_10`	Pull back the chips worth 10.
8	`pull_50`	Pull back the chips worth 50.
9	`pull_100`	Pull back the chips worth 100.
10	`put_down_left`	Place the held card onto the left position.
11	`put_down_right`	Place the held card onto the right position.
12	`show_left`	Reveal the face of the left card.
13	`show_right`	Reveal the face of the right card.

Physical Scoring Protocol. The following scene-preservation rules were used for the reported physical evaluation. Task completion and disruption are adjudicated independently after the robot and objects settle. The playable region is the tabletop surface visible in a designated reference camera, represented by a calibrated tabletop mask rather than the whole camera image. An object leaves this region if it falls off the table or its entire tabletop footprint lies outside the mask; touching or partially crossing the boundary alone is not a violation. Task-required lifting and holding of the target card are permitted changes, not a loss of that object from the scene.

A terminal scene is disruptive if any task-relevant object leaves the playable region, if objects expected to remain separate physically overlap or stack, or if an object outside the primitive's allowed-to-move set exceeds its displacement tolerance. Pre-existing stacks and contacts explicitly required by the task are excluded from the overlap criterion. For approximately circular objects, such as chips and the dealer button, the tolerance is one object radius; for non-circular objects, such as cards or card boxes, it is half the shortest side. Displacement is evaluated in calibrated tabletop coordinates relative to the initial footprint, including changes due to in-plane rotation, rather than center translation alone. Temporary occlusion by the robot is not itself disruption. If no disruption condition is present, the scene is classified as preserved.

The target goal is checked separately at each primitive's endpoint. For pickup (IDs 0–1), the selected card must be clear of the tabletop and remain supported by the hand after the pickup motion has settled. Merely touching or sliding the card, lifting only an edge while the card remains supported by the table, or briefly lifting it and dropping it before the endpoint does not satisfy this goal. Recognizing the card's rank and suit is a separate agent-level operation. Put-down (IDs 10–11) starts with the card held by the hand and is complete only after the hand releases it and it comes to rest face down at the requested left or right position. A card that remains grasped, rests face up, or settles at the wrong

side does not satisfy the put-down goal. Show requires revealing the requested card face; each push or pull primitive has one intended chip target of the requested denomination, selected as specified below. Table 9 summarizes these task relations. A completed pickup is judged before any subsequent put-down; later placement failure does not change whether the preceding pickup reached its endpoint goal. Completion does not override a disturbance elsewhere in the scene.

In the paired pickup–put-down evaluation, failure to pick up the selected card automatically makes the corresponding put-down trial a task-completion failure. The put-down trial remains in the scheduled denominator even though its required held-card starting state was not reached. It is classified as TF if the scene is preserved and DF if the scene is disrupted. Thus, the reported put-down rates include failures inherited from pickup.

Chip Target Selection and Scoring. Each chip-motion primitive is intended to move one chip. For push, the target is the requested-denomination chip at the highest-priority occupied position in the demonstration layout:

$$\begin{array}{ll} 3 & \\ 2 \quad 5 & \text{priority: } 5 \to 4 \to 3 \to 2 \to 1. \\ 1 \quad 4 & \end{array}$$

These numbers index spatial positions, not denominations or primitive IDs. Successive push calls follow this priority, consistent with the demonstration behavior, and move the selected chip into the forward betting region. For pull, any one chip of the requested denomination may be selected and moved into the corresponding robot-side destination; no particular instance is prescribed when multiple chips share that denomination. For either operation, only the single target chip is allowed to move beyond the scene-preservation tolerance. Other chips, including those of the same denomination, remain subject to the non-target displacement and overlap rules. Task completion is determined by whether the target reaches its destination; moving an additional chip does not negate completion. In particular, moving two same-denomination chips into the destination is scored as DC, not SP: the target goal is met, but the extra chip constitutes scene disruption.

Table 9: Primitive goal relations used for task-completion scoring. Displacement tolerances and endpoint rules are specified in Section A.2.

IDs	Target and intended change	Goal relation
0–1	Selected left/right card; grasp and lift	Card remains held clear of the tabletop at the pickup endpoint
2–5	One requested-denomination chip; spatial priority $5 \to 4 \to 3 \to 2 \to 1$	Selected target reaches the forward betting region
6–9	Any one requested-denomination chip; pull	Selected target reaches its robot-side destination
10–11	Held card; release at left/right position	Card is released and rests face down at the requested side
12–13	Requested card; reveal	Requested card face is exposed

Scoring release. We will release the reference-camera tabletop mask as a polygon in normalized image coordinates, its tabletop calibration, the footprint-displacement procedure, primitive-specific allowed-to-move sets and goal tolerances, and timeout and endpoint-adjudication rules. The guide will include annotated positive and negative examples for all four categories. These assets make the geometric and task-specific decisions explicit; the four-category schematic in Figure 4 summarizes the scoring logic. The project website provides online scoring guidelines, including the four outcome categories and the SPSR/TCR definitions.

Demonstration Dataset.

Hardware Setup. DexHoldem is collected on a real Texas Hold'em tabletop environment using a ShadowHand mounted on a UR10e arm. The setup belongs to the same broader class of real-world dexterous-hand data-collection systems as RealDex and DexH2R, which also emphasize physical robot demonstrations, teleoperation, and rich visual sensing for dexterous manipulation (Liu et al., 2024; Wang et al., 2025b). In contrast to grasp-only or handover-only settings, our scene is organized around card and chip manipulation on a fixed tabletop layout, where the robot must preserve the

Table 10: Summary of the DexHoldem benchmark setup and collected demonstration dataset. Primitive definitions, train/validation splits, and evaluation schedules are detailed in Table 8; model-facing observation and action formats are detailed in Section A.1.

Item	Value
Platform	ShadowHand (Shadow Robot Company, 2025a) mounted on a Universal Robots UR10e arm in a real-world tabletop setup
Objects	Standard poker cards and four poker-chip denominations: 5, 10, 50, and 100
Sensors	Three policy-side RealSense D435i RGB-D cameras (top-down, third-person, wrist), robot proprioception, and a separate dedicated agent-view camera
Skill set	14 instruction-conditioned atomic manipulation primitives
Demonstrations	1,470 teleoperated trajectories, with 105 demonstrations per primitive

surrounding game state while executing each primitive. The objects are standard poker cards and poker chips with denominations 5, 10, 50, and 100.

Reference Scene Geometry. The world origin is the center of the upper tabletop surface, with $+X$ along the table's short side, $+Y$ along its long side, and $+Z$ vertically upward. The table measures $0.60 \times 1.40 \times 0.10\,\mathrm{m}$; its top surface spans $x \in [-0.30, 0.30]\,\mathrm{m}$ and $y \in [-0.70, 0.70]\,\mathrm{m}$ at $z = 0$. The UR10e–ShadowHand base is at $(-1.45, -0.12, 0.10)\,\mathrm{m}$ with quaternion $(w, x, y, z) = (1, 0, 0, 0)$. The reference scene configuration specifies the collision surface, visual top layer, support geometry, and initial poses of nine cards and 21 chips. This geometry also defines the reconstruction used in Section C; evaluation layouts vary as described below.

Multi-View Observations. The sensing layout uses three Intel RealSense D435i RGB-D cameras to cover complementary spatial scales. A top-down view captures the card and chip layout, a third-person view observes the arm and global scene geometry, and a wrist-mounted view provides close-range evidence for hand–object contact and placement. In the world frame above, the top-down camera is at $(-0.18, -0.05, 0.70)\,\mathrm{m}$ and looks at $(-0.18, -0.05, 0)\,\mathrm{m}$. The third-person camera is at $(-1.15, -0.30, 0.50)\,\mathrm{m}$ and looks at $(-0.25, 0.20, 0)\,\mathrm{m}$. The wrist camera is rigidly attached to `rh_forearm` with local offset $(0, -0.10, 0.10)\,\mathrm{m}$ and views along its local optical $+Z$ axis. The data collector records RGB and depth streams at approximately 15 Hz, aligns each depth stream to its corresponding color stream, and stores the frames with the robot state for each trajectory.

Skill Suite. The demonstration set covers the 14 atomic primitives used throughout the benchmark. These primitives include two card-pickup tasks, two face-down card-placement tasks, two card-revealing tasks, and eight chip-motion tasks that vary by denomination and push/pull direction. Each trajectory is paired with an instruction ID and the corresponding natural-language task description. Primitive names, instructions, the train/validation split, and real-world evaluation schedule are provided in Section A.2 and Table 8.

Teleoperated Data Collection. Demonstrations are collected with a Vive-based Shadow teleoperation system. Shadow Robot's technical specification describes a teleoperation stack built around UR10e arms, Shadow Dexterous Hands, Shadow Gloves, HTC Vive tracking hardware, a pedal interface, emergency-stop devices, and ROS-based software infrastructure (Shadow Robot Company, 2025b). In our collection, the operator uses this teleoperation interface to produce successful executions for each primitive, while the benchmark recorder logs multi-view RGB-D observations, robot joint states, instruction IDs, and 30-dimensional joint-position action targets. Failed attempts are excluded from the released demonstration set through primitive-specific success checks, so the final dataset contains 105 accepted demonstrations for each primitive.

Primitive-Level Evaluation. Primitive-level evaluation measures whether a policy can execute each of the 14 atomic skills under the fixed physical schedule in Table 8. Each primitive is tested under five evaluation configurations; the two pickup primitives are additionally repeated because they initialize downstream card-placement and card-revealing trials, yielding 80 scheduled primitive-level trials per policy, including put-down trials automatically counted as failures after an unsuccessful pickup. Before each independently initialized rollout, the dexterous hand and tabletop objects are

Table 11: Primitive-group success rates for the physical policy evaluation. Each entry reports SPSR/TCR in percent: the left value counts only scene-preserving successes, while the right value counts both scene-preserving successes and disruptive completions. The overall column matches the aggregate results in Table 1.

Policy	Pickup 20 trials	Chip push 20 trials	Chip pull 20 trials	Put-down/show 20 trials	Overall 80 trials
$\pi_{0.5}$	100.0/100.0	25.0/35.0	15.0/30.0	50.0/80.0	47.5/61.2
π_0	100.0/100.0	25.0/30.0	15.0/20.0	50.0/80.0	47.5/57.5
RDT	75.0/80.0	15.0/25.0	5.0/10.0	25.0/70.0	30.0/46.2
DP (DINO)	50.0/50.0	25.0/45.0	10.0/20.0	20.0/30.0	26.2/36.2
DP-Transformer	25.0/25.0	10.0/15.0	15.0/20.0	5.0/20.0	13.8/20.0
RDT-small	25.0/25.0	15.0/20.0	5.0/5.0	10.0/20.0	13.8/17.5
ACT	25.0/30.0	5.0/5.0	0.0/0.0	10.0/25.0	10.0/15.0
BAKU	20.0/30.0	0.0/0.0	0.0/10.0	5.0/10.0	6.2/12.5
DP-UNet	0.0/0.0	0.0/0.0	5.0/5.0	0.0/0.0	1.2/1.2

reset, and the initial tabletop configuration is varied within the benchmark layout. For each chip-motion primitive, the five-trial evaluation uses standard chips in three trials and chips with attached tape in two trials to vary surface properties and friction. For chip-pushing primitives, each target denomination is tested once in scenes containing 1, 2, 3, 4, and 5 chips, respectively; the target chip is always present, and the remaining chips act as distractors. For chip-pulling primitives, each target denomination is evaluated on five layouts derived from an initial five-chip scene containing chips on both the left and right sides, all four chip denominations, and a second instance of the target denomination. Either instance of the requested denomination may be the single pull target. The first trial uses this full layout, and the remaining four trials progressively remove one chip from the left side twice and one chip from the right side twice. Each physical rollout is labeled with the four-category rubric in Sections A.2 and 3.1: scene-preserving success, disruptive completion, task failure, or disruptive failure. We report exact outcome counts, scene-preserving success rate (SPSR), and task completion rate (TCR), which counts both scene-preserving successes and disruptive completions.

Primitive Group Analysis. Table 11 decomposes the aggregate policy results in Table 1 into four balanced primitive groups. Each group contains 20 real-world trials per policy: pickup contains the two card-lifting primitives, chip push and chip pull each contain four denomination-specific chip-motion primitives, and put-down/show contains the two face-down card-placement primitives and the two card-revealing primitives. Each table entry uses the format SPSR/TCR, where the left number is scene-preserving success rate and the right number is task completion rate, both in percent. For example, 25.0/35.0 means that 25.0% of trials completed the requested primitive while preserving the scene, while 35.0% completed the primitive if disruptive completions are also counted. We use this left-to-right ordering to match the aggregate metrics in Table 1 and to make the gap between clean completion and disruptive completion visible within each primitive group.

The grouped results show that the aggregate scores are not driven uniformly across primitive types. The strongest π-series policies solve pickup reliably, but their chip-motion success rates remain much lower, especially for chip pull. Put-down/show tasks expose a different failure mode: several policies complete the requested card placement or reveal at a higher rate than they preserve the full scene, producing a large SPSR–TCR gap. This pattern suggests that DexHoldem separates object-level task completion from interaction precision, and that stronger aggregate performance still leaves substantial room for policies that can move chips and reveal cards without disturbing the surrounding tabletop state.

A.3 Agentic Perception Bench Details

The DexHoldem agentic perception benchmark contains 36 problems, `p1`–`p36`, drawn from states encountered during system-level deployment. Each run receives a single current tabletop observation, the action-sequence context, the robot's hole-card cache, and permitted annotated predecessor states. Visual evidence determines the current state; predecessor context supports inheritance of durable fields. Only the structured summary is scored against held-out labels.

Model and Harness Configuration. All perceivers use the same `v2-native` workflow, user prompt, ten field-specific visual guidelines, and absence of additional runner-level budgets. This workflow performs perception directly and installs no visual subagents. Model names are specified in Table 2, and Table 12 summarizes their shared harness settings. The GPT 5.6/6 models are invoked as `gpt-5.6-luna`, `gpt-5.6-sol`, `gpt-6-astra`, and `gpt-6-sol`.

Table 12: Shared perception settings by native harness. Models are named in Table 2. All use the same native workflow, full prompt, visual guidelines, and output schema, without additional runner-level token, tool-call, turn, or time limits.

Harness	Reasoning / service	Permission and session settings
Codex	medium / fast	Workspace-write; ephemeral session; plugins and apps disabled
Claude Code	medium / default	`acceptEdits`; no session persistence
Gemini CLI	no override / default	`yolo` approval mode; `skip-trust`

Tools, Budgets, and Termination. The Codex configuration uses an isolated problem copy, a run-local configuration, workspace-write permissions, and an ephemeral session. Its invocation is `codex exec` with `-disable=plugins`, `-disable=apps`, `-skip-git-repo-check`, `-ephemeral`, and `-C` pointing to the problem copy. The batch scheduler runs up to six independent problems concurrently. Its `max_threads=9` setting applies only to subagents, which native perception does not install. Image inspection, context-file reading, and output-file writing use the harness's native tools, and the prompt prohibits helper-script execution and robot actions.

The runners impose no additional per-problem token cap, tool-call cap, turn cap, or subprocess timeout. They wait for the native CLI process to exit and then validate its artifacts; native harness and provider limits still apply, so inference compute follows each harness rather than being equalized across models. For Claude, `-effort medium` is passed with `-permission-mode acceptEdits` and `-no-session-persistence`. The Gemini runner passes `-approval-mode yolo` and `-skip-trust` with no reasoning-effort override. These harness-specific settings are shared across models within each harness.

Prompt and Visual Guidelines. Figure 6 shows the full user prompt shared by all perceivers, with the concrete run ID replaced by `<run_id>`. The accompanying provenance bundle contains `prompt.txt`, the installed native-setup instruction, the runner and preflight sources, and byte-exact copies of all ten visual guidelines: bet recognition, blind-button recognition, chip recognition, community cards, held-card recognition, robot behavior, scene stability, showdown outcome, table geometry, and turn detection. The guidelines are copied from `skills/dexholdem-v2-native/visual_guidelines/`; they specify spatial regions, interpretation rules, and uncertainty handling. Native preflight copies these guidelines into the workspace and writes the setup instruction there as a separate file.

The bundle's `manifest.json` records checksums of the prompt, source files, visual guidelines, and per-run configuration artifacts to identify the materials used for evaluation.

Table 13 reports the per-column statistics of the released bench: each row gives the number of problems that contribute to one scoring column under the deterministic evaluator, together with the exact problem IDs in that subset. The deterministic evaluator scores nine columns—an Overall column plus eight sub-capability columns—grouped by applicability. The universal columns loop stage (**LS**), turn ownership (**TO**), and blind assignment (**BI**) are scored on all 36 problems. The chip-state columns current bet chips (**CB**), robot chip inventory (**RCI**), and opponent chip inventory (**OCI**) are scored only on the 16 `table_decision` and `outcome_judge` problems, where chip and bet state are routing-relevant; the chip and bet dictionaries must match exactly across all four denominations (5, 10, 50, 100) on each side. The community-card column (**CC**) is scored on the 13 problems within that subset that have visible community cards (3, 4, or 5 cards), with order-insensitive set matching. The showdown-outcome column (**SO**) is scored only on the 7 `outcome_judge` problems and requires the agent to declare `win` or `lose` from visible cards or from a detected opponent fold. An `outcome_judge` problem must satisfy all eight sub-capability columns simultaneously, while `turn_gate`, `robot_progress`, `held_card_read`, and `recovery_safety` problems only need the three universal columns. Because Overall is conditioned on the per-problem applicable set rather than averaged over fields, it is intentionally not an average of the sub-column

````
Run the current DexHoldem perception step.

# Task
Read the capture image at `s_current/00_capture.jpg` and extract visual state.
Follow the field-specific guidelines in `visual_guidelines/`. Do not execute
robot actions, run helper scripts, or choose a poker action.

# Context files
- `action_sequence.json`: PRIOR workflow state (what was dispatched when the
  image was captured). Use as CONTEXT only -- visual evidence determines the
  CURRENT state.
- `hole_card_cache.json`: cached robot hole cards for showdown comparison.
- `sN-1/01_parsed_state.md`: previous parsed state for inheriting durable fields.

# Output
Write to `runs/<run_id>/`:
- `visual_raw/native.md` -- raw perception evidence
- `visual_summary.json` -- structured output (schema below)
- `eval_report.md` -- brief summary of evidence and merging

Schema for `visual_summary.json`:
```json
{
  "loop_stage": "idle",
  "blind": "big_blind",
  "showdown_outcome": "not_showdown",
  "table": {
    "scene_stable": true,
    "is_my_turn": true,
    "community_cards": [],
    "my_chips": {"5": 4, "10": 3, "50": 3, "100": 3},
    "opponent_chips": {"5": 4, "10": 4, "50": 3, "100": 3},
    "my_current_bet": {"5": 0, "10": 0, "50": 0, "100": 0},
    "opponent_bet": {"5": 0, "10": 0, "50": 0, "100": 0},
    "uncertain_fields": []
  }
}
```

Field values:
- `loop_stage`: idle, acting, atom_idle, down, to_recover, win, lose, show_hand
- `blind`: big_blind, small_blind, none
- `showdown_outcome`: win, lose, tie, not_showdown
- Chip fields: objects with keys "5", "10", "50", "100" (integer counts)
- `uncertain_fields`: field names with inherited/inferred/low-confidence values

Do not write null or placeholder values -- every field needs a concrete estimate.
Verify outputs exist before finishing.
````

Figure 6: Complete perception prompt. The runner substitutes the concrete run ID for `<run_id>`. The prompt, output schema, and field-specific visual guidelines are shared across perceivers.

Table 13: Per-column problem applicability for the 36-problem perception benchmark. Each row lists the problem IDs that contribute to one scoring column under the deterministic evaluator.

Column	#Problems	Problem IDs
Overall	36	p1–p36
LS (loop stage)	36	p1–p36
TO (turn ownership)	36	p1–p36
BI (blind info)	36	p1–p36
CC (community cards)	13	p13, p14, p18, p19, p20, p23, p27, p30, p31, p32, p33, p35, p36
CB (current bet chips)	16	p9, p11, p13, p14, p18, p19, p20, p23, p26, p27, p30, p31, p32, p33, p35, p36
RCI (robot chip inventory)	16	p9, p11, p13, p14, p18, p19, p20, p23, p26, p27, p30, p31, p32, p33, p35, p36
OCI (opponent chip inventory)	16	p9, p11, p13, p14, p18, p19, p20, p23, p26, p27, p30, p31, p32, p33, p35, p36
SO (showdown outcome)	7	p19, p23, p30, p31, p32, p33, p36

accuracies and can exceed a difficult sub-column accuracy that is evaluated on a smaller, more specialized subset of problems.

Coverage and history-copy diagnostic. The evaluator does not directly score held-card identity, `scene_stable`, or `uncertain_fields`. In particular, a correct `held_card_read` problem means that LS, TO, and BI match; it does not demonstrate a correct card read. Problem types describe the scene context, whereas the applicability table defines the measured targets.

We add a deterministic previous-state-copy baseline using the same released problem inventory, normalization, column matching, and applicability rules. It copies only the latest provided predecessor's structured annotation, without reading target pixels or the target annotation. Twelve problems have no predecessor and count as incorrect, so the denominator remains 36. It scores 8/36 Overall (22.2%), or 8/24 on the subset with history. By column it obtains LS 10/36, TO 23/36, BI 17/36, CC 6/13, CB 5/16, RCI 6/16, OCI 7/16, and SO 0/7. This tests direct history reuse, not an ablation of each model's use of context. Missing-history cases and the fixed problem mixture should be considered when interpreting comparisons.

Table 14: Previous-state-copy exact match by problem type. Problems without predecessors remain in the denominator and count as failures. This baseline uses no model inference.

Problem type	Problems	No history	Exact match
Turn gate	5	5	0/5 (0.0%)
Robot progress	8	0	4/8 (50.0%)
Held-card read	2	0	0/2 (0.0%)
Recovery/safety	5	0	2/5 (40.0%)
Table decision	9	2	2/9 (22.2%)
Outcome judge	7	5	0/7 (0.0%)
All	36	12	8/36 (22.2%)

Agentic Perception Evaluation. Agentic perception evaluation follows the benchmark design in Section A.3. Each run receives the benchmark observation and prompt artifacts for one tabletop state, writes the required visual-summary and evidence artifacts, and undergoes artifact validation. Missing, invalid, or unparsable outputs count as incorrect and remain in each applicable column's denominator; only valid predictions proceed to semantic matching. The structured visual summary is then compared with ground-truth labels using deterministic column-level checks over the eight perception challenges defined in Section 3.2: loop stage, turn ownership, blind information, community cards, current bet chips, robot chip inventory, opponent chip inventory, and showdown outcome. The overall perception score is exact match over the scored challenges applicable to the state, conditioned on the provided annotated history.

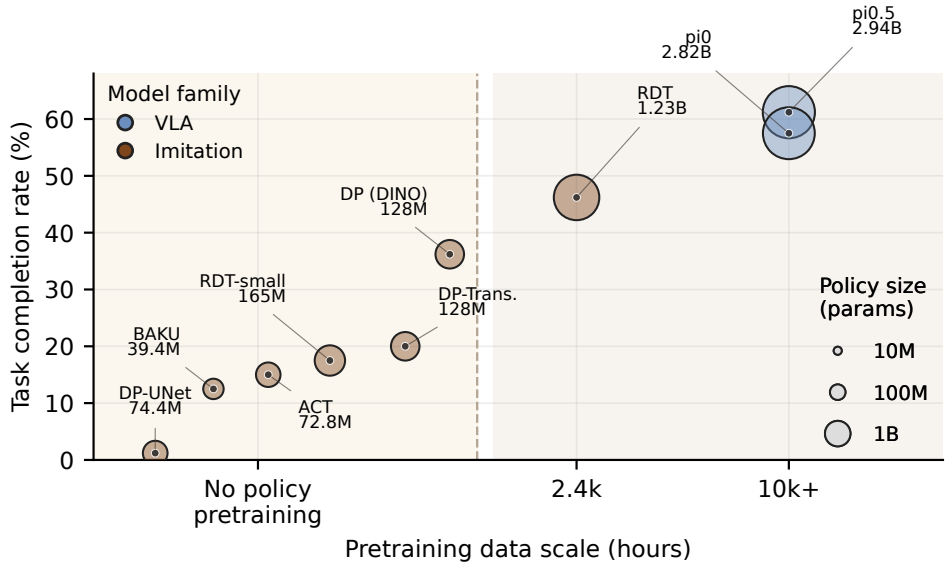


Figure 7: Policy pretraining data scale, policy size, and physical task completion rate on DexHoldem. Models without policy pretraining are grouped at the zero-pretraining tick and lightly spread for visibility, while pretrained models are placed by their stated or estimated pretraining-hour values on the compressed x-axis. Marker size and the value under each model name report policy-only parameter count, excluding visual encoders. RDT uses an estimated 2,400 hours, and $\pi_0/\pi_{0.5}$ use the reported 10k+ hour lower bound.

B Experiment Details

B.1 Policy Pretraining Scale Diagnostic

Figure 7 reports the standalone diagnostic that relates policy pretraining scale, policy size, and physical task completion rate. The plot complements the aggregate policy results in Table 1 by placing task-trained imitation baselines and adapted pretrained policies on a common visual axis. This diagnostic is intended to summarize relative scale and observed completion behavior, while the quantitative comparison in the main text should be read from the trial counts and rates in Table 1.

B.2 RDT Fine-Tuning Curve Details

We report the full train-time validation curves for the RDT fine-tuning data-scaling probe in Figure 8, which supports the data-scaling analysis in Section 4.3. This diagnostic compares the same RDT architecture under two initialization regimes: random initialization and initialization from the pre-trained RDT checkpoint. For each data ratio, both regimes use the same DexHoldem task subset and the same fixed validation split. The 10%, 20%, 50%, and 100% settings correspond to 10, 20, 50, and 100 training trajectories per primitive, respectively, sampled from the 100-trajectory training split defined in Table 8; the five held-out validation trajectories per primitive are unchanged across all ratios.

The paired curves should be interpreted under the held-out validation-loss criterion. Each curve reports train-time validation loss over the completed paired seeds, and the shaded region denotes one standard deviation across those runs. Lower loss indicates better prediction of normalized action sequences under the supervised objective, so we treat lower validation loss as better held-out policy fit for this ablation. The curves show that pretrained initialization does not create a distinct low-data regime at 10% data, while it yields a modest lower-loss offset once more dexterous-hand demonstrations are available. These curves support a configuration-specific comparison of held-out action-prediction loss; they do not establish a causal mechanism for the difference or physical data efficiency.

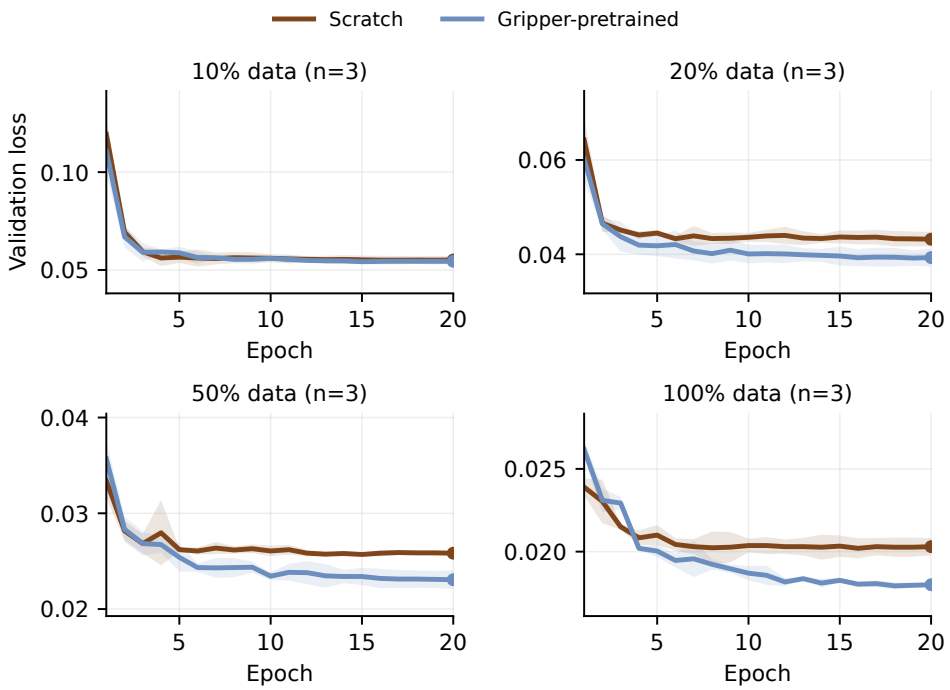


Figure 8: Train-time validation-loss curves for the representative RDT fine-tuning study across DexHoldem dexterous-hand data ratios. Each panel compares a randomly initialized RDT model with the same architecture initialized from the pretrained RDT checkpoint. The 10%, 20%, 50%, and 100% settings use 10, 20, 50, and 100 training trajectories per primitive, respectively, while preserving the same five validation trajectories per primitive. Curves show the mean over completed paired seeds, shaded bands denote one standard deviation, and lower validation loss indicates better held-out policy fit.

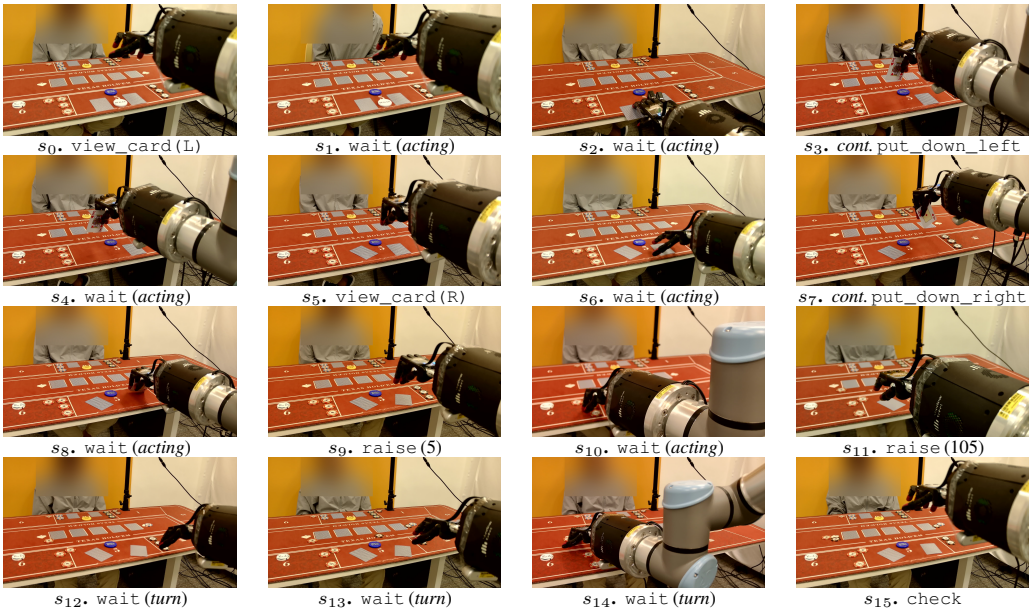
B.3 System-Level Trajectory Panels

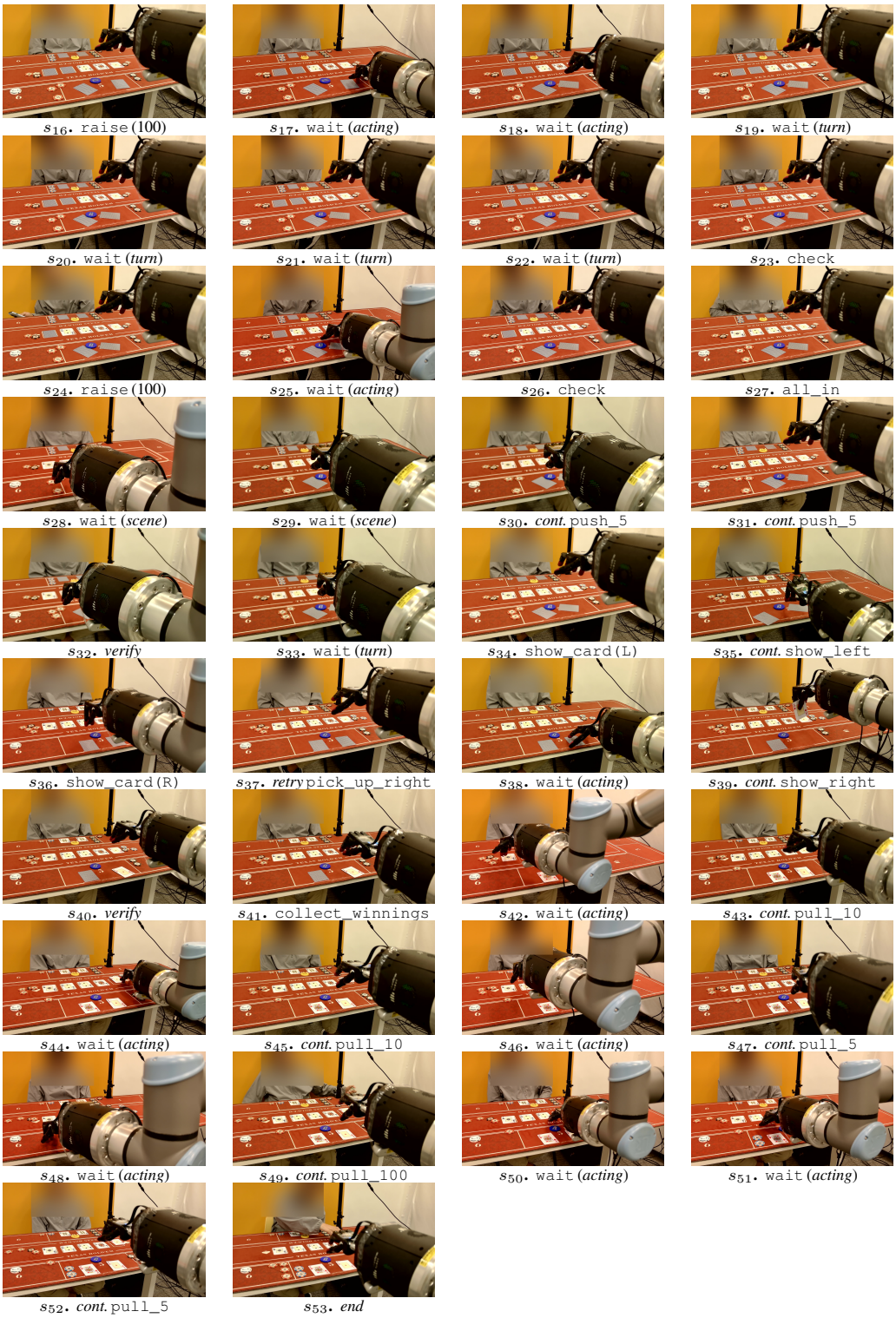
This section shows the per-state agent-view captures for the three system-level rollouts (i)–(iii) summarized in Table 3. All three are run with the Codex GPT 5.5 agent harness paired with π_0 as the dexterous primitive policy. State labels follow the agent-primitive to dexterous-policy-primitive mapping documented in Table 4: top-level agent primitives chosen by the main agent are written in `typewriter` with their arguments (e.g. `view_card(L)`, `raise(10)`, `check`, `call`, `show_card(R)`, `collect_winnings`, `request_human`); states inside the wait branch are written as `wait` (*reason*), where *reason* is one of *scene* (scene unstable), *acting* (robot still acting), or *turn* (not robot's turn); and intermediate router gates that advance a multi-atom translation already chosen by an earlier agent primitive are written as *cont.* `dexterous-policy primitive` (e.g. the next `push_5` atom in a `raise(105)` sequence). The *cache hole card* step is the visual read between the pickup and put-down halves of a `view_card` primitive, while *verify*, *complete*, and *retry* denote router-level verification, completion, and recovery gates. Cells labeled *end* mark the terminal state of the rollout.

Trajectory (i). 22 states; the agent dispatches `view_card` on both hole cards, twice escalates to `request_human` after a returned hole card lands outside its area (left) or face-up (right), and then plays the post-flop sequence `raise(10)`, `check`, `check`, `call`, with the final `call` interrupted before the chip-push translation completes.

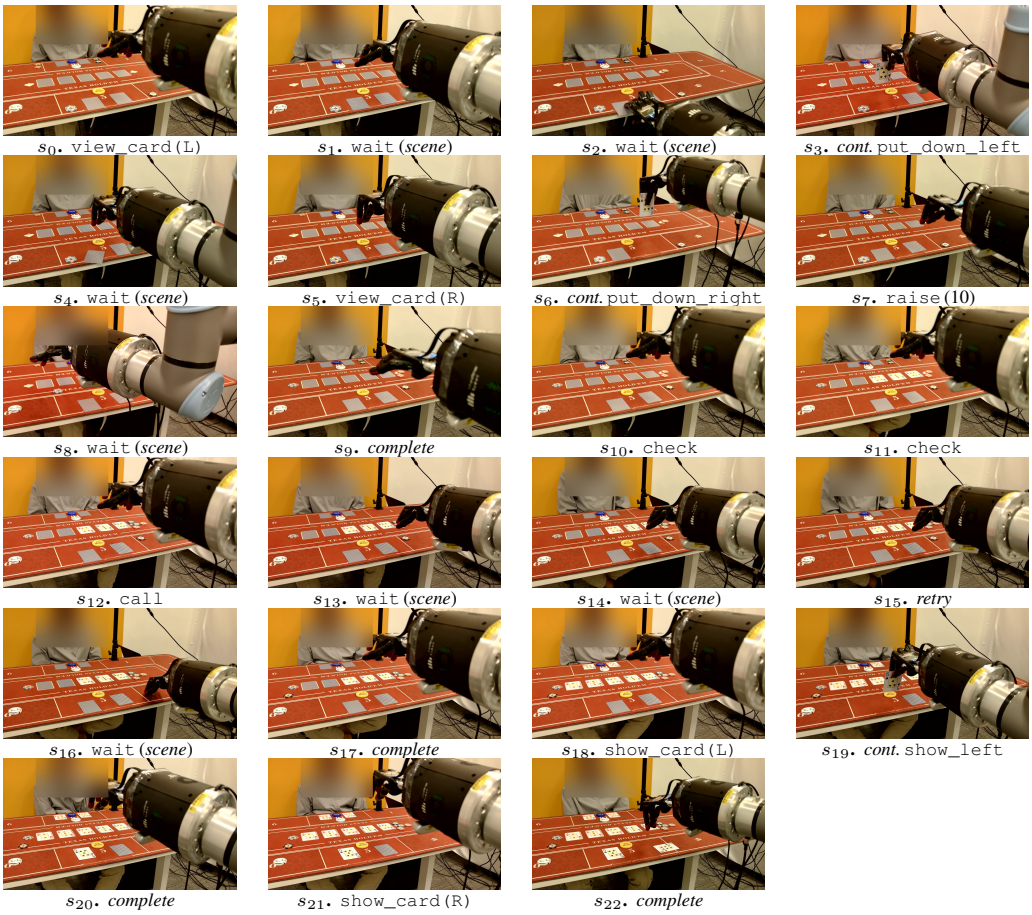


Trajectory (ii). 54 states; the longest of the three rollouts. The agent reads both hole cards, posts a small blind via `raise(5)`, escalates the bet through `raise(105)`, `raise(100)`, `raise(100)`, and `all_in`, completes a showdown with two `show_card` primitives (the second after one pickup retry), and finally executes `collect_winnings`, which expands into six chip-pull atoms across the opponent-bet and own-bet stacks.





Trajectory (iii). 23 states; the case study analyzed in Section 4.5. The agent reads both hole cards, plays `raise(10)`, `check`, `check`, `call` on the betting rounds, and finishes with a showdown via `show_card(L)` and `show_card(R)`, with one router-level *retry* along the way.



C SIMULATION CHECK

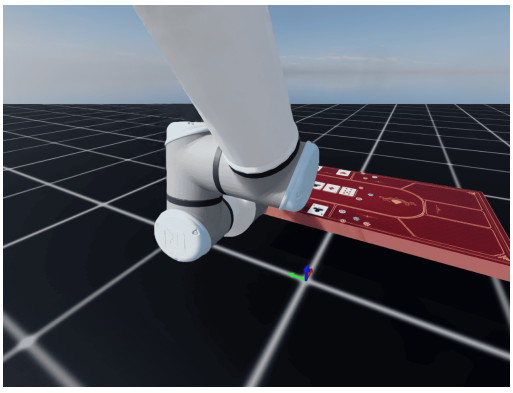
We include a simulation check to document the real-to-sim reconstruction used in DexHoldem. The purpose of this check is to verify whether recorded real trajectories can be replayed in a reconstructed simulation environment with the same robot embodiment, tabletop layout, object categories, and primitive definitions. This check is not used as an additional policy-evaluation experiment.

We reconstruct the simulation environment by matching the UR10e arm and ShadowHand, the poker table layout, the task-relevant cards and chips, and their source and target regions. This defines the simulator–task mapping used in the check: the robot initial state, task-relevant objects, object placements, and target regions are aligned with the real setup. We then replay the real arm and ShadowHand joint trajectories in simulation as open-loop commands. This allows us to check whether the replayed trajectory follows the intended primitive and whether the reconstructed scene is aligned with the real task setup.

Figure 9 shows the reconstructed simulation environment under the three camera views used in DexHoldem, with camera poses matched to the real system. Figure 10 further shows a representative top-down replay sequence, illustrating how a recorded real trajectory is replayed in the reconstructed real-to-sim environment.

This simulation check is qualitative and is not intended as a quantitative validation experiment. It documents the simulator–task mapping and replay consistency, but it does not remove the real-sim gap. This gap is especially relevant for thin cards and chips, where contact behavior is sensitive to friction, contact compliance, small pose errors, and tabletop disturbances. Therefore, the simulation check should be interpreted as documentation of reconstruction and replay consistency, while the policy results in the main paper are still reported from physical robot rollouts.





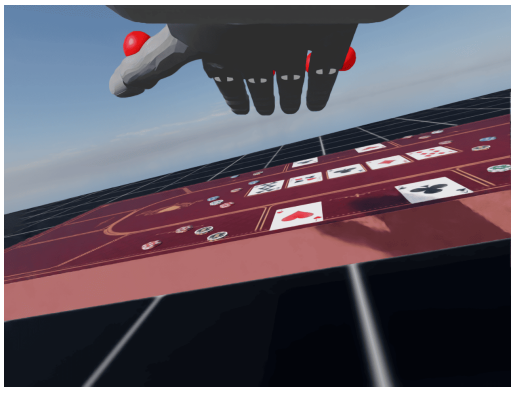


Figure 9: Simulation check through real-to-sim trajectory replay under the three camera views used in DexHoldem. The first row shows the top-down view, while the second row shows the third-person view and the wrist view. All replay frames are rendered from camera poses matched to the real multi-view camera setup.



Figure 10: Representative top-down replay sequence in the reconstructed simulation environment. The six frames show successive stages of replaying a recorded real-world trajectory in simulation, providing a qualitative check that the motion remains consistent with the intended primitive and the reconstructed task setup.