AGENT AS POLICY FOR ROBOTIC MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We demonstrate that a general-purpose agent can directly drive a physical robot throughout task execution without any task-specific or environment-specific training. We introduce **Agent as Policy** (AGP), which places task planning and execution under the agent's control. Given a task and a robot interface, the agent interprets visual evidence, writes executable programs, issues motion commands, and revises its actions in response to physical outcomes. This brings the agent's reasoning and programming capabilities into continuous interaction with the physical world. We study AGP across multiple real-world manipulation tasks spanning precision manipulation, dynamic motions, and deformable objects. These include assembly from human videos, block construction from goal images, dice flipping, targeted throwing, and bimanual towel folding. AGP achieves success rates of at least 80% in seven of eight task configurations and significantly outperforms previous agentic robot systems. We further study efficiency through task experience accumulation and find that reusing saved procedures and programs shortens execution time across repeated trials. These findings support a path for general-purpose agents to act as robot policies, extending their autonomy to physical manipulation through runtime reasoning, programming, and interaction.

1 INTRODUCTION

Multimodal large language models (MLLMs) demonstrate strong general capabilities (Li et al., 2026b). Agents built on these models tackle digital tasks such as software development and computer use (Yang et al., 2024; Wang et al., 2024; Xie et al., 2024). Recent work extends these agents to robot manipulation, bringing their task execution capabilities from digital environments into the physical world (Li et al., 2026a; Zhang et al., 2026; Galanti et al., 2026). These agentic robot systems broadly follow two approaches. The first approach uses agents to generate executable programs in advance, expressed as code or computation graphs that the robot runs during task execution (Liang et al., 2022; Chen et al., 2026). The second approach uses agents as orchestrators of existing learned policies, selecting and sequencing these policies according to task goals and execution feedback (Li et al., 2026a; Zhang et al., 2026; Galanti et al., 2026). Both approaches, however, have limitations. Generated programs must express runtime decisions through explicit rules, making it difficult to anticipate how uncertain observations and unexpected outcomes should affect execution. Policy orchestration gives agents control over which policy runs next, while motion generation remains limited by what the selected policies can perform.

Yet physical tasks often require the flexibility to adapt both how evidence is interpreted and how actions are generated. After a failed insertion, for example, the robot may need a closer view, a revised alignment estimate, or a different approach direction. Choosing and carrying out the appropriate response requires reasoning jointly about the current scene, the attempted motion, and its outcome. To this end, we introduce **Agent as Policy (AGP)**, which uses the agent itself as the policy throughout task execution. We grant the agent full autonomy over runtime task decisions: it decides what to observe, how to interpret evidence, and which motion to request, and can author and revise local programs as needed. Each result informs the next decision, including whether to gather more evidence or change the execution strategy. This gives the agent control over perception, motion generation, and recovery at the level where new physical evidence becomes available.

We implement AGP by giving a general purpose agent an interface for observing and controlling the robot (Figure 1). The interface provides the agent with camera images, the robot's state, and action feedback. The interface also lets the agent request observations or control the robot through programs

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.

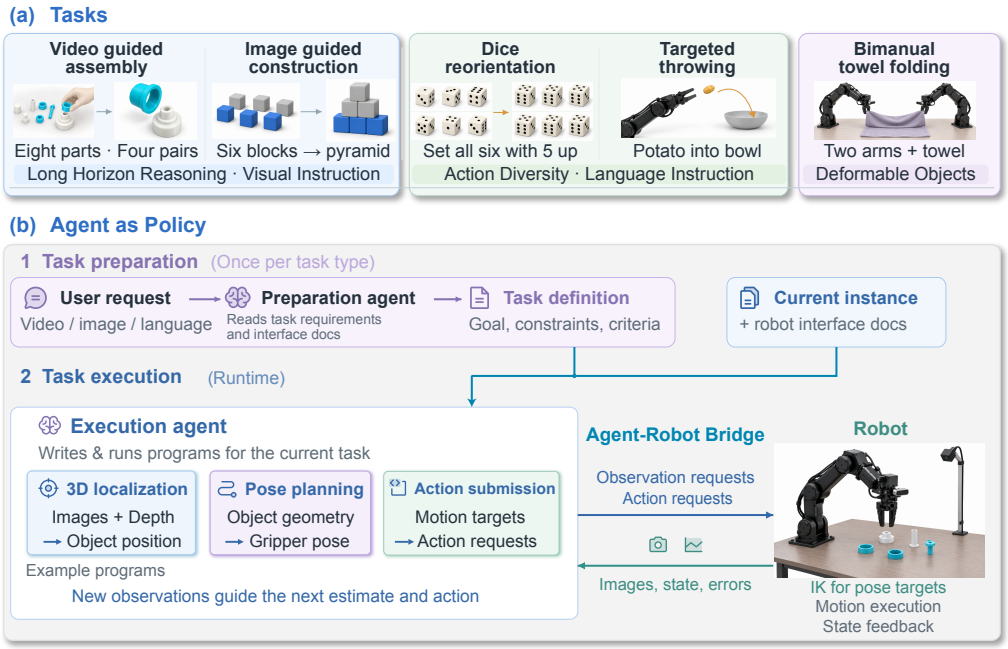


Figure 1: Overview of AGP. (a) Evaluated tasks include video guided assembly, image guided construction, dice flipping, targeted throwing, and bimanual towel folding. (b) A preparation agent creates a reusable task definition. At runtime, the execution agent writes and selects programs to interpret observations, compute motion targets, and request robot actions through the robot interface. After each action, the agent uses motion feedback and new observations to refine its estimates and adjust subsequent actions.

it writes and runs. Given task instructions, the agent is free to plan, reason, and decide what to do next. It autonomously direct task execution without a fixed sequence of steps. Model parameters remain unchanged throughout execution.

We evaluate AGP on real robots across five tasks: part assembly, block construction, dice flipping, targeted throwing, and bimanual towel folding. AGP achieves success rates of at least 80% in seven of eight task configurations, including 100% success in four. It also achieves substantially higher success rates than Graph as Policy (Chen et al., 2026) and ASPIRE (Lu et al., 2026), which use agents to build fixed robot execution pipelines rather than using an agent directly as the robot policy at runtime. We further study experience accumulation and transfer. Experience saved in previous trials improves efficiency in subsequent trials and can be transferred to a weaker model to improve its task success and efficiency. We hope this study provides insights for future research on using agents for more capable and efficient robot manipulation. We release our code at `https://anonymous.4open.science/r/review-artifact-14B2`.

2 Related Work

2.1 Agentic Robot Systems

Agentic robot systems use language models for reasoning, tool use, and adaptation through execution feedback, with two main roles for the agent. **Program synthesis and refinement.** One line generates programs that map observations to actions through perception and control interfaces, with validation supporting execution in simulation and on physical robots (Singh et al., 2022; Liang et al., 2022; Chen et al., 2024; Mu et al., 2024). Recent work uses execution evidence to iteratively refine programs or computation graphs and retain reusable solutions (Lu et al., 2026). AGP keeps a runtime agent in the decision loop during deployment, allowing it to choose new approaches, write new programs, and revise workflows based on observations and execution feedback. **Runtime policy orchestration.** A

second line selects and sequences learned skills according to task goals, affordances, and environment feedback (Ahn et al., 2022; Huang et al., 2022). Recent systems extend this approach to vision language action (VLA) policies with progress monitoring and recovery (Li et al., 2026a; Zhang et al., 2026; Galanti et al., 2026). These systems separate skill selection from motion generation within the invoked policy or primitive. AGP uses the runtime agent itself as the policy without a separately trained action policy, enabling zero-shot task execution in previously unseen environments.

2.2 Robot Manipulation

Classical robot manipulation combines geometric modeling, task and motion planning, and feedback control (Garrett et al., 2021). Learned visuomotor policies predict actions from robot trajectory data, with diffusion models capturing action sequences (Chi et al., 2023). Vision language action models extend this approach to diverse tasks through image and language conditioning (Brohan et al., 2023; Kim et al., 2024; Black et al., 2024). Video and world action models further couple actions with future visual states to inform action generation and evaluation (Li et al., 2025; Ye et al., 2026). AGP introduces an alternative to policies trained specifically for action generation by using a general purpose foundation multimodal large language model (MLLM) directly as the robot policy.

2.3 Agentic Workflows Beyond Robotics

Language agents combine reasoning, tool use, and reflection on interaction feedback (Yao et al., 2022; Schick et al., 2023; Shinn et al., 2023). Coding agents apply these mechanisms to editing and testing software (Yang et al., 2024; Wang et al., 2024), while computer use agents act through browser and desktop interfaces, adapting to interface changes (Zhou et al., 2023; Xie et al., 2024). Multiagent frameworks distribute workflows among specialized roles that communicate and share intermediate results (Wu et al., 2023; Hong et al., 2023). These workflows establish mechanisms for tool use, memory, feedback, and coordination. AGP studies how they shape the robot policy boundary, with agents governing physical task execution through perception, geometry, planning, and control interfaces.

3 A General Purpose Agent as the Robot Policy

3.1 Problem Formulation

Objective. We consider physical manipulation tasks specified by video, image, or language instructions, or a combination of these. A robot operates in a persistent scene. Its objective is to complete the task within fixed budgets for elapsed time, observations, and actions.

Task preparation. When a new task is introduced, a preparation agent is first called to prepare the task context for the execution agent. It collects the user's instructions, available goal images or demonstration videos, robot interface documentation, and existing task prompts. It uses these materials to define the goal and explain how to use the available robot tools. The resulting instructions state the task constraints and completion criteria, point to reference materials, and specify tool access, execution budgets, and logging requirements. They may also include practical guidance from previous runs when available. The execution agent reads the supplied materials and carries out the task. For subsequent instances of an existing task type, a launch program starts the execution agent directly with the saved definition and the current instance details.

Agent as the robot policy. The execution agent is a general purpose agent (e.g., Codex) that serves directly as the robot policy. It receives a task specification g and a documented robot interface I. The task specification includes the instructions collected by the preparation agent and other demonstration materials. The agent maintains a local workspace W with scripts, notes, and observations collected during execution. It interprets the task, plans actions, and interacts with the robot through tools. We describe its tool selection process as

$$u_k \sim \pi_\theta(\cdot \mid g, I, H_k, W_k), \tag{1}$$

where u_k is the tool call selected at step k. Here, H_k contains the conversation and tool results available at that step, and W_k contains the current workspace files. A tool call may gather information or run a program that submits action targets to the robot controller. The model parameters θ remain fixed throughout execution.

3.2 AGENT-ROBOT BRIDGE

The robot interface I supports two-way information exchange between the agent and the robot. The agent sends observation and action requests and receives camera observations, robot state, and action results. The interface is implemented as a Python command-line client and a separate server process. The agent calls the client, which exchanges requests and responses with the server through JSON files in the agent's workspace. The server communicates with the robot and cameras through a separate program that handles hardware access.

Robot to agent. The interface provides two sources of information: camera observations and robot state. An overhead camera provides a view of the scene, and another wrist camera provides close-up images and depth values aligned with the image pixels. Camera calibration data and camera poses allow the agent to convert these visual measurements into positions in the robot's coordinate frame. The interface also reports joint angles measured by the motors, together with the gripper's position and orientation computed from these angles through forward kinematics. After a motion, the interface reports the available updated state and errors relative to the requested target. This feedback helps the agent check whether the robot reached the intended pose. Observations are saved in the agent's workspace so it can view them or process them with programs.

Agent to robot. The agent sends two types of requests through the interface. *Observation requests* ask the robot to capture images or report its state. *Action requests* specify arm movements or gripper opening and closing. To move the arm, the agent specifies either target joint angles or a target position and orientation for the gripper. Separate gripper requests specify how far the fingers should open or close. The robot controller converts these targets into commands that drive the joints and gripper. When the agent specifies a gripper pose, the controller uses Mink (Zakka, 2026) to compute target joint angles through inverse kinematics. It then generates joint trajectories within workspace and motion limits and uses motor feedback to follow them. The robot controller operates independently of agent inference, so the agent can continue reasoning while the robot moves. Between requests, the controller holds the arm in place while the agent reasons about its next step. Each arm executes motion commands in order, while different arms can move at the same time.

3.3 RUNTIME PROGRAMMING AND EXECUTION

Runtime programming. The agent writes programs during execution to interpret observations, compute action targets, and submit them through the robot interface. It can inspect reference videos and camera images and process saved data with Python and available libraries. For example, it can estimate an object's position from camera observations and depth measurements, compute a target gripper pose, and issue the corresponding robot commands. Appendix E analyzes recorded programs for geometric estimation, visual processing, and robot call composition.

Feedback and adaptation. After executing an action, the agent uses motion feedback and new observations to revise its estimates and subsequent actions. For example, an unsuccessful insertion can prompt a new alignment estimate and a modified approach. The agent determines grasp poses, action sequences, observation timing, and recovery during this cycle. It continues until it reports completion or ends the attempt within the task budget. Physical success is assessed against the task criteria using the resulting scene and recorded evidence.

Execution efficiency. We provide instructions to reduce tool overhead. The agent is asked wait longer for an action to finish, so as to avoid generating redundant tool calls. The instructions also encourage combining consecutive operations in one tool call, such as moving the arm, capturing an image, and displaying it.

4 EXPERIMENTS

We evaluate AGP on real-world robot tasks spanning long horizon reasoning, action diversity, and object diversity under video, image, and language instructions.

4.1 Experimental Setup

4.1.1 Robot and Agent

Robot Platform. We use an I2RT YAM (I2RT) arm with six revolute joints and a parallel gripper in a tabletop workspace. AGP provides the robot interface described in Section 3.2. The towel folding task uses two arms with a common coordinate frame and coordination selected by the agent. The targeted throwing task uses an interface with timed motion programs, documented in Appendix A.1.

Cameras. A wrist mounted Intel RealSense D405 provides aligned RGB and depth at 640×360 pixels. A fixed overhead Logitech BRIO provides rectified RGB at 1920×1080 pixels. Both cameras capture at 30 frames per second, and the agent requests observations on demand.

Model Configuration. We use general purpose agents as robot policies in our experiments. These include Codex (GPT-5.6 Sol, Terra, and Luna; GPT-6 Astra) and Claude Code (Claude Opus 5 and Claude Fable 5.1). Model weights remain fixed. Details are in Appendix A.

4.1.2 Tasks and Evaluation

Task Design. We find that conventional "pick and place" tasks, such as placing an object into a bowl, pose little challenge for AGP, motivating more demanding tasks to evaluate its capabilities. Therefore, we use five different task groups. More details are in Appendix B.1.

- **Four pair assembly.** The robot assembles eight parts into four pairs (Figure 9). It follows a human demonstration video and must identify the pairings and order the operations while preserving earlier progress. This task tests *long horizon reasoning* under video instructions. The parts are adapted from the AutoMate dataset (Tang et al., 2024).
- **Block construction.** The robot builds a block structure from a goal image (Figure 10). The targets comprise a pyramid, two towers, and a six-block tower. This task tests *long horizon reasoning* under image instructions. It requires interpreting spatial relationships and ordering placements while preserving earlier progress.
- **Dice flipping.** The robot reorients six randomly placed dice (Figure 11). All six must show the number specified by the language instruction on their upward faces. This task tests *action diversity*: the agent must choose grasps and rotations based on each die's current orientation and account for how it settles after release.
- **Targeted throwing.** The robot throws an object toward a target (Figure 12). The target is specified through language instructions. This task tests *action diversity* because the robot must release the object during a circular swing. The agent must coordinate the swing with gripper opening to send the object toward the target.
- **Towel folding.** The robot folds a towel using two arms (Figures 13 and 14). It follows video instructions and tests *object diversity*: it must coordinate both arms to handle a soft and deformable object.

Trial Protocol & Eval Metrics. Each trial starts from the same predefined scene for its task. We set fixed budgets for elapsed time and for observation and action requests. For each agent, we vary the task order to reduce order bias. The agent may choose to autonomously recover from errors within these budgets. We assess success against predefined criteria using the physical outcomes and execution video. We primarily evaluate AGP by its success rate. We also report completion time, token usage, and model inference cost. More details are in Appendix B.2 and B.3

4.2 Experimental Results

4.2.1 Main Results

Table 1 presents the main results using GPT-6 Astra. We have the following observations.

- **Strong zero-shot performance.** AGP achieves success rates of at least 80% in seven of eight task configurations and succeeds in every trial in four configurations. These results support its effectiveness for zero-shot manipulation across the evaluated tasks.

Table 1: **Main results using GPT-6 Astra as policy.** Success is shown as successful trials over total trials (n/N). Time, tokens, and cost show means with [min, max] below, calculated only on successful trials (except for throwing). † Throwing instead uses all ten trials and measures total session time, including final reporting. Tokens count input and output usage, including reasoning tokens and retries. Cost uses Standard API rates with cached input pricing. Tags denote long horizon reasoning (LHR), action diversity (AD), object diversity (OD), video instruction (Vid), image instruction (Img), and language instruction (Lan).

Task	Capability	Instruction	Success (n/N)	Time ↓ (min)	Tokens ↓ (million)	Cost ↓ (USD)
Four pair assembly	LHR	Vid	8/10	37.2 [16.0, 59.1]	13.02 [3.96, 27.12]	16.62 [5.55, 31.50]
Block construction	LHR	Img	10/10	21.6 [14.9, 32.2]	9.24 [6.27, 12.94]	11.69 [8.01, 15.28]
Pyramid						
Two towers			10/10	20.6 [14.8, 29.2]	7.94 [4.94, 14.39]	9.92 [6.27, 16.85]
Six block tower			9/10	28.2 [11.6, 55.0]	11.57 [4.50, 19.85]	14.93 [6.50, 26.78]
Dice flipping	AD	Lan	10/10	37.9 [22.6, 56.0]	17.08 [10.16, 25.00]	21.07 [12.57, 31.20]
Targeted throwing	AD	Lan	9/10	17.71† [10.8, 28.6]	5.71 [2.58, 14.15]	8.16 [4.10, 18.97]
Potato						
Towel folding	OD	Vid	5/5	50.8 [33.8, 63.3]	18.12 [10.13, 25.26]	24.14 [14.16, 30.90]
Sequential						
Simultaneous			3/5	21.4 [15.7, 31.2]	5.06 [4.27, 5.68]	7.46 [6.16, 8.67]

- **Task complexity and execution overhead.** More complex tasks generally take longer and incur higher inference costs. For example, building the six-block tower takes more time and costs more than building the pyramid or the two towers. Potato throwing takes 17.71 minutes per session on average across ten trials, including trajectory analysis, verification, and final reporting. Appendix D.1 provides a detailed breakdown of execution time.
- **Challenges in deformable object manipulation.** Sequential towel folding succeeds in all five trials but takes 50.8 minutes and costs USD 24.14 on average. Both averages are the highest among the evaluated tasks. Simultaneous folding follows a demonstration in which both short ends fold inward together. It has the lowest observed success rate at 3/5, highlighting the difficulty of coordinating movements when manipulating deformable material.

4.2.2 Agent and Thinking Effort Comparison

Table 2 compares agent and model configurations on a two-pair assembly task based on video demonstration. The task pairs a hexagonal ring with a short post and a circular sleeve with a stepped cylinder. Within Codex, we evaluate GPT-6 Astra at low, medium, and high thinking effort and GPT-5.6 Sol, Terra, and Luna at high thinking effort.

The comparison also includes Claude Code with Claude Opus 5 and Claude Fable 5.1, both at high effort. Task references, robot functions, and trial budgets are shared across configurations.

The results show that AGP can complete the assembly task with both Codex and Claude Code. However, performance differences among models show that model choice remains important. Astra achieves 100% success at all tested thinking effort levels, showing consistently strong performance on this task.

GPT-5.6 Sol has lower inference costs than Astra but takes longer, while Terra and Luna struggle to complete the task. Within Claude Code, Fable has a lower success rate than Opus. On successful trials, it uses fewer tokens but takes longer and costs more.

4.2.3 BASELINE COMPARISON

To the best of our knowledge, AGP is the first to use a general purpose runtime agent itself as the robot policy for zero-shot manipulation in previously unseen environments. Given the lack of a directly matching baseline, we compare with the broader class of agentic robot programming methods. We select Graph-as-Policy (GaP) (Chen et al., 2026) and ASPIRE (Lu et al., 2026), which share AGP's training free formulation with fixed model weights, providing a common basis for comparison. GaP uses multiple agents to compose robotic skills into an executable computation graph and refines its structure and parameters using simulation feedback. ASPIRE iteratively debugs robot programs using detailed execution traces and evolutionary search, consolidating validated repairs into a reusable skill library. We evaluate three relatively easy tasks from our suite, namely two pair assembly, four pair assembly, and pyramid block construction.

Table 2: **Agent and thinking effort comparison on a two-pair assembly task.** Time, tokens, and cost show means with [min, max] below, calculated only on successful trials. Models other than GPT-6 Astra uses high thinking effort.

Model	Success (n/N)	Time (min)	Tokens (million)	Cost (USD)
Codex				
GPT-6 Astra				
Low effort	5/5	9.9 [6.0, 19.0]	3.69 [1.44, 7.72]	4.79 [2.15, 9.56]
Medium effort	5/5	9.2 [6.3, 15.4]	3.03 [1.84, 5.80]	4.09 [2.76, 7.48]
High effort	5/5	9.2 [7.9, 10.6]	3.30 [2.58, 4.02]	4.47 [3.72, 5.19]
GPT-5.6 Sol	5/5	14.2 [9.1, 19.3]	7.13 [4.12, 9.72]	3.94 [2.40, 5.30]
GPT-5.6 Terra	1/5	19.0 [19.0, 19.0]	6.28 [6.28, 6.28]	1.98 [1.98, 1.98]
GPT-5.6 Luna	0/5	–	–	–
Claude Code				
Claude Opus 5	5/5	22.2 [14.4, 31.9]	12.40 [9.05, 16.73]	9.75 [6.94, 13.03]
Claude Fable 5.1	3/5	27.7 [18.3, 39.8]	9.27 [6.87, 11.10]	11.30 [8.33, 14.36]

Both baselines require iterative refinement in simulation before real robot execution. We therefore build MuJoCo environments matching the real experimental setup and allow both methods to complete their required construction and refinement procedures. AGP executes zero-shot on the physical robot. All three methods use the same underlying agent, GPT-6 Astra with high thinking effort. For ASPIRE, Table 3 reports real only / sim + real time and cost. Real only covers successful program execution, while sim + real adds the full shared construction and refinement overhead in simulation and on the real robot to each successful run, without amortization. AGP time covers task delivery through final physical verification, including reasoning, programming, execution, and recovery. Accounting details are in Appendix J.

Table 3: **Baseline comparison.** Time and cost average successful runs. ASPIRE reports real only / sim + real, with full construction and refinement overhead in the latter. Execution makes no coding model calls, so its token cost is zero. Green/red marks success above/below 50%.

Task	Method	Success ↑ (n/N)	Time ↓ (min)	Cost ↓ (USD)
Two pair assembly	GaP	0/5	–	–
	ASPIRE	2/5	2.9 / 64.8	0.00 / 18.90
	AGP	5/5	9.2	4.47
Four pair assembly	GaP	0/10	–	–
	ASPIRE	0/10	–	–
	AGP	8/10	37.2	16.62
Pyramid construction	GaP	0/10	–	–
	ASPIRE	1/10	9.2 / 271.7	0.00 / 78.13
	AGP	10/10	21.6	11.69

Task Success. Across the three tasks in Table 3, GaP fails in all 25 trials, and ASPIRE succeeds in only 3 of 25 trials, while AGP fails in only 2 of its 25 trials. A likely reason is the gap between simulation and reality. Contact dynamics and friction can differ, while lighting, shadows, and depth noise make real point clouds harder to interpret, impairing perception and execution after transfer. Such variation can disrupt the predefined programs used by GaP and ASPIRE, while AGP's runtime agent adjusts execution and repairs failures using real time feedback. **Efficiency.** Despite ASPIRE's shorter execution time on successful two-pair assembly runs, AGP requires substantially less total time and inference cost when preparation is included. Charging the full preparation overhead to each successful run, ASPIRE takes about seven times AGP's time and incurs 4.2 times its inference cost, with preparation accounting for roughly 95% of its total time. This preparation overhead limits

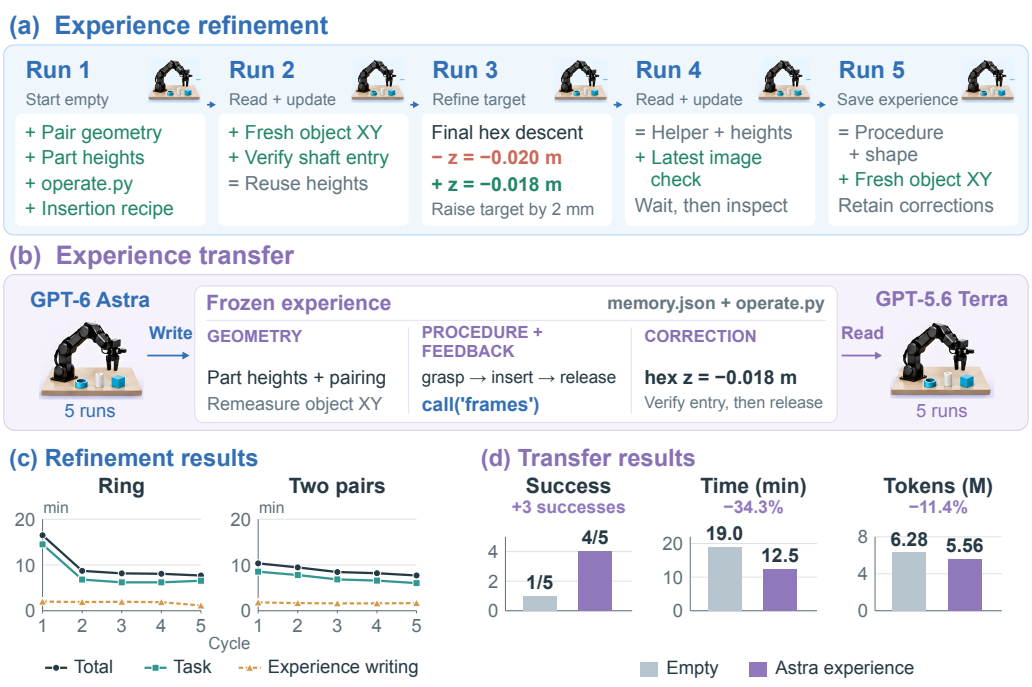


Figure 2: **Experience refinement and transfer.** (a) Experience updates across repeated runs. (b) Transfer of frozen experience from Astra to Terra. (c, d) Experimental results for refinement and transfer, respectively.

ASPIRE in zero-shot settings that require immediate execution of unseen tasks, whereas AGP can begin from current observations and adapt during execution.

4.3 Improving Efficiency through Experience

As shown in previous sections, AGP can be slow sometimes due to long thinking times. We study two ways to improve execution efficiency using learned experience, as shown in Figure 2. In *experience refinement*, the same agent repeats a task and updates the experience (recorded in documents) after each run. The next run uses the learned experience as context. In *experience transfer*, a stronger model first performs a task and records its experience. A weaker model then uses these documents to perform the same task.

Experience Refinement. We test GPT-6 Astra on two tasks. The first involves disassembling and reassembling a ridged ring and its base, and the second is the same two-pair assembly as Section 4.2. Each task is repeated five times: the first run starts without prior experience; later runs read and update the saved documents. These documents record grasp and insertion settings that worked, corrections from earlier runs, and related processing scripts. For example, Figure 2(a) illustrates how two-pair assembly runs reuse saved part heights and an operation script while refreshing object positions and refining the final insertion height based on execution feedback. Figure 2(c) shows that both tasks become faster over repeated runs. The ring task improves most after the first run, while two-pair assembly improves more gradually. Total runtime decreases even when considering experience writing time.

Experience Transfer. We transfer experience from Astra's five two-pair assembly runs to GPT-5.6 Terra. Terra performs five trials with Astra's saved experience and five without it. Every trial with experience reads the same documents without updating them. Figure 2(d) shows that success increases from 1/5 to 4/5 after seeing prior experience. Among successful trials, mean completion time falls from 19.0 to 12.5 minutes (34.3%), and mean token usage falls from 6.28M to 5.56M (11.4%).

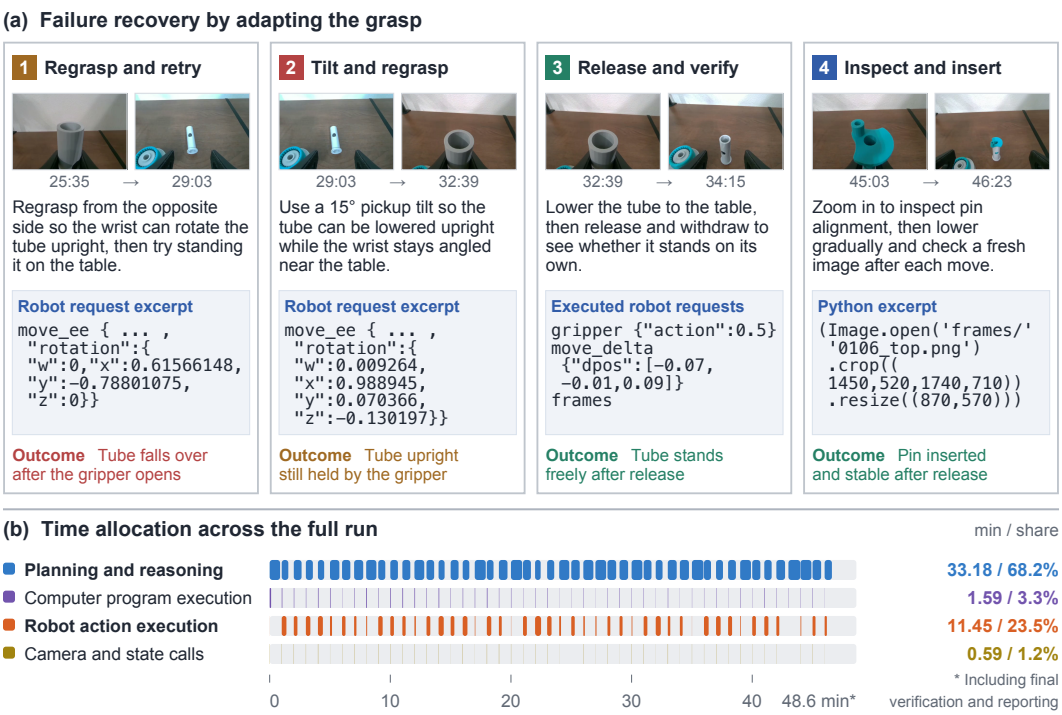


Figure 3: **Failure recovery during four pair assembly.** (a) After the tube falls over following an attempt to stand it upright, the agent reasons about a different grasp and introduces a tilted pickup that allows the robot to stand the tube upright successfully. Timestamps indicate elapsed time within the run. (b) Time allocation among planning and reasoning, computer program execution, robot action execution, and camera and state calls across the full run.

These experiments prove that past experience can help models complete tasks more efficiently and can be transferred across models. Appendices D.2 and D.3 provide the detailed setups and analyses for experience refinement and transfer, respectively.

4.4 Case Study of Failure Recovery

We examine a four pair assembly run to illustrate how AGP uses observations to revise its actions during failure recovery. Figure 3(a) links selected recovery steps to camera images, robot requests, and a Python excerpt. After a regrasp from the opposite side leaves the tube falling over upon release, the agent introduces a 15° pickup tilt so it can lower the tube upright while keeping the wrist angled near the table. It then releases the tube and withdraws the gripper to verify that the tube stands on its own. During the subsequent insertion, the agent crops and enlarges an image to inspect pin alignment, lowers the pin gradually, and checks fresh images until the assembly is stable after release. This case illustrates how visual feedback informs changes to grasp geometry and subsequent robot commands. Figure 3(b) separates planning and reasoning, computer program execution, robot actions, and camera and state calls across the full 48.6 minute run. Planning and reasoning account for 68.2% of this time, while robot actions account for 23.5%.

5 Conclusion

We introduce AGP, which uses a general purpose agent as the robot policy through runtime programming and physical feedback. In real robot experiments, AGP achieves success rates of at least 80% in seven of eight task configurations, and significantly outperforms previous agentic robot systems. Tests with Codex and Claude Code show successful task execution across multiple models. Reusing experience saved in documents improves execution efficiency, and transferring this experience to a weaker model improves task success while reducing time and token usage on successful trials.

AI USE STATEMENT

During the preparation of this manuscript, the authors used generative AI tools to assist with drafting and language editing, including refining sentence structure and improving clarity and readability. The authors are responsible for reviewing and revising the resulting text to ensure that it accurately represents the research. The authors take full responsibility for the content of the manuscript, including the methodology, experimental results, interpretations, and conclusions.

REPRODUCIBILITY STATEMENT

Section 3 describes the agent formulation, robot interface, and runtime execution procedure. Section 4.1 summarizes the experimental setup, with robot hardware, control parameters, and agent configurations detailed in Appendix A. Appendix B describes the tasks, reference materials, success criteria, trial protocol, and metric definitions. The assembly parts are adapted from the AutoMate dataset (Tang et al., 2024), as described in Appendix B.1. Appendix C describes the supplementary coding agent comparison on pyramid construction and its usage and cost accounting. Appendix D documents the execution time analysis and repeated execution study, including experience checkpoints and timing boundaries. Appendix I presents visual references and representative execution sequences.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Anthropic. Claude Code model configuration. `https://code.claude.com/docs/en/model-config`. Accessed September 8, 2026.

Kevin Black, Noah Brown, Danny Driess, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Junting Chen, Yao Mu, Qiaojun Yu, Tianming Wei, Silang Wu, Zhecheng Yuan, Zhixuan Liang, Chao Yang, Kaipeng Zhang, Wenqi Shao, Yu Qiao, Huazhe Xu, Mingyu Ding, and Ping Luo. RoboScript: Code generation for free-form manipulation tasks across real and simulation. *arXiv preprint arXiv:2402.14623*, 2024.

Kaiyuan Chen, Shuangyu Xie, Letian Fu, Justin Yu, William Pacini, Sandeep Bajamahal, Hudson Kim, Jaimyn Drake, Daehwa Kim, Haoru Xue, Jonathan Francis, Christian Juette, Peter Schaldenbrand, Muhammet Yunus Seker, Ruwan Wickramarachchi, Uksang Yoo, Guanzhi Wang, Adithyavairavan Murali, Balakumar Sundaralingam, S. Shankar Sastry, Spencer Huang, Yuke Zhu, Linxi Fan, and Ken Goldberg. GaP: A graph-as-policy multi-agent self-learning harness for variational automation tasks. *arXiv preprint arXiv:2607.05369*, 2026.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

Liane Galanti, Dhruv Shah, and Tri Dao. Addressing the orchestration gap in generalist robots via physical agency. *arXiv preprint arXiv:2607.21725*, 2026.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:265–293, 2021. doi: 10.1146/annurev-control-091420-084139.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

Wenlong Huang, Fei Xia, Ted Xiao, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

I2RT. I2RT robot software and YAM platform. `https://github.com/i2rt-robotics/i2rt`. Accessed September 8, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, et al. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Ruiying Li, Yunlang Zhou, YuYao Zhu, Kylin Chen, Jingyuan Wang, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, Jiayao Ma, Xin He, Yongjian Shen, Yang Yang, Guanghui Ren, Maoqing Yao, Wenhao Wang, and Yao Mu. RoboClaw: An agentic framework for scalable long-horizon robotic tasks. *arXiv preprint arXiv:2603.11558*, 2026a.

Shuang Li, Yihuai Gao, Dorsa Sadigh, and Shuran Song. Unified video action model. *arXiv preprint arXiv:2503.00200*, 2025.

Zongxia Li, Xiyang Wu, Hongyang Du, Fuxiao Liu, Huy Nghiem, and Guangyao Shi. A survey of state of the art large vision language models: Alignment, benchmark, evaluations and challenges. *arXiv preprint arXiv:2501.02189v7*, 2026b. URL `https://arxiv.org/abs/2501.02189v7`. Revised August 22, 2026.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi Fan, and Guanzhi Wang. ASPIRE: Agentic /skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026.

Yao Mu, Junting Chen, Qing-Long Zhang, Shoufa Chen, Qiaojun Yu, Chongjian Ge, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, Peize Sun, Haibao Yu, Chao Yang, Wenqi Shao, Wenhai Wang, Jifeng Dai, Yu Qiao, Mingyu Ding, and Ping Luo. RoboCodeX: Multimodal code generation for robotic behavior synthesis. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 36434–36454. PMLR, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

Bingjie Tang, Iretiayo Akinola, Jie Xu, Bowen Wen, Ankur Handa, Karl Van Wyk, Dieter Fox, Gaurav S. Sukhatme, Fabio Ramos, and Yashraj Narang. AutoMate: Specialist and generalist assembly policies over diverse geometries. *arXiv preprint arXiv:2407.08028*, 2024.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Kevin Zakka. Mink: Python inverse kinematics based on MuJoCo. `https://github.com/kevinzakka/mink`, 2026.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Yi Nie, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Jiaji Rao, Zhengru Fang, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering frozen VLAs into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Contents of Appendix

A Implementation Details **14**

A.1 Robot Platform and Control Interface 14

A.2 Agent Configuration and Execution Conditions 15

A.3 Model and Reasoning Effort Configurations 15

B Tasks and Evaluation Protocol **15**

B.1 Task Definitions and Success Criteria 15

B.2 Trial Protocol 16

B.3 Metric Definitions and Resource Accounting 16

C Supplementary Coding Agent Comparison on Pyramid Construction **17**

D Efficiency and Experience Reuse Analysis **18**

D.1 Task Execution Time Analysis 18

D.2 Repeated Execution with Persistent Experience 18

D.3 Transferring Frozen Experience to Terra 20

E Runtime Program Analysis **20**

F Error and Recovery Study **22**

F.1 Six Block Tower Collapse During Withdrawal 22

F.2 Grasp Loss and Unstable Placement in Four Pair Assembly 22

F.3 Insertion Misalignment and Contact During Recovery 22

F.4 Recovery Near the Workspace Boundary 22

F.5 Fabric Slippage and Incomplete Fold Verification 23

F.6 Throwing Shortfall and Bowl Displacement During Recovery 23

G Broader Impact **23**

H Execution Across Arm and End Effector Embodiments **23**

I Qualitative Task Examples **24**

I.1 Four Pair Assembly 24

I.2 Block Construction 24

I.3 Dice Flipping 24

I.4 Targeted Throwing 26

I.5 Bimanual Towel Folding 26

J Baseline Resource Accounting **26**

A Implementation Details

A.1 Robot Platform and Control Interface

Figure 4 shows the platform described in Section 4.1.1.

Robot platform. The YAM arm uses a linear_4310 parallel gripper with a nominal maximum inner jaw opening of 95.5 mm. The i2rt Python library supplies robot models, motor communication, and gravity compensation over a 1 Mbit/s CAN connection. Bimanual Cartesian poses share a coordinate frame through a calibrated rigid transform.

Observations and geometric grounding. Captures from the cameras specified in Section 4.1.1 include calibration, timestamps, and robot state. Wrist depth maps image pixels into the arm base frame. Overhead pixels are located by intersecting camera rays with a plane at an estimated height. The agent can request additional views to refine these estimates. Camera stream rates are independent of the agent's decision rate.

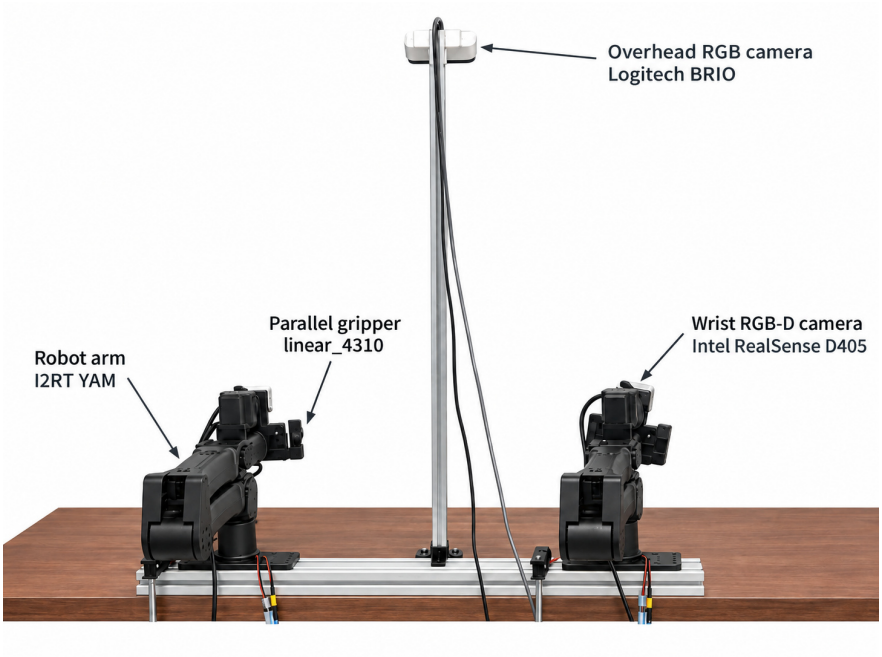


Figure 4: **Robot platform and camera placement.** The setup contains two I2RT YAM arms mounted on a shared tabletop rail. The annotations identify an arm, a linear_4310 parallel gripper, a wrist mounted Intel RealSense D405 RGB and depth camera, and the fixed overhead Logitech BRIO RGB camera. The two arms support bimanual towel folding.

Action interface. The robot interface provides seven commands. `state` returns joint positions, the end effector pose, gripper opening fraction and width, and robot health information. `status` reports the operating status of the robot interface, request budget usage, and coordinate frame information. `help` lists the available commands and their arguments. `frames` captures images from the requested cameras, optionally saves wrist depth, and returns paths to the saved observations and calibration data.

`move_ee` requests a target end effector position and quaternion orientation in the shared coordinate frame and returns the achieved pose and target error, or failure information. `move_joints` requests target joint angles in radians and returns the resulting joint positions and end effector pose. `gripper` requests opening, closing, or a normalized opening fraction between 0 and 1 and returns the resulting opening fraction and width. The agent computes geometry from saved images, depth, and calibration in local programs. Cartesian motion invokes the underlying inverse kinematics and trajectory

execution software. Ordinary motion uses joint velocity and acceleration limits of $20°$/s and $40°/s^2$, a Cartesian translation limit of 0.03 m/s, and an orientation limit of $10°$/s. The gripper target changes at 0.25 of its normalized travel per second. The session server enforces a radial target envelope of 0.12 to 0.65 m, a grasp point height envelope of -0.050 to 0.60 m, and a maximum Cartesian step of 0.25 m. For targeted throwing, the recorded case uses a task specific runtime with timed motion programs to coordinate the swing and gripper release. Each program specifies joint offsets at time knots and scheduled gripper targets. The executor interpolates commands on a 50 Hz grid and records measured joint position, velocity, effort, and dispatch timing. Within these programs, joint 4 uses velocity and acceleration limits of $180°$/s and $360°/s^2$.

A.2 Agent Configuration and Execution Conditions

Trials use the inputs and workspace defined in Section 3.1. Standard trials start with empty task experience. Experience reuse conditions supply the saved files specified in Section 4.3. Each trial records model identifiers, agent software versions, host specifications, budgets, and inference settings.

The execution efficiency instructions in Section 3.3 are held fixed within each condition. Agent traces record waiting intervals, intermediate status checks, and operations grouped within each tool call.

A.3 Model and Reasoning Effort Configurations

The configurations are listed in Table 2. The Astra effort sweep holds the model version, agent environment, task inputs, and robot interface fixed. Comparisons across Codex and Claude Code evaluate complete agent systems under shared scenes, references, and budgets. Claude Code model and effort settings use its native controls (Anthropic). Claude Opus 5 uses identifier claude-opus-5. Each trial records the requested and resolved model identifiers and effective effort setting.

B Tasks and Evaluation Protocol

B.1 Task Definitions and Success Criteria

Table 4 defines the tasks, target capabilities, instruction types, and success criteria used in the main evaluation. Controller arrival tolerances are reported separately from task success tolerances.

Assembly from video. The AutoMate parts (Tang et al., 2024) form a ridged sleeve and broad round base, a hexagonal nut and short post, a smooth sleeve and stepped cylinder, and a long pin and hollow tube, in demonstration order. All methods receive the same human video, extracted frames, and timestamped textual guide. Figure 9 shows an execution.

Assembly part dimensions. Table 5 reports the nominal printing dimensions and mating clearances of the AutoMate meshes, scaled with their aspect ratios preserved.[1] Successful assembly with submillimeter nominal radial clearances illustrates precise alignment (Figure 9). Physical insertion tolerance also depends on printing accuracy and angular alignment.

Block construction from images. Initial cube poses vary across trials while the goal photograph remains fixed within each configuration. Block sets and placement tolerances are fixed before evaluation. The three configurations appear in Figure 10.

Action diversity. Dice flipping uses distinct initial and target faces balanced across predefined conditions. Grasp selection, rotation, and regrasping are annotated from video. For throwing, object and target geometry, initial arrangement, and permitted swing conditions are fixed before evaluation. Landing error is measured relative to the target center. Execution records and video establish joint motion, gripper opening, and object separation.

[1]The plug and socket meshes are `asset_plug.obj` and `asset_socket.obj` in each official AutoMate asset directory.

Table 4: **Task definitions and success criteria.** Capability identifies the primary manipulation challenge, and Instruction type identifies how the task goal or procedure is communicated. The final column specifies the required behavior and physical outcome. Success is assessed from the final state and execution video using criteria fixed before evaluation. Required assembly, support, orientation, and fold relationships must remain stable after gripper release and arm withdrawal.

Task	Capability	Instruction type	Task requirements and success criteria
Four pair assembly	Long horizon reasoning	Video	Assemble eight parts into four pairs in the demonstrated order, inferring mating relationships and insertion motions from the video and current scene. Success requires all four pairs to remain assembled, preserving completed pairs throughout subsequent operations.
Block construction	Long horizon reasoning	Image	Arrange six cubes into a pyramid, two towers, or a six block tower matching the goal image. Infer object identities, orientations, and support relations, and choose a placement order that maintains intermediate stability. Success requires the complete target structure.
Dice flipping	Action diversity	Language	Reorient six randomly placed dice to the requested upward face through grasp selection and rotation. Success requires all six dice to show the requested number after placement.
Targeted throwing	Action diversity	Language	Coordinate a circular joint swing with gripper release to throw an object toward the specified target. Success requires the object to separate from the moving hand, travel through the air, and come to rest within the target region.
Towel folding	Object diversity	Video	Use both arms to reproduce the demonstrated folds. Fold the near long edge onto the far edge, then fold both short ends inward, either right then left (Sequential) or beginning together (Simultaneous). Success requires both inward folds with reasonably aligned edges. Small wrinkles and creases are permitted, and the final table position is unrestricted.

Bimanual towel folding. Each configuration provides an overhead demonstration video and images of the initial and final towel states. Trials begin with the towel approximately matching the demonstrated initial arrangement. The agent may stage placements to maintain arm clearance. Final folds are assessed from overhead images after release and withdrawal. Figures 13 and 14 show the two configurations.

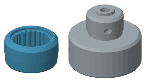
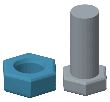
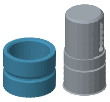
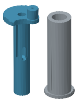
B.2 Trial Protocol

Trials follow Section 4.1.2. The records distinguish physical failure, budget exhaustion, human intervention, and infrastructure interruption under a predefined inclusion policy. Geometric queries and local computation are logged separately from budgeted observation and action requests.

B.3 Metric Definitions and Resource Accounting

Success and time. Success rate is $100s/n$, where s is the number of completely successful trials among n evaluated trials. The task criteria are specified in Table 4. Completion time, token usage, and cost use the same successful trials, with sample counts and summary statistics reported in Tables 1 and 2. A configuration with no successes has undefined successful trial resource means.

Table 5: **Printing dimensions and mating clearances.** Assembly identifies each AutoMate part pair in demonstration order, with plug and socket meshes shown in blue and gray, respectively. Plug overall size and Socket overall size give the respective mesh dimensions as $x \times y \times z$ in the original mesh frame. Hole D_h is the effective bore diameter, measured by the largest cylinder that fits through the bore, including its narrowest ridges. Shaft D_s is the effective shaft diameter, measured by the smallest cylinder enclosing the mating shaft segment. The diametral clearance is $\Delta = D_h - D_s$, and the conservative radial clearance $c = \Delta/2$ gives the available transverse offset for parallel mating axes. All dimensions and clearances are in millimeters. The hole is in the plug mesh for the first three pairs and in the socket mesh for `00446`. Clearances are computed before rounding.

Assembly	Plug overall size	Socket overall size	Hole D_h	Shaft D_s	Δ	c
 `00081`	$52.95 \times 52.98 \times 27.53$	$71.59 \times 71.65 \times 58.26$	37.41	35.86	1.56	0.78
 `00581`	$55.58 \times 48.32 \times 19.46$	$45.52 \times 39.42 \times 78.83$	28.18	25.80	2.38	1.19
 `00007`	$51.08 \times 50.94 \times 39.60$	$42.42 \times 42.30 \times 81.44$	42.86	37.20	5.66	2.83
 `00446`	$39.25 \times 39.16 \times 105.04$	$31.32 \times 31.32 \times 90.81$	18.01	16.54	1.47	0.73

Completion time follows the task delivery to final physical verification boundary in Section 4.1.2. It includes program construction, observation, computation, motion, and recovery. The broader timing analysis in Appendix D.1 also includes unsuccessful trials and terminal reporting.

Interaction accounting. The index k in Equation 1 counts agent tool decisions. The interaction budget counts individual observation and action requests, including those grouped within a single tool call.

Token usage. Total tokens sum input and output usage across all model requests in a trial, including retries. Repeated context counts on each request, with cached input counted once within that request's input total. Provider records are normalized so that reasoning contributes once to output. Table entries report mean total tokens per successful trial in millions.

C Supplementary Coding Agent Comparison on Pyramid Construction

Table 6 extends the agent comparison to pyramid construction using GPT-5.6 Sol and GPT-6 Astra within Codex at high thinking effort. The task follows Table 4. Sol's unsuccessful sessions comprise two supervisor interventions with incomplete structures and one robot bridge disconnection before physical execution.

Usage and pricing. Inference costs are estimated using API prices retrieved on September 9, 2026, with token usage defined in Appendix B.3.

Interpretation. Sol uses more tokens per successful session and has lower estimated inference cost than Astra.

Table 6: **Supplementary coding agent comparison on pyramid construction.** Both Codex models use high thinking effort, extending the comparison in Table 2. Successes count completed pyramids recorded in the session reports. Interrupted sessions count as unsuccessful. Tokens and cost are means over successful sessions for each model. Total cost covers all ten sessions for each model. Cost is estimated using API prices for each model.

Model	Successes	Tokens (million)	Cost (USD)	Total (USD)
GPT-5.6 Sol	7/10	18.86	9.22	87.25
GPT-6 Astra	10/10	9.24	11.69	116.88

D Efficiency and Experience Reuse Analysis

D.1 Task Execution Time Analysis

Figure 5 reports time allocation for the seven task configurations with complete timing decompositions in Table 1. Each panel includes all evaluated trials for its configuration, including failures. Timing runs from task delivery to agent completion or timeout and includes final reporting. For tasks other than throwing, the main table instead summarizes successful trials through physical verification.

Time categories. The visual loop combines image capture, geometric queries, image processing, and the model response following an image. That response can include planning and code generation. Planning and reasoning time covers intervals for planning, reasoning, code generation, and associated response delays outside the visual loop and other measured activities. Local tools covers file access and local computation. Action service covers robot request processing, including motion planning, movement, settling, and rejected requests. Check covers final verification after the last action, and Report covers writing the final result through agent completion. Overlapping activities, including simultaneous arm movements, count once. These categories describe observable activity, with visual reasoning and programming measured together.

Observed costs. Figure 5 shows that observing the scene and deciding how to act account for substantial execution time. The balance varies by task. Assembly and towel folding devote a large share to the visual loop. Simultaneous folding has shorter total time than sequential folding, with less time spent in both the visual loop and robot execution. Task strategy and trial outcomes also differ, so these timings alone do not isolate the benefit of parallel arm motion.

D.2 Repeated Execution with Persistent Experience

Task protocols. Each task in Section 4.3 is repeated five times using GPT-6 Astra at high thinking effort. The first execution starts with empty experience. Each later execution uses a fresh agent context with saved measurements, programs, and lessons from earlier attempts. The agent updates object positions from current images.

For the ring task, each cycle starts assembled, removes the ring onto the table, and reassembles it on the base. All cycles succeed after release and withdrawal, with video and action logs documenting execution. The two pair task assembles a hexagonal ring onto a narrow post and a circular collar onto a stepped cylinder. The agent reports stable assemblies in every trial based on recorded images.

The ring task has one failed physical stage attempt in cycle 1 and none in later cycles; two pair assembly has none in any cycle. Motion and gripper requests / image capture requests are 37 / 37 in the first ring cycle and 32 / 33 in each later cycle, and 25 / 27 in the first two pair cycle and 23 / 25 in each later cycle. These counts span task start through experience saving.

Time measurements. Task time runs from the first task start marker to successful assembly verification, including the return to the observation posture for two pair assembly. Experience writing covers the subsequent saving of updated experience. Within task time, Tool execution measures recorded time spent running programs, processing images, and operating the robot. Simultaneous activities count once. Time when tools run while the model responds is assigned to Tool execution. The remaining task time estimates Reasoning and programming, including planning, code generation,

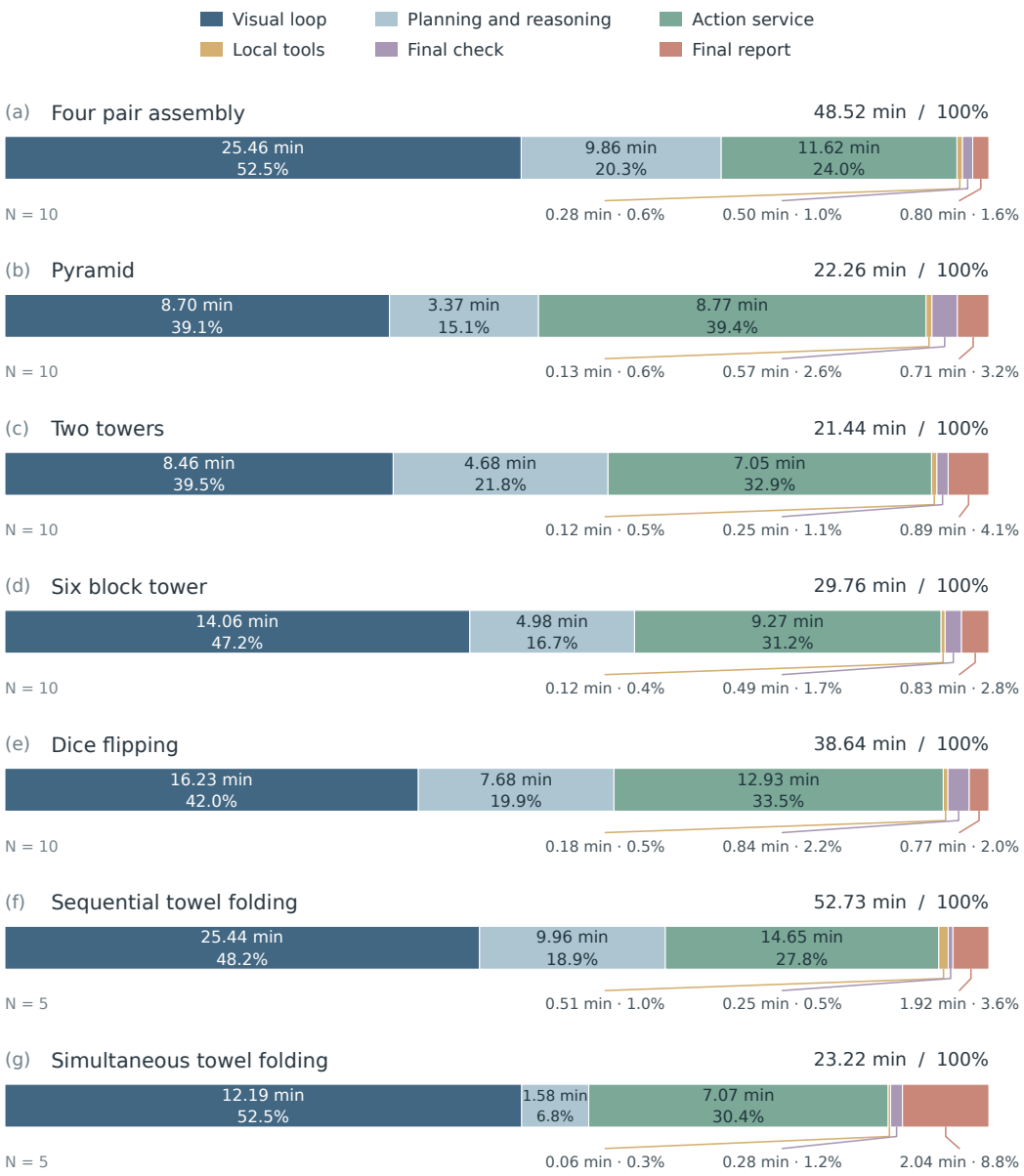


Figure 5: **Time allocation across manipulation tasks.** Seven task panels show mean time allocation over all N evaluated trials, including failures. Every bar spans 100% of its task's total session time, including final reporting. Each component is labeled with its mean minutes and percentage of the total. The seven tasks have complete timing decompositions. Throwing is omitted because its component timings are unavailable. Displayed values are rounded.

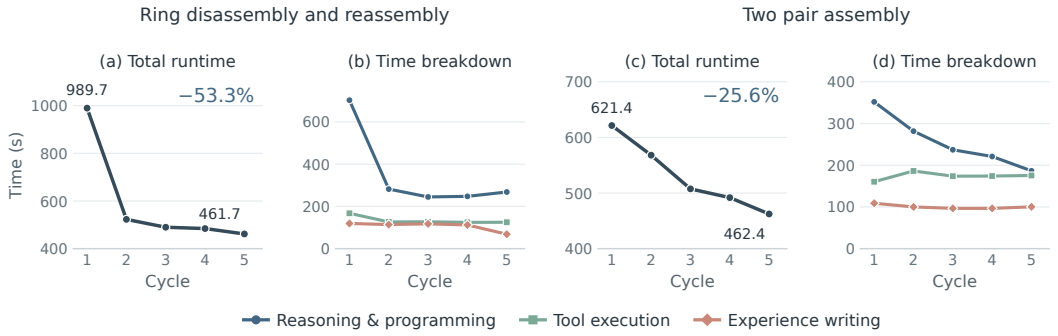


Figure 6: **Execution time over repeated tasks.** Total runtime and its three components across five cycles of ring disassembly and reassembly (a–b) and two pair assembly (c–d). Times are seconds; the components sum to total runtime up to rounding. Percentages indicate the total runtime reduction from cycle 1 to cycle 5. Vertical scales differ across panels.

and delays in receiving model responses or starting tools. This definition makes the components additive while treating model activity as a combined estimate.

What experience changes. Figure 6 shows that the main time savings occur in reasoning and programming. For two pair assembly, this time decreases while tool execution takes slightly longer. For the ring task, the largest improvement follows the first cycle, when the agent establishes an alignment procedure and recovers from a failed insertion.

The logs show how later agents use the saved experience. They call existing measurement and motion programs, update object coordinates in earlier scripts, and reuse corrected grasp and insertion procedures. These observations link shorter response times with reuse of programs and task knowledge. The trials form an ordered sequence with changing object positions, so a comparison that starts every trial with empty experience is needed to isolate the effect of experience reuse.

D.3 Transferring Frozen Experience to Terra

The transfer study in Section 4.3 gives every Terra trial the same frozen Astra experience files. Operator review confirms successful assembly in four trials. In the unsuccessful trial, the circular pair is completed, but the hexagonal collar remains separate and the narrow post tips during recovery.

The empty experience comparison uses the Terra trials in Table 2. Time and resource accounting follow that table. Completion is marked by the final result update after motion execution, or the recorded task completion time when no subsequent update exists. Costs are estimated using API prices for Terra inference. The resource means compare one successful trial without experience with four using experience, so the observed reductions rest on small and unequal samples.

E Runtime Program Analysis

We analyze the programs constructed during 65 robot task executions to characterize the runtime programming described in Section 3.3.

Program inventory. The session workspaces retain 37 Python working files across 18 sessions. Agent event logs identify their creation or renaming during execution. Table 7 groups these files by their primary purpose. We count each retained working path once within its session. The inventory separately identifies 63 Python file copies supplied through inherited knowledge and seven copies exported for subsequent reuse. These copies are excluded from the 37 working files.

The logs also contain 135 shell command events with inline Python across 48 sessions, including all 18 sessions with saved working files. These events contain image processing and geometric computation as well as dependency checks, report writing, and file verification. An event can

Table 7: **Programs constructed during execution.** Primary purposes of the 37 retained Python working files. Image crops, demonstration montages, and some object detection routines also appear as inline code and are outside this file inventory.

Primary purpose	Files	Observed computations and operations
Perception and geometry	29	Depth measurement, color region selection, ray intersection, triangulation, surface fitting, and grasp offset estimation.
Robot call composition	7	Pose and gripper commands followed by capture, concurrent calls to different arms, compact responses, and full response logging.
Visual annotation	1	Projection of spatial reference points into camera images for inspection.

combine several operations or create a saved program, so event and file counts are reported separately. The inventory reflects the available interfaces, which provide either geometric queries or images, depth, and calibration for local computation.

Surface fitting and grasp offset compensation. In a four pair assembly execution, the agent creates `fit_head.py` to estimate the pose of a held pin from a wrist depth image. It samples triples of points to identify a planar surface, then uses singular value decomposition to estimate its normal. The following excerpt refines the plane estimate from the selected surface points.

```
ctr = best.mean(0)
_, _, v = np.linalg.svd(best - ctr)
normal = v[-1]
normal *= np.sign(normal[2])
```

The agent extends the program to intersect an overhead pixel ray with this plane, estimate the pin tip using an assumed pin length, and express the tip offset in the gripper frame. A further extension computes a corrected target orientation and position. The resulting horizontal target is approximately $(0.51325, -0.15820)$ m. The next motion request uses $(0.5133, -0.1582)$ m and the computed quaternion, establishing a direct link between the program output and physical action selection. That request returns a target arrival error. The agent then adjusts the requested position and continues the insertion sequence. This trace illustrates successive program extensions and the use of their outputs within a feedback loop.

Triangulation from two camera views. In another four pair assembly execution, the agent creates `triangulate.py`. It converts selected wrist and overhead pixels into calibrated rays in the robot base frame. Given ray origins `t1` and `t2` and unit directions `d1` and `d2`, the program computes the closest points on the two rays using the following operations.

```
ab = np.linalg.lstsq(
    np.stack([d1, -d2], 1), t2 - t1, rcond=None
)[0]
p1 = t1 + ab[0] * d1
p2 = t2 + ab[1] * d2
```

It returns their midpoint as the position estimate and their separation as a consistency measure. The agent subsequently calls the same program on additional targets. Pixel correspondences are supplied by the agent, and the ray separation measures agreement between those inputs under the saved calibration. Position accuracy also depends on correspondence and calibration error.

Visual processing and execution helpers. Other executions contain inline programs for color segmentation, circle detection with multiple thresholds, and demonstration frame montages. During bimanual towel folding, the agent creates `step.py` to read both arm states, dispatch one operation per arm concurrently, and capture observations after the operations. It also retains full responses in a local log while printing compact results. This helper combines the interface operations described in Section 3.2. The cases document program construction and use, while controlled comparisons would be needed to measure their contribution to task performance.

F Error and Recovery Study

F.1 Six Block Tower Collapse During Withdrawal

After the fourth cube turns out of alignment during placement, the agent regrasps and corrects it. To overcome limited reach at the sixth level, it assembles the top two cubes on the table and transfers them together by grasping the lower cube. All six cubes stand after release, but the tower collapses during the subsequent arm withdrawal. The observations establish when the collapse occurs, while the exact contact or instability that causes it remains unresolved.

The agent then localizes the scattered cubes to assess whether it can rebuild. Both overhead geometry and wrist depth place one grey cube near $x = 0.70$ m in the robot base frame, beyond the interface's 0.65 m radial grasp limit. It ends the attempt as unsuccessful despite having time and command budget remaining.

This case distinguishes a recoverable alignment error from a collapse that moves a required cube beyond reach. It also shows why stability during withdrawal is part of the success criterion.

F.2 Grasp Loss and Unstable Placement in Four Pair Assembly

In one four pair assembly execution, the agent completes three pairs before failing on the long pin and narrow tube. The first vertical insertion catches the tube rim and tips the tube. During a horizontal recovery, the tube slides with the advancing pin, and a temporary engagement is lost when the agent rotates the pair. After restoring the tube upright, the agent attempts another vertical insertion and loosens the jaws when descent stops short. The pin leaves the gripper, but subsequent views show an empty tube and fail to locate the pin. The record establishes grasp loss and incomplete assembly, while the pin's final location remains unresolved.

Another four pair assembly execution illustrates unstable placement after engagement. Repeated regrasping and relocation eventually produce an engaged pin and tube, but the combined assembly repeatedly tips after release. The session ends without a verified stable placement. Together, these trials show that partial insertion and engagement while supported by the gripper are insufficient evidence of a completed assembly. Recovery also changes the grasp and support geometry, requiring renewed checks before release.

F.3 Insertion Misalignment and Contact During Recovery

In a two pair assembly execution with Fable, repeated lowering leaves the hexagonal nut supported on the top of the stud. One attempt partially engages, but further descent jams and the arm tilts. Releasing the nut then displaces it, and the stud also moves. The session ends before the agent attempts the circular pair.

The execution report identifies offsets between the held part and the reported grasp point, together with residual positioning error under contact, as possible contributors. These are interpretations from the session rather than an independently calibrated error decomposition. The observed sequence nevertheless shows how contact can change both part poses during insertion. Continuing from the previous alignment then requires remeasuring the held part and receiver. In the four pair assembly case described above, the agent successfully recovers an earlier hexagonal insertion after measuring and compensating for a held-part offset, before the separate pin failure described above.

F.4 Recovery Near the Workspace Boundary

The unsuccessful Terra experience-transfer trial places the narrow receiver at a measured radius of approximately 0.118 m, just inside the interface's 0.120 m lower limit. The initial transfer request is rejected before motion. The agent then offsets the insertion target to satisfy the workspace constraint. Although it initially describes the collar as engaged, inspection after withdrawal shows the parts separate. Subsequent descents return settling errors, and recovery eventually leaves the narrow receiver on its side. The circular pair remains assembled.

This case shows a limit of transferring a successful procedure to a changed scene. The inherited insertion sequence does not resolve the receiver's accessibility. Moving an insertion target into the

permitted region also changes its alignment with the physical receiver. The recorded rejection, failed engagement, and later tipping identify successive obstacles in recovery without establishing a single mechanical cause for every contact.

F.5 Fabric Slippage and Incomplete Fold Verification

In one simultaneous towel folding execution, the agent loses the left corner during transport and then loses the right grip after regrasping. Further attempts translate and rotate the towel as a whole, while the intended free edge remains curled. The agent changes its anchoring strategy and completes the two inward end motions, but the final image retains an unfolded corner. This sequence links repeated grasp loss and movement of the lower layer to an incomplete fold arrangement. The record does not isolate whether grasp depth, fabric friction, or applied tension dominates each slip.

In another execution, the agent recovers an initially empty right grasp and completes the planned motion sequence, but operator review also identifies an unfolded corner. The agent describes the result as completed with curled corners. This disagreement exposes a verification failure in addition to imperfect manipulation. Executing the intended fold sequence and producing a compact footprint do not establish that each required corner and layer has reached its target arrangement.

F.6 Throwing Shortfall and Bowl Displacement During Recovery

In an unsuccessful throwing execution, the agent uses an initial release probe whose predicted reach is short of the bowl. The agent's analysis of the recordings places object detachment approximately 0.49–0.62 s after the scheduled opening command. This interval reflects the observed release of the held object, including jaw motion and slipping. The report identifies an impact on the table before the object rebounds into the bowl, so bowl occupancy alone does not verify the required direct throw.

The agent then attempts to retrieve the object for another launch. Retention fails during recovery and the bowl shifts visibly, by approximately 3 cm according to the report's image-based estimate. The final photograph shows the object inside the shifted bowl. The agent stops and reports failure because the direct flight was unverified and the bowl moved, violating the stationary-target requirement. This case separates successful final occupancy from compliance with the prescribed motion, and shows how recovery can introduce an additional task violation.

G Broader Impact

AGP may reduce the task specific engineering required to prototype manipulation from video, image, and language instructions. Saved programs and execution records could also support research on reusable robot skills.

A potential application is collecting robot trajectories for policy learning through repeated AGP execution. Synchronized visual observations, actions, and measured robot states could provide demonstrations for training vision language action policies, while failed attempts and recovery segments could support research on robustness. Its practical value would depend on trajectory quality, collection costs, and the performance of policies trained on these data.

Applying agents to physical systems introduces risks from inaccurate perception, faulty programs, and inappropriate motion requests, which can cause equipment damage or injury. Our implementation separates agent reasoning from motion execution and applies workspace and motion limits through the robot runtime. Deployment in settings involving people or unfamiliar environments requires additional safety evaluation, appropriate supervision, and independent protective mechanisms. Inference costs and execution latency also affect the practicality and accessibility of this approach.

H Execution Across Arm and End Effector Embodiments

The bottle experiment in Figure 7 uses a seven joint P7 arm and a RealHand L6 dexterous hand. The hand has five fingers controlled through thumb flexion, thumb abduction, and one channel for each remaining finger. A fixed overhead RealSense D435i and a wrist mounted RealSense D435 provide

color images at 640×480 pixels and 15 Hz. This setup tests the observation and action loop with different arm kinematics and finger control.

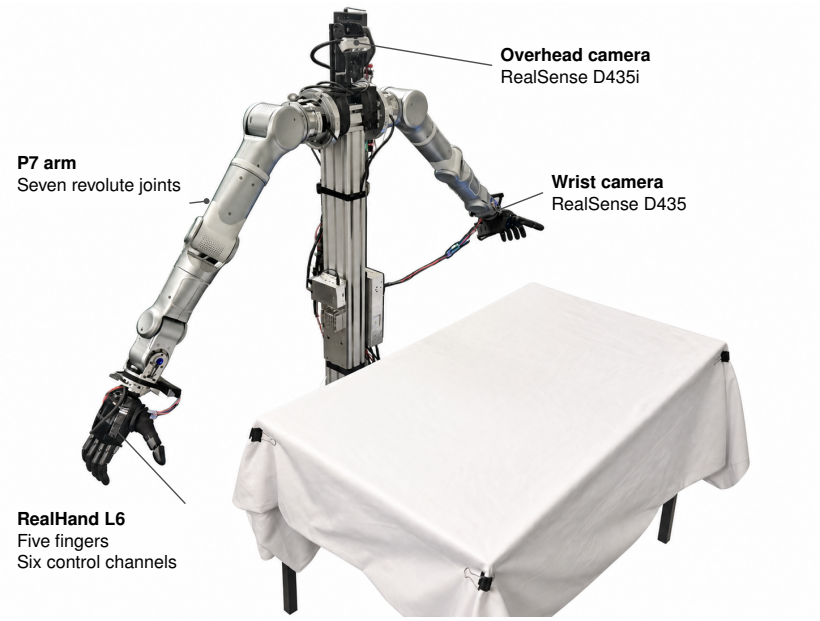


Figure 7: **P7 and L6 platform for the cross embodiment experiment.** The annotations identify the P7 arm, RealHand L6 dexterous hand, overhead RealSense D435i, and wrist mounted RealSense D435.

Figure 8 shows the grasp and lift sequence. From a stationary tabletop preparation pose, the agent reorients the wrist, opens the thumb, and approaches the bottle through successive position adjustments. It then closes the fingers in stages and commands a 2 cm upward wrist motion to raise the bottle. The demonstrated grasp and lift provides qualitative evidence that AGP can coordinate a different arm and dexterous hand through their available interfaces.

I Qualitative Task Examples

I.1 Four Pair Assembly

Figure 9 shows recovery during the four pair assembly task. An unsuccessful insertion displaces the nut onto the table. The agent adjusts its grasp position and gripper orientation, seats the nut, and continues with the remaining assemblies while preserving earlier progress. The final frame shows all four pairs assembled after release and withdrawal.

I.2 Block Construction

Figure 10 shows how placement order follows the target support structure. The pyramid is built tier by tier. The two towers are built together, with both bases placed before the middle and top cubes. The six block tower grows above a retained base, requiring alignment to be maintained over successive placements. All three structures remain standing after release and withdrawal.

I.3 Dice Flipping

Figure 11 shows grasp adjustments that accommodate the arm's reachable poses. The agent combines a 60° tilt with a low release so that a die settles onto the intended face. Further observations guide regrasping and verify orientation. The final frame shows all six target faces upward after withdrawal.

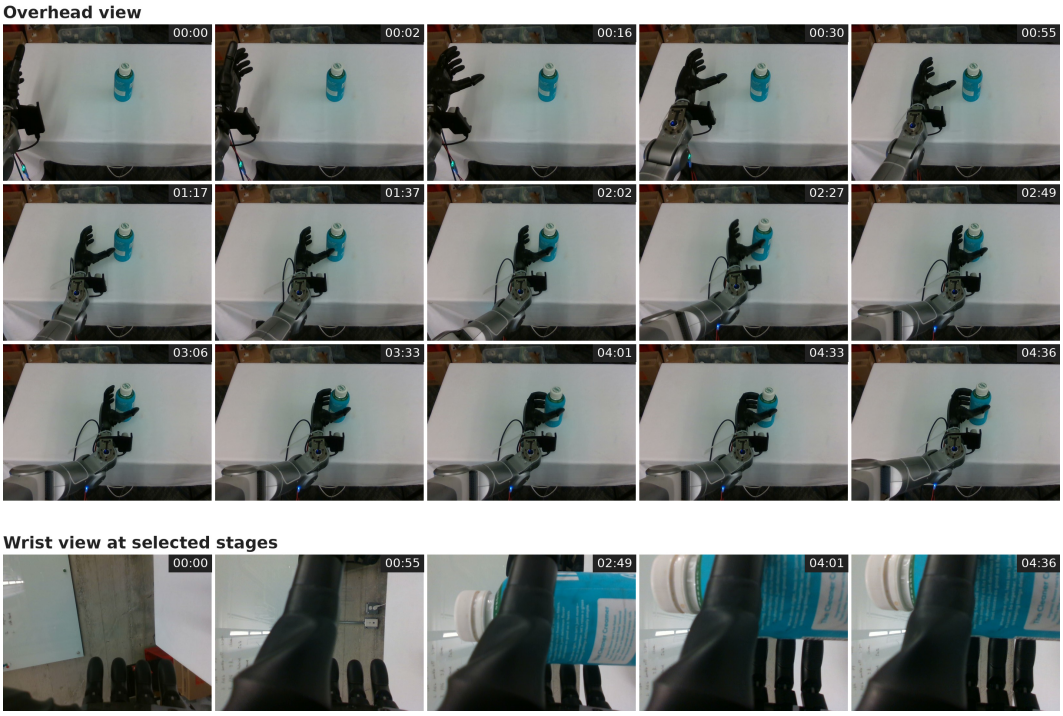


Figure 8: **Bottle grasp and lift with the P7 arm and L6 hand.** The upper panel contains 15 overhead frames read row by row, showing wrist reorientation, approach, finger closure, and lifting. The lower panel shows five corresponding stages from the wrist camera. Timestamps measure elapsed minutes and seconds from the first grasp action after stationary preparation. Frames are sampled at unequal intervals, with the last frame at 04:36. The two views are approximately aligned using their recorded host timestamps.

I.4 Targeted Throwing

In Figure 12, the agent performs two motion probes while retaining the object, then adjusts the swing and gripper opening schedule. Joint 4 drives the swing while the other joint targets remain fixed. The views show separation from the gripper and the object resting inside the bowl. Occlusion obscures part of the rising trajectory and its apex, so the full throwing assessment also uses the execution records and video.

I.5 Bimanual Towel Folding

Sequential folding. In Figure 13, a corner slips during the initial paired carry. The agent recovers it and corrects a turned back edge before bringing the doubled towel closer to the arm bases. One arm then holds the towel while the other folds an end inward. Further grasps flatten a raised corner.

The final towel preserves the demonstrated fold order after release and withdrawal. A buckle and edge offsets remain, and the execution report judges the result a partial visual match. The sequence illustrates successful recovery of the fold arrangement with residual alignment errors.

Simultaneous folding. In Figure 14, one arm holds its corner while the other recovers a slipped edge. After the long edge fold, both arms carry the short ends inward, then place the flaps in sequence to maintain clearance. A further correction anchors an outer edge while drawing a raised inner hem toward the center.

Both inward folds are completed after release and withdrawal, with residual skew, a curled edge, and a gap near the central seam. The trial is counted as successful under the criteria in Table 4. The coordinated inward transport and staged placement show how the agent adapts the demonstration to the two arms' workspace.

J Baseline Resource Accounting

Table 3 reports two accounting scopes for the same ASPIRE runs. Real only measures successful program execution from launch to completion using the duration recorded by the trial runner. This duration includes initialization and result recording. Sim + real adds the full shared program construction and refinement overhead in both simulation and the real world to each successful run. For sim + real, ASPIRE time sums the elapsed intervals of the dedicated simulation refinement, skill promotion, and real robot adaptation workers, together with successful physical execution. These intervals include program generation, debugging rollouts, internal waiting, and delivery of the resulting program or skills. Gaps between worker sessions and separate simulation evaluation sweeps are excluded. Inference cost is estimated using API prices for the corresponding coding model requests, including cached inputs and retries. Supervising operator conversations, hardware setup, and local perception service computation are outside this accounting. Frozen program replay adds physical execution time without further coding model inference.

For two pair assembly, simulation construction and refinement take 42.14 minutes and USD 11.70, and real robot adaptation takes 19.71 minutes and USD 7.20. The two successful physical executions take 166.3 and 183.8 seconds. The real only values are 2.9 minutes and USD 0.00. Adding the full simulation and real robot adaptation overhead gives 64.8 minutes and USD 18.90 for sim + real.

For pyramid construction, the simulation cost includes the four successive curriculum tasks used to build the transferred skills. These take 233.76 minutes and USD 63.60. Real robot adaptation takes 37.91 minutes and USD 14.52, including the successful 553.1 second physical execution. That execution is counted once. The real only values are 9.2 minutes and USD 0.00, and sim + real reports 271.7 minutes and USD 78.13. The reported 1/10 outcome includes this success during adaptation, so it measures the reported sequence of attempts rather than a separate evaluation of ten trials with a frozen program. GaP has no successful runs on any of the three tasks, so its successful run resource means are undefined.



Figure 9: **Four pair assembly from a human demonstration video.** The upper panel shows four demonstrated assemblies. The lower panel shows the robot assembling eight parts into four pairs, illustrating video instruction following and long horizon reasoning. Dark red bars mark failed nut insertion and displacement. Yellow bars mark grasp retries. Read each panel row by row. Timestamps give elapsed minutes and seconds in each video at unequal intervals.



Figure 10: **Block construction from static goal images.** From top to bottom are the pyramid, two towers, and six block tower. Each occupies two rows of five images, beginning with the goal image labeled Demo and followed by nine execution frames read row by row. Timestamps give elapsed minutes and seconds in each recording at unequal intervals.



Figure 11: **Dice flipping across six dice.** The goal photograph in the cell labeled Demo specifies six pips upward on each of three black and three light blue dice. Nine overhead execution frames show the initial faces, successive manipulations, and final state with all six target faces upward. The task requires selecting grasps and rotations that produce the requested orientation after release. Read the execution frames row by row. Timestamps show elapsed minutes and seconds in the source video.

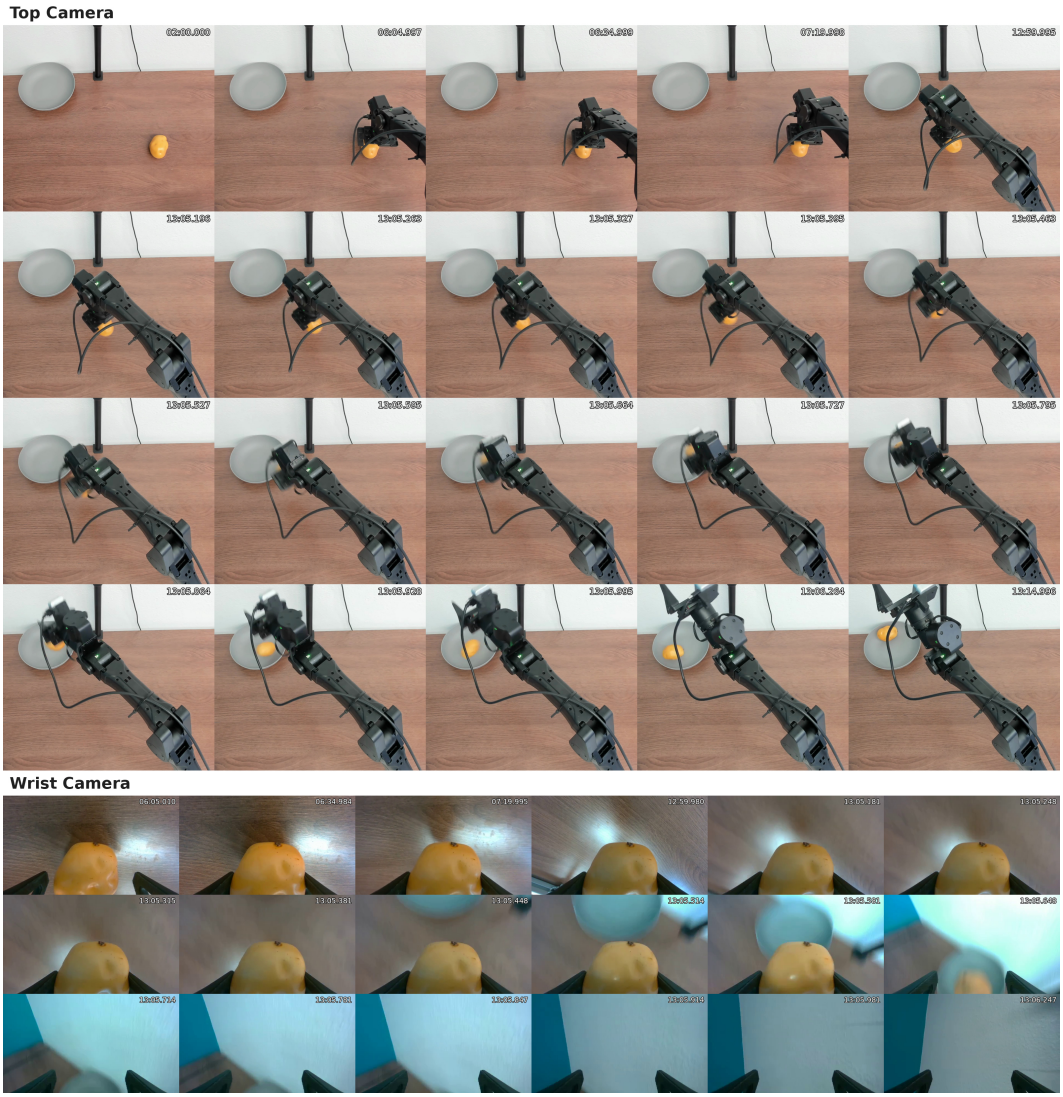


Figure 12: **Targeted throwing from a language instruction.** The task requires a joint driven circular swing and timed gripper release toward a bowl, illustrating the dynamic release component of action diversity. The overhead sequence shows the initial arrangement, grasp, swing, and final object position inside the bowl. The wrist sequence shows retention and separation at the gripper. Occlusion limits the visible flight trajectory. Read each grid row by row. Timestamps show elapsed minutes, seconds, and milliseconds on a common clock, with denser sampling around release and landing.

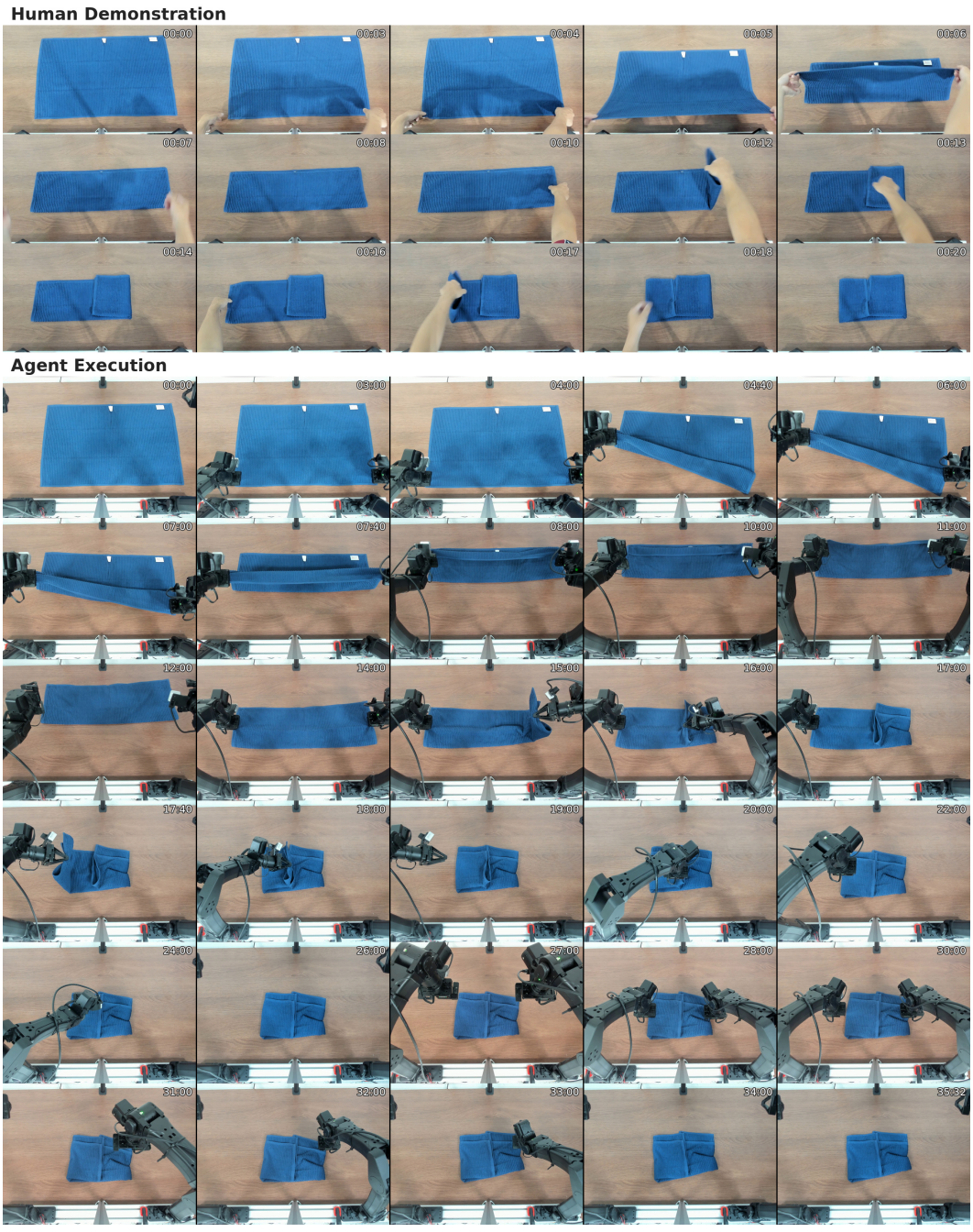


Figure 13: **Sequential bimanual towel folding from a human demonstration video.** The upper panel contains 15 frames showing the long edge fold followed by the right and left ends folding inward. The lower panel contains 30 frames showing coordinated grasping, recovery after a corner slips, the three folds, and subsequent adjustments. The final towel preserves the demonstrated fold arrangement with a remaining buckle and edge offsets. Read each panel row by row. Timestamps show elapsed minutes and seconds in the respective source videos, with unequal sampling intervals.

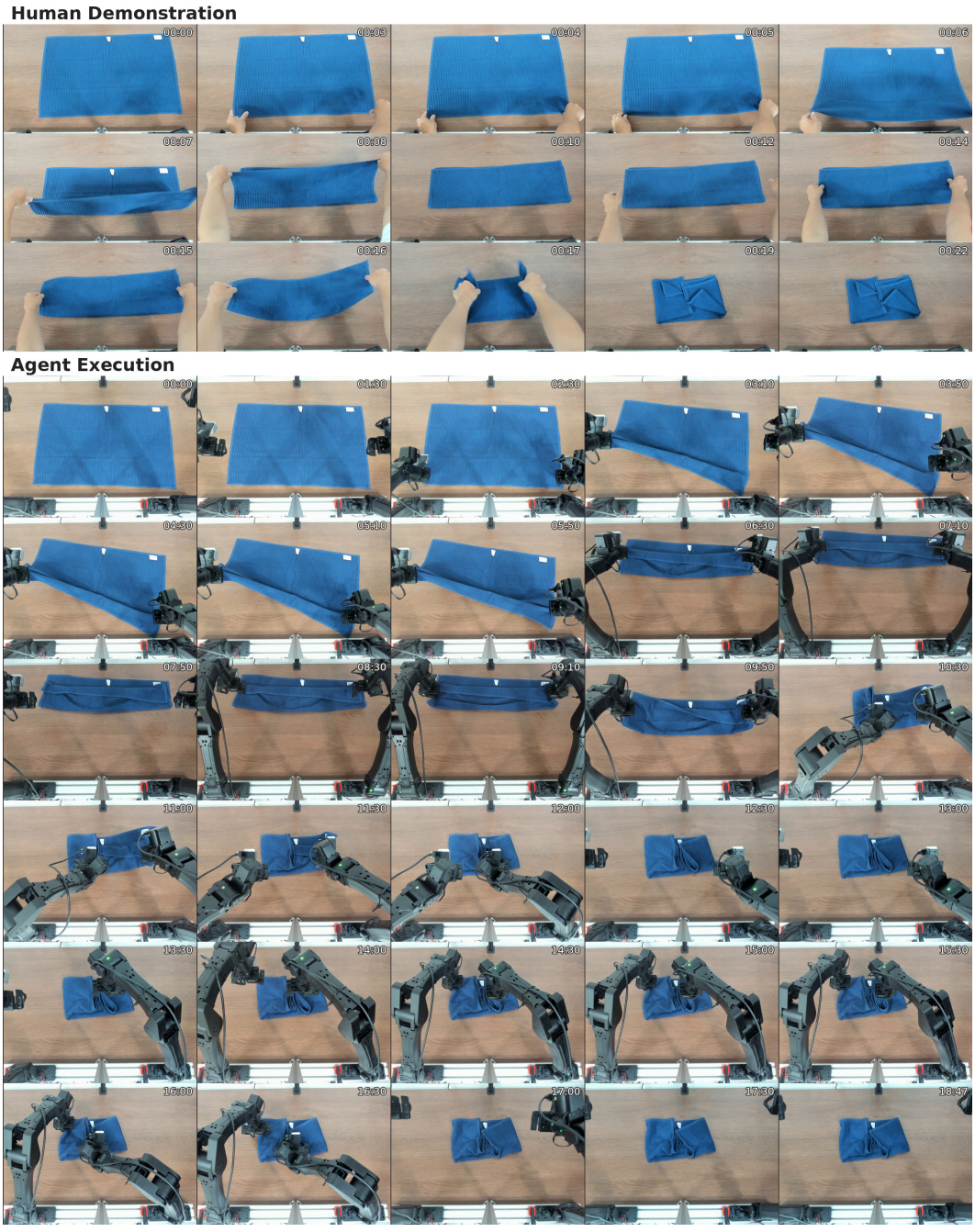


Figure 14: **Simultaneous bimanual towel folding from a human demonstration video.** The upper panel shows the long edge fold followed by both short ends folding inward together. The lower panel shows the robot execution, including recovery of a slipped corner, the long edge fold, inward transport of both ends, staged placement of the flaps, and alignment correction. The final state contains both inward folds with residual skew and edge offsets, with both arms withdrawn. Read each panel row by row. Timestamps show elapsed minutes and seconds in the respective source videos, with unequal sampling intervals.