MAKE CODE AS POLICY GREAT AGAIN: FRONTIER AGENTS WRITE, CALL, AND EVOLVE ROBOT TOOLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Frontier models can control robots, but reasoning through every reach, grasp, and retreat makes manipulation slow and token-intensive. We revisit code as policy with a different division of labor: models build executable tools, code handles multi-phase motions, and models decide what to do next. We introduce URAI (Universal Robot–Agent Interface), which couples a programming agent that constructs robot tools with an execution agent that uses them in a feedback loop. The programming agent writes reusable and task-specific tools from task intent and refines them through execution feedback and human guidance. The execution agent selects and parameterizes these tools from current observations; each call runs a complete motion locally before returning control to the agent. Unlike delegating subsequent decisions to a generated program, this design retains model-level decision-making between tool executions. Validated tool revisions persist across episodes without updating foundation-model weights, and a shared GUI and API make the same tools available to humans and agents. Across five RoboDojo tasks and four frozen execution agents, URAI raises aggregate success from 18.0% to 53.0% relative to direct fingertip control, with the largest gain on Swap Blocks; with the same tools, a program written in advance reaches only 24% against 56% for two agents deciding after each call. All four agents also finish episodes faster, three of them 1.3–1.5$\times$ faster with 1.5–1.7$\times$ fewer execution-agent output tokens. We further evaluate URAI on seven real-world AgileX dual-arm tasks, spanning object manipulation, cloth folding, and human-interactive tic-tac-toe. URAI connects the coding and decision-making capabilities of frontier agents, organizing robot control around reusable tools that agents can both invoke and revise.

1 INTRODUCTION

Frontier models are becoming capable of complex spatial work through software tools: GPT-6 Astra reconstructs three-dimensional objects by generating CAD code (OpenAI, 2026) and builds editable Blender scenes that it inspects through computer use (Ricouard, 2026). The same model also completes physical manipulation tasks by selecting end-effector targets and gripper commands from visual observations (Robocurve, 2026): it inspects the scene, chooses a Cartesian pose and gripper state for the robot's inverse kinematics and local controller to realize, and observes the outcome before deciding what to do next. In this setting the model itself is the policy, one motion at a time; GPT-Policy studies it on real robots with in-context demonstrations and human interaction (Cheng et al., 2026).

The difficulty is turning this capability into an execution speed suitable for practical use. When a manipulation is decomposed into many model-selected motions, each observation–reasoning–action cycle adds waiting time and token consumption. In an independent block-placement evaluation, GPT-6 Astra completed 19 of 20 trials with a reported mean of 2.5 minutes and 2.1k output tokens per run under the study's operating limits (Robocurve, 2026); InSight separates model latency from robot motion in a comparison with CaP-X (Wang et al., 2026). Practical deployment therefore requires evaluating how long a successful interaction takes, alongside whether it succeeds.

Action granularity is one lever. A model asked to specify successive end-effector targets must manage motion details that a local tool can handle; a tool can instead combine several motion phases into one parameterized request, reusable across tasks or written for one. VLA models predict tok-

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.

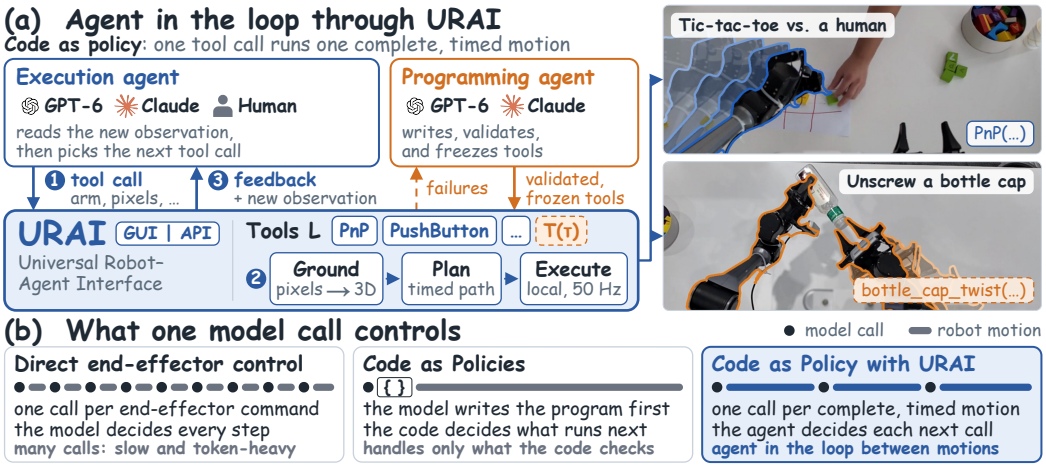


Figure 1: **URAI overview.** (a) A programming agent writes and validates reusable and task-specific tools. An execution agent selects tools from observations and feedback, while a shared GUI/API backend grounds, checks, and executes multi-phase motions locally. Fading outlines in the demonstration-video frames indicate earlier arm poses within one tool call. (b) Decision boundaries: direct control queries the model for each end-effector command; Code as Policies delegates subsequent decisions to generated code, which may query perception; URAI returns control to the agent between tool executions.

enized or continuous low-level actions (Kim et al., 2025; Yuan et al., 2026) but generalize poorly to new scene layouts: on RoboDojo, VLA and world action model (WAM) policies keep a median of 12–16% of their success when the layouts are randomized, whereas GPT-6 Astra acting through RoboDojo's own RoboProbe tool harness keeps 87% on the official leaderboard (RoboDojo Team, 2026). Agentic interfaces expose discrete motion commands to a frozen VLM (Guo et al., 2026; Chen et al., 2026b), and program- and constraint-based systems delegate execution to local controllers (Liang et al., 2023; Huang et al., 2025); the design choice is which decisions stay with the model.

Software development offers a template: with coding agents, developers state what they want, run what the agent writes, and ask for changes, a practice Karpathy (2025) named vibe coding. We ask whether robot control can work the same way: *can the policy be code that agents write and revise from intent and feedback, so that a frontier model decides which complete motion to run instead of every step, and does this cut end-to-end time and tokens while keeping manipulation reliable?*

Code as Policies showed that a language model can write a robot policy as a program, generated hierarchically and able to call perception (Liang et al., 2023); at run time its code, not the model, decides what runs next, and multi-turn variants regenerate code from execution feedback between runs (Fu et al., 2026). We make code as policy great again with frontier coding agents, keeping the agent in the loop between tool executions (Figure 1). The robot policy combines agent-written executable tools with feedback-driven tool selection by a frozen foundation model, the execution agent; a programming agent vibe-codes the tools from natural-language intent and execution feedback, starting from reusable tools such as pick-and-place (PnP), PushButton, or Home and writing a task-specific tool when these do not adequately cover a task. The execution agent selects and parameterizes a tool from the current observation and decides again from the feedback after every tool execution; a PnP request covers approach, grasp, transfer, placement, and retreat without a model call for each phase. Unlike casual vibe coding, a tool enters the policy only after it passes validation, and the version used for evaluation is frozen; the two roles separate the cost of writing the policy from the cost of running it. Direct end-effector control also leaves inverse kinematics and local control to the robot; what changes is the granularity and frequency of model decisions: one call executes a complete multi-phase motion, and the model decides the next step from feedback between calls.

The tools run behind URAI, a universal robot–agent interface that exposes the same tool contract through a GUI and a structured API over one backend that grounds requests, checks planned motion,

and executes locally with explicit paths, orientations, timing, and gripper events; humans can steer the programming agent in natural language and operate the same tools through the GUI. Because the tools are code, the policy can also improve without training: guided by failures and natural-language steering, the programming agent revises the tool collection L, tool code C, and execution strategy P with foundation-model weights fixed; validated versions are retained for later episodes and frozen before evaluation.

Recent systems share parts of this design: Agent as Policy drives a real robot with a coding agent that writes and revises programs from physical feedback, with separate preparation and execution agents (Jia et al., 2026); CaP-X adds multi-turn feedback and skill synthesis to Code as Policies (Fu et al., 2026); ENPIRE and ASPIRE refine code policies from rollouts (Xiao et al., 2026; Lu et al., 2026); LATM, CRAFT, and Voyager split tool makers from users or accumulate code skills (Cai et al., 2024; Yuan et al., 2024; Wang et al., 2024). None of these elements is new in isolation; what we isolate is the handoff boundary: a fixed tool collection, a frozen execution agent that decides again after every complete tool execution, and the same tool contract for humans through a GUI, evaluated against direct low-level control and against a program written in advance over the same tools.

Our contributions are fourfold: (1) a revival of code as policy in which the policy combines agent-written executable tools with feedback-driven tool selection by a frozen execution agent, separating tool programming from tool execution; (2) URAI, a universal robot–agent interface shared by humans and agents, with identical GUI and API semantics over reusable and task-specific tools and a local timed-motion backend; (3) persistent tool revision without weight updates, covering the tool collection, tool code, and execution strategy, with a case study of author-guided extension of a rigid-object PnP tool to cloth manipulation; and (4) an evaluation of task success, time, and token cost on five RoboDojo tasks (Chen et al., 2026a) with four execution agents and on AgileX dual-arm tasks against published step-by-step control. In simulation, all four execution agents succeed more often through URAI than under direct fingertip control (53.0% against 18.0%; 42.5% against 22.5% without swap blocks), a program written in advance over the same tools reaches 24% against 56% for two agents deciding after each call, and all four finish episodes faster with fewer output tokens, three of them by 1.3–1.5$\times$ and 1.5–1.7$\times$. On the real robot, against published runs of the same model on other robots under different speed limits and timing scopes, URAI finishes tic-tac-toe games 2.7$\times$ faster than GPT-Policy with the same 3/3 successes, and block into bowl 2.4$\times$ faster with 4.7$\times$ fewer output tokens than direct end-effector control.

2 Related Work

Frontier models as robot decision makers. Code as Policies establishes hierarchical program generation, with perception calls inside the program, as an interface to perception and control (Liang et al., 2023); CaP-X extends it with multi-turn interaction, structured execution feedback, and automatic skill synthesis, and exposes a trade-off between the convenience of higher-level primitives and the expressivity of lower-level control (Fu et al., 2026); Agent as Policy drives a physical robot with a coding agent that writes programs and revises them from physical feedback, with a preparation agent for reusable task definitions and an execution agent per trial (Jia et al., 2026). VoxPoser composes three-dimensional value maps (Huang et al., 2023), MOKA and PIVOT use visual marks and iterative visual prompting (Liu et al., 2024; Nasiriany et al., 2024), ReKep generates constraints that a local optimizer converts into reactive motion (Huang et al., 2025), RoboStream keeps a frozen VLM in a closed loop through an object-centric scene-graph memory (Huang et al., 2026), and Inner Monologue feeds environment feedback back into language-model planning (Huang et al., 2022); GPT-Policy takes the step-by-step route with a frontier VLM on real robots (Cheng et al., 2026). Persistent improvement outside model weights has precedents in Voyager's code skill library (Wang et al., 2024), in LATM and CRAFT, which separate tool makers from tool users for language agents (Cai et al., 2024; Yuan et al., 2024), and in robotics in ASPIRE's validated skill repairs (Lu et al., 2026), ENPIRE's reset–execute–verify–refine loop on physical rollouts (Xiao et al., 2026), and RHO's search over policy repositories against simulated reward (Elmaaroufi et al., 2026). URAI claims none of these elements in isolation. Its distinction is the handoff boundary it evaluates: a fixed collection of agent-written tools, a frozen execution agent that decides again after every complete tool execution, and one tool contract shared by the GUI and the API.

Frontier-model control and action granularity. Robocurve evaluates GPT-6 Astra on physical YAM arms through absolute end-effector pose commands, with a 20-call budget and a 25% speed cap (Robocurve, 2026), and Su et al. (2026) compare direct control with hybrid control that accepts or corrects $\pi_{0.5}$ proposals. OpenVLA predicts learned, quantized per-dimension motor tokens (Kim et al., 2025), Qwen-RobotManip emits continuous action chunks (Yuan et al., 2026), VGGT-DP and Spatial Policy ground learned visuomotor policies in 3D foundation features and spatially conditioned video prediction (Ge et al., 2026; Liu et al., 2025), and RoboDawn (Guo et al., 2026) and Show-Harness (Chen et al., 2026b) give a frozen VLM a fixed vocabulary of discrete translation, rotation, and gripper commands. URAI also presents discrete tool calls, but a call can run a reusable tool such as PnP or a task-specific procedure whose internal motion phases execute locally; the distinction is the duration and responsibility assigned to one model-level decision, together with the cost of authoring the tool.

Interfaces and efficiency. Trajectory and sketch interfaces condition learned policies or frontier agents on rough trajectories, image-space paths, editable sketches, sparse waypoints, or a browser-based three-dimensional interface (Gu et al., 2023; Li et al., 2025; Tan et al., 2026; Sundaresan et al., 2024; Hu et al., 2026), and human interfaces range from sketching (Iwanaga et al., 2025) to teleoperation and demonstration capture (Zhao et al., 2023; Zhang et al., 2018; Chi et al., 2024); Show-Harness's GUI lets humans and VLMs issue the same single-step units (Chen et al., 2026b). URAI likewise shares one action vocabulary between human and model operators, but each entry is a complete, timed tool with explicit metric arguments rather than a fixed increment. Computer-use work treats the observation/action boundary as part of an agent system (Zhou et al., 2024; Xie et al., 2024; Agashe et al., 2024) and motivates measuring efficiency in addition to success (Abhyankar et al., 2025); in robotics, InSight reduces VLM thinking time through acquired primitives (Wang et al., 2026), and Jev and RoboJEV shorten each decision (Almeida, 2026; Liu, 2026). These systems lower the cost of each decision; URAI instead lowers the number of decisions per episode.

3 Code as Policy with the Agent in the Loop

3.1 The Policy: Agent-Written Tools and Feedback-Driven Selection

We write the robot policy as $\pi_{\theta,\varphi}$, where θ are the fixed weights of the foundation model behind the execution agent and $\varphi = (L, C, P)$ is the external system: the tool collection L, the executable tool code C, and the execution strategy P, meaning the task guidance and call conventions given to the execution agent. At each decision, the execution agent maps the instruction, the current observation, and the execution history to one tool call (ℓ, ξ) with $\ell \in L$ and arguments ξ. The code C_ℓ turns the call into a timed motion that runs locally until it completes or is interrupted, so one policy decision spans a complete multi-phase motion rather than one end-effector target. Adaptation happens in φ, never in θ. The programming agent proposes φ_{k+1} from φ_k, development feedback, and natural-language intent, runs the candidate, and keeps it only if it passes validation: a vibe-coding loop (Karpathy, 2025) in which every accepted version is tested and frozen before evaluation.

3.2 Interaction Contract

The programming agent receives a task specification, available tool definitions, and development feedback, and can revise a reusable tool or create a task-specific tool, including its code, argument contract, and execution strategy. The execution agent, or a human calling the tools of L through the GUI with the same execution semantics, receives the task instruction, camera and depth-derived observations, robot state, and execution feedback, and invokes an available tool through the GUI or API. The shared backend grounds and validates each request, produces a timed trajectory, and executes it locally; after completion or interruption the execution agent can observe again and issue another request, and recurrent failures can be passed to the programming agent for a versioned tool revision (Figures 1 and 2).

3.3 A Motion Representation with Explicit Timing

For an arm, we describe a request as

$$a = \big(p(s), R(s), v(s), E, \mathcal{C}\big), \qquad s \in [0, 1], \tag{1}$$

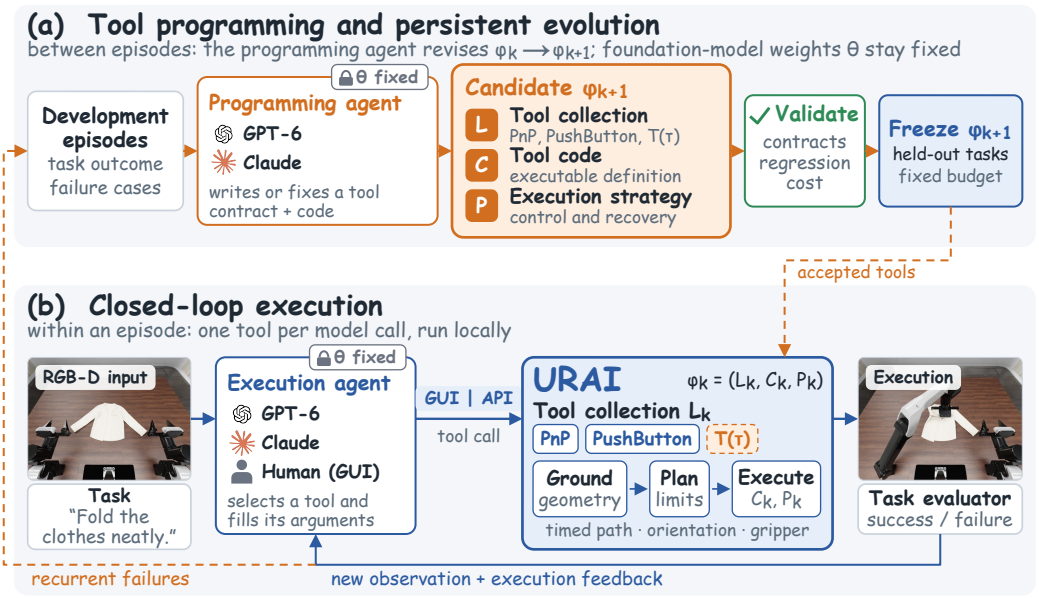


Figure 2: **Tool development and execution in URAI.** (a) Between episodes, the programming agent revises the tool collection, code, and execution strategy using development feedback. Validated versions are frozen before evaluation; model weights remain fixed. (b) Within an episode, the execution agent selects and parameterizes tools from observations and feedback, while the backend executes each motion locally.

where $p(s) \in \mathbb{R}^3$ is a metric path, $R(s) \in \mathrm{SO}(3)$ is a desired end-effector orientation, $v(s)$ is the requested translational speed profile, E is an ordered set of gripper events and holds, and $\mathcal{C}$ specifies execution settings such as pose tolerances and orientation constraints. Progress s indexes the path; it is not wall-clock time. The prototype accepts sampled control points and bounded mathematical expressions, and interpolates rotations on $\mathrm{SO}(3)$. For a regular translating segment with positive requested speed, the nominal timing relation is

$$\frac{\mathrm{d}t}{\mathrm{d}s} = \frac{\|p'(s)\|_2}{v(s)}. \tag{2}$$

This is the classical path–velocity decomposition (Bobrow et al., 1985; Kant & Zucker, 1986; Pham & Pham, 2018); what the interface adds is that path, speed profile, and gripper events are the arguments of one tool call. Pure rotations, endpoint ramps, stationary holds, angular speed, and robot-dependent motion limits require further timing rules, so a requested speed is a command specification, not a guarantee of achieved end-effector speed. Gripper events are tied to motion progress and execution conditions; ordinary placement and dynamic release use different event semantics.

3.4 Grounding, Editing, and Local Execution

Points selected on observed surfaces are back-projected from calibrated depth, whereas the height of a free-space path is edited in depth-derived orthographic side views. The GUI adds continuous drawing, arm selection, orientation handles, and a speed-profile editor; the API exposes the corresponding control points, profiles, and motion settings through a bounded arithmetic interpreter, so an agent can specify an entire curve without issuing a mouse action for each point. The backend checks the full trajectory, including the approach from the current pose, against the configured kinematic and geometric constraints and streams references locally at a nominal 50 Hz on the AgileX backend, a reference-dispatch frequency rather than a demonstrated tracking bandwidth. Its checks cover the robot and calibrated table, not every object in a scene, and the position-based controller leaves contact-rich tasks requiring force regulation outside the demonstrated scope. The empirical question is whether programmed tools reduce execution-agent interaction costs while maintaining task success.

4 EXPERIMENTS

We evaluate task success, time, and token cost in simulation across execution agents and interfaces, time and token efficiency on a real robot against published step-by-step control, and author-guided tool revision in a case study. Claude Fable 5.1 in Claude Code serves as the programming agent, and Claude Opus 5.5 in Claude Code for the task-specific pouring tool of pour balls; the execution agent is named per row, and in simulation four different frozen models take that role. Agent conditions use the API tool-call path; a human operator driving the same tools through the GUI succeeds in 23 of 25 simulated episodes.

4.1 ROBODOJO: FIVE-TASK SUCCESS RATE

Table 1 compares four frozen execution agents, Claude Fable 5.1 and Claude Opus 5.5 in Claude Code, GPT-6 Astra in the Codex CLI, and DeepSeek-V4-Flash in an open-source coding agent, on multiple RoboDojo tasks (Chen et al., 2026a): *Pour Liquid Into Cup*, *Pour Balls Into Vase*, *Fold Clothes*, *Swap Blocks*, and *Stack Blocks*. Each agent controls the same simulator either through URAI tool calls or *natively*, by commanding fingertip targets with a yaw or, for one arm, any end-effector orientation (so pouring is possible natively), and gripper values. Each episode starts a new agent session in a fresh working directory, isolated from memory, ledgers, and other episodes; the agent sees only the camera images and a public depth probe, chooses pixels and arguments itself, and only RoboDojo's own terminal state counts. Each task has one episode per seed 0–4, with the same thirteen reusable tools and, for pour balls, a task-specific pouring tool, all frozen before evaluation; the agents mainly used the pouring tool, the PnP tool of Section 4.3 (fold and stack), and place-on-pad with button pressing (swap).

Protocol and environment repair. Both conditions use the same task-specific simulator-step budgets (400, 600, 500, 700, and 550, in table order) and a 100-command limit. Agents stop when they judge the task complete or impossible, lack steps for another motion, or reach the command limit, then return both arms home once for official scoring. Wall-clock limits are 45 minutes for URAI and 60 for native control; agent failures and timeouts count as failures. Only one native episode exceeded 45 minutes, and it failed either way. Time spans session launch to exit; tokens count execution-agent output, including reasoning where reported. Development costs are excluded.

A simulator reset defect left the swap-blocks spring button depressed, preventing official success; all 35 initial episodes were voided and rerun after restoring the declared drive targets at reset. All our swap-blocks conditions use this same environment patch, which changes neither tools nor scoring; seeds 5 and 6 were used to diagnose and verify it.

Results. Through URAI, every agent succeeds more often than under native control (52 vs 16% for Claude Fable 5.1, 60 vs 32% for Claude Opus 5.5, 52 vs 16% for GPT-6 Astra, and 48 vs 8% for DeepSeek-V4-Flash; 53.0% against 18.0% overall), and all four finish an episode faster with fewer output tokens, three of them by 1.3–1.5× and 1.5–1.7×. Paired by agent, task, and seed, only URAI succeeds in 40 of 100 pairs and only native control in 5 (exact two-sided McNemar $p = 7.9 \times 10^{-8}$). The largest single gap is swap blocks, solved in 19 of 20 URAI episodes and in none of the 20 native ones; without it, URAI still leads for every agent, 42.5% against 22.5% (21 against 5 discordant pairs, $p = 0.0025$). A program written in advance over the same tools isolates the agent's decisions between calls: in this control, Claude Opus 5.5 or GPT-6 Astra first inspects the scene with observations, depth probes, and dry runs, then writes one program that calls the tools and branches on their returns, and the program runs without the model. It succeeds in 12 of 50 episodes, against 28 of 50 when the agent decides after each call (17 against 1 discordant pairs, $p = 1.4 \times 10^{-4}$), and in 11 against 19 of 40 without swap blocks (8 against 0, $p = 0.0078$). Deciding between calls, not the tool collection alone, therefore accounts for part of the gain. The upper block lists leaderboard values for reference, measured under another protocol and not paired with our episodes, so we do not test them: the best learned policy averages 58.1%, and GPT-6 Astra averages 34.4% through the RoboProbe harness against 52.0% through URAI.

Where execution time is spent. Table 2 shows that URAI replaces many short motion calls with fewer, longer ones: 3.5–10.0 calls per episode lasting 29–72 s each, versus 21.0–25.6 calls of 19–20 s natively. For Fable, Opus, and Astra, time outside robot and observation/probe commands

Table 1: **Success and execution cost on five RoboDojo tasks.** Upper block: success rates as posted on the public leaderboard (RoboDojo Team, 2026) (official protocol, not paired with ours, no cost data) for the three learned policies with the highest five-task mean, $\pi_{0.5}$, and GPT-6 Astra through the RoboProbe harness (Zhang et al., 2026). Lower block: five episodes per task (seeds 0–4); success is the official terminal state; Mean weights tasks equally; Time and Tokens average all episodes, failures included, and count execution-agent output with reasoning. Program: one program over the same tools, written in advance and run without the model (time includes both phases). Pour balls uses its task-specific pouring tool. Bold: best success per task over all rows (ties included); within the leaderboard block and within each agent, the best mean and the lowest time and tokens.

Method	Pour liquid into cup	Pour balls into vase	Fold clothes	Swap blocks[a]	Stack blocks	Mean ↑	Time (min) ↓	Tokens (k) ↓
$\pi_{0.5}$	28.0	13.3	38.7	0.0	13.3	18.7	–	–
GPT-6 Astra, RoboProbe	8.0	4.0	72.0	0.0	**88.0**	34.4	–	–
Xiaomi-Robotics-1	**64.0**	46.0	73.3	0.0	21.3	40.9	–	–
Liber-0 Preview	57.0	31.0	79.0	1.0	72.0	48.0	–	–
Simate-beta	57.3	45.3	**80.0**	38.7	69.3	**58.1**	–	–
DeepSeek-V4-Flash, native	0.0	0.0	0.0	0.0	40.0	8.0	21.4	44.3
DeepSeek-V4-Flash + **URAI**	40.0	**60.0**	0.0	**100.0**	40.0	**48.0**	**20.2**	**40.3**
GPT-6 Astra, native	0.0	0.0	20.0	0.0	60.0	16.0	13.8	6.9
GPT-6 Astra, program	40.0	40.0	0.0	0.0	40.0	24.0	**7.0**	**4.6**
GPT-6 Astra + **URAI**	40.0	**60.0**	0.0	**100.0**	60.0	**52.0**	9.4	4.7
Claude Fable 5.1, native	0.0	0.0	20.0	0.0	60.0	16.0	25.0	52.8
Claude Fable 5.1 + **URAI**	20.0	40.0	20.0	**100.0**	80.0	**52.0**	**17.4**	**30.4**
Claude Opus 5.5, native	40.0	0.0	40.0	0.0	80.0	32.0	17.4	32.3
Claude Opus 5.5, program	0.0	40.0	0.0	20.0	60.0	24.0	15.1	38.0
Claude Opus 5.5 + URAI	40.0	**60.0**	40.0	80.0	80.0	**60.0**	**13.4**	**20.0**

[a]The tools used for swap blocks bundle a task-level place-on-pad procedure with button pressing, so URAI and native control differ in tool semantics as well as call granularity; the program rows share these tools and give the cleaner contrast.

falls by about 45%, 38%, and 32%, respectively, consistent with less time spent deciding between motions. DeepSeek shows the opposite pattern: although its motion-call count falls from 25.6 to 10.0, time outside commands rises from 441 to 509 s, and observations and probes take a larger share of the episode. Coarser motion calls therefore do not ensure lower total cost; the execution agent's observation and decision behavior also matters.

4.2 Real-robot feasibility demonstrations and external references

The real-robot platform is our AgileX PiPER-X dual-arm system with four RealSense RGB-D cameras. Seven tabletop tasks are run (Figure 3), all with GPT-6 Astra as the execution agent calling URAI tools. Unscrew bottle cap and tic-tac-toe correspond to GPT-Policy's dual-arm tasks R3 (its best condition) and R8 (human interaction, same success criterion) (Cheng et al., 2026), and block into bowl repeats Robocurve's task, run there under direct end-effector control (Robocurve, 2026). These published numbers come from other hardware and operating limits, so Table 3 treats them as *published references*, not same-robot measurements. Hot-dog serving, tossing blocks into a bowl, pouring blocks, and clothes folding have no published same-task reference and are reported for URAI alone (lower block).

Measurement protocol. The URAI tasks with published references run at $1\times$ speed; time spans task release to confirmed completion or failure, including robot motion, retries, and waiting for human moves in tic-tac-toe. Output tokens include reasoning; time, tokens, and calls average all three trials, failures included; calls count unique execution-agent model responses, including observations, polling, and recovery, whereas GPT-Policy counts robot-tool requests. Robocurve uses YAM arms, a 25% speed cap, and a 20-call budget over 20 trials per task. The first bottle-cap trial omits an unmetered diagnosis-and-repair phase; tool development was not metered.

Results. All three tasks with a published reference succeed in 3/3 trials (Table 3); equal counts do not establish equal success rates, since the exact 95% lower confidence bound of 3/3 is 29%. In tic-tac-toe, the robot lost none of three games against a human (5.0 minutes, 37.0 model calls, and 3.26k output tokens per game), against GPT-Policy's 13.6 minutes and 69.7 decisions with the same model: $2.7\times$ faster at the same 3/3 success. Unscrew bottle cap took 5.1 minutes and 24.7 calls per trial against R3's 17.9 minutes and 54.7 decisions ($3.5\times$ faster, also 3/3). Block into bowl took

Table 2: **Execution-time decomposition on RoboDojo.** Motion and observation/probe columns give mean per-episode shares of wall time; Other is time outside logged commands. All evaluated episodes, including failures, are included. Bold: within each agent, the fewer motion calls and the less time outside logged commands; the other columns describe the composition of an episode.

Condition	Motion calls ↓	s/call	Motion (%)	Obs./probe (%)	Other (%) ↓	Other (s) ↓
Claude Fable 5.1 + URAI	**4.3**	63	31	31	**38**	**426**
Claude Fable 5.1, native	22.8	20	34	16	51	772
Claude Opus 5.5 + URAI	**3.5**	72	37	35	**28**	**250**
Claude Opus 5.5, native	21.8	20	44	18	38	405
GPT-6 Astra + URAI	**5.6**	44	47	21	**33**	**188**
GPT-6 Astra, native	21.0	19	50	16	34	276
DeepSeek-V4-Flash + URAI	**10.0**	29	31	34	36	509
DeepSeek-V4-Flash, native	25.6	19	40	26	**34**	**441**

Seconds per call pools motion calls; other entries are per-episode means.

Table 3: **Real-robot results.** Upper block: tasks with a published GPT-6 Astra reference; URAI reports three trials per task with mean time, output tokens, and model calls per episode; the references differ in hardware, operating limits, and call definitions and are not same-robot controls. Lower block: tasks without a reference; progress is the fraction completed (hot-dog: items of four; toss blocks: blocks of six; fold clothes: operator score); success requires the whole task. Bold: the better value within each task where both rows report a comparable measure (success only at equal trial counts; a call budget is not a measured count).

Task	Method	Success ↑	Decisions / model calls ↓	Tokens ↓	Time (min) ↓
Unscrew bottle cap	GPT-Policy, published (R3)	**3/3**	54.7	–	17.9
	GPT-6 Astra + URAI	**3/3**	**24.7**	1.52k	**5.1**
Tic-tac-toe	GPT-Policy, published (R8)	**3/3**	69.7	–	13.6
	GPT-6 Astra + URAI	**3/3**	**37.0**	3.26k	**5.0**
Block into bowl	Robocurve, published	19/20	$\leq$20 (budget)	2.1k	2.5
	GPT-6 Astra + URAI	3/3	6.3	**443**	**1.0**
URAI-only tasks, ten trials each (success; mean progress); cost over the first three trials					
Hot-dog serving	GPT-6 Astra + URAI	10/10 (100%)	–	2.40k	7.9
Toss blocks into bowl	GPT-6 Astra + URAI	7/10 (91.6%)	–	4.69k	8.1
Pour blocks	GPT-6 Astra + URAI	10/10 (100%)	–	2.29k	3.1
Fold clothes	GPT-6 Astra + URAI	8/10 (90.0%)	–	1.41k	4.1

URAI calls count execution-agent model responses; GPT-Policy counts robot-tool requests; $\leq$20 is a call budget. The first bottle-cap trial excludes unmetered diagnosis and repair. –: not reported.

61.3 s, 6.3 calls, and 443 output tokens per trial against Robocurve's 2.5 minutes, at most 20 calls, and 2.1k tokens (19/20) with the same model under direct end-effector control: 2.4× faster with 4.7× fewer tokens. The first trial (71.2 s, 731 tokens) also wrote the execution script that the other two reused (51.5 s, 182 tokens; 61.2 s, 417 tokens); on its own it is still 2.1× faster with 2.9× fewer tokens. Among the URAI-only tasks, run ten times each, hot-dog serving and pour blocks succeed in all ten trials, fold clothes in 8/10 (90.0% mean progress), and toss blocks into bowl in 7/10 (91.6%); their token and time columns are means over the first three trials, the only ones with a token log. Without a same-robot step-by-step baseline, the ratios above describe our runs relative to published protocols and are not causal estimates of a speedup. Further calls can finish an incomplete motion: in the second pour-blocks trial the agent added one pour to complete the task. Missing geometry can still prevent success: in the third toss-blocks trial, after a missed grasp beside the left-arm base, no reliable depth could be obtained, and the last block stayed outside the bowl.

4.3 Tool evolution: revising one tool for fold clothes

The programming agent, steered by the author in natural language, revised the existing two-point PnP tool of Table 1 seven times (eight versions with the initial one) instead of writing a folding routine. The initial version rejected every click on the shirt as a large rigid object; the revisions

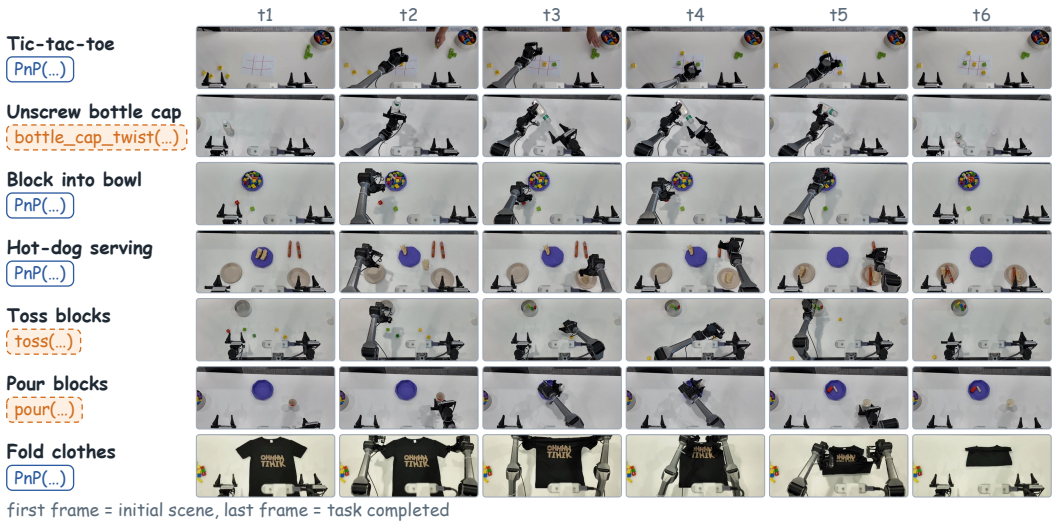


Figure 3: Key frames of the seven AgileX tasks under URAI with GPT-6 Astra, initial scene (t1) to completion (t6); chips name the tool called. Fold clothes is the demonstration-video run; other rows are evaluated trials.

add sheet pinching with local thickness checks, synchronized dual-arm transport, outward wrist tilt while carrying, vertical release, and a drop height and clearance taken from depth; for example, replacing the median layer height with its local 0.9 quantile lets the fingers release above an already folded layer. Each version keeps the call signature, ships with unit tests, and has no branch on seed, garment, or task name. The frozen final version is tested by the four frozen execution agents in fresh sessions, choosing pixels themselves: 3 of 20 episodes succeed on seeds 0–4 (Table 1) and 2 of 20 on seeds 5–9, whereas a human operator choosing pixels through the GUI succeeds in 4 of 5. With this tool, choosing pinch and drop points is the agents' main difficulty: in most failed GPT-6 Astra and Claude Opus 5.5 episodes, the first sleeve pinch sits at the sleeve tip, where the fingers close on nothing.

5 Limitations and Discussion

Our measurements cover five simulated and seven real-robot tasks with few episodes per cell; under our protocol, with this tool collection and task mix, all four execution agents have higher aggregate success and lower execution-phase time and token cost, DeepSeek-V4-Flash only slightly. Pooled over agents, the success gain is significant with and without swap blocks (McNemar $p = 7.9 \times 10^{-8}$ and $p = 0.0025$), but five episodes per cell (GPT-6 Astra was released shortly before submission) do not establish per-task differences. The program control covers two agents on seeds 0–4 and withholds run-time model inference, not decision timing alone; no prompt- or memory-only revision control was run. The real-robot comparisons are against published runs on other hardware with no same-robot step-by-step baseline, so the reported ratios are not causal estimates of a speedup. Tool programming and the author's steering of it cost unmetered time and tokens that may offset the execution-phase savings; the interface is validated on one simulator and one dual-arm platform, so portability is not established; and agent-written tools inherit the risk of vibe coding.

6 Conclusion

Code as policy with the agent in the loop pairs agent-written tools with feedback-driven tool selection through URAI, which humans share. On five RoboDojo tasks, four frozen execution agents succeed more often through URAI than under direct fingertip control, and deciding after each call succeeds more often than a program written in advance over the same tools; all four are also faster and use fewer output tokens. Whether these benefits hold for other tool collections and tasks, and once tool-authoring cost is counted, remains open.

AI USE STATEMENT

In this work, we used generative AI tools to help develop the conceptual framework, to design and give feedback on the evaluation methodology, to implement methods (the robot tools and the evaluation harness are written by coding agents, which is the subject of the paper), to clean and reformat datasets (extraction of the public leaderboard data), to interpret results, and to translate internal notes. We have not used generative AI tools to generate synthetic data sets, and formulating mathematical claims, proving them, or supporting qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to draft and edit parts of this paper, to create and edit scientific figures and software code, to identify and summarize related literature, to format references, and to propose the title. We have reviewed all AI-assisted work: every tool is validated in simulation or on the robot before it is used, outcomes of our automated simulation experiments use the benchmark's own success signal recorded by the harness, while human GUI outcomes are operator-reported, LLM-generated text and figures were checked against the underlying logs, sources, and photographs, and LLM-written code was reviewed and tested by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Reyna Abhyankar, Qi Qi, and Yiying Zhang. OSWorld-Human: Benchmarking the Efficiency of Computer-Use Agents, 2025. URL https://arxiv.org/abs/2506.16042.

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S: An Open Agentic Framework that Uses Computers Like a Human, 2024. URL https://arxiv.org/abs/2410.08164.

Diogo Almeida. Introducing System One Models & Jev. TypeSafe AI blog, 2026. URL https://typesafe.ai/blog/introducing-system-one-models-and-jev. Published September 15, 2026. Accessed September 21, 2026.

James E. Bobrow, Steven Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3–17, 1985.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.17126.

Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Haoran Lu, Weijie Wan, Baijun Chen, Songling Liu, Haowen Yan, Honghao Su, Zhiyang Dou, Kaixuan Wang, Dandan Zhang, Yunze Liu, Yan Qin, Qiwei Liang, Qiwei Wu, Zijian Lin, Wenwei Lin, Yuran Wang, Minghua He, Tianshu Wu, Ruihai Wu, Jingquan Zhou, Kai-Chong Lei, Haibao Yu, Yuanfeng Ji, Weiyang Jin, Guanyu Lin, Xiaofan Li, Qi Xiong, Renjing Xu, Zhongyu Li, Wenhao Chai, Enze Xie, Ziwei Wang, Yao Mu, Hao Dong, Wojciech Matusik, Mingyu Ding, Wenbo Ding, Ping Luo, and Masayoshi Tomizuka. RoboDojo: A Unified Sim-and-Real Benchmark for Comprehensive Evaluation of Generalist Robot Manipulation Policies, 2026a. URL https://arxiv.org/abs/2607.04434.

Yanzhe Chen, Zechen Bai, Zhijun Cao, Wenzheng Zeng, Kevin Qinghong Lin, Yiqi Lin, Guoqiang Liang, Kevin Yuchen Ma, Qiming Huang, and Mike Zheng Shou. Show-Harness: Just a VLM Agent Can Play Robots, 2026b. URL https://arxiv.org/abs/2609.10522.

Dongzhou Cheng, Taoran Yi, Ye Fang, Xingwu Zhang, Fan Feng, Yixuan Li, Gengxiong Zhuang, Rongze Wang, Shuai Yang, Wei Song, Weizhi Xue, Minyan Wu, Jie Gui, Jiaqi Wang, and Tong Wu. In-context robot learning with vlm agents, 2026. URL https://arxiv.org/abs/2609.19138.

Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots, 2024. URL https://arxiv.org/abs/2402.10329.

Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your Coding Agent is Secretly a Roboticist, 2026. URL https://arxiv.org/abs/2606.16458.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi "Jim" Fan. CaP-X: A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation, 2026. URL https://arxiv.org/abs/2603.22435.

Shijia Ge, Yijun Liu, Yinxin Zhang, Shuzhao Xie, Weixiang Zhang, Mingcai Zhou, and Zhi Wang. VGGT-DP: Generalizable Robot Control via Vision Foundation Models, 2026. URL https://arxiv.org/abs/2509.18778.

Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, and Ted Xiao. RT-Trajectory: Robotic Task Generalization via Hindsight Trajectory Sketches, 2023. URL https://arxiv.org/abs/2311.01977.

Meng-Hao Guo, Zhe-Han Mo, Jia-Jun Wang, Yi Zhang, Kejin Wang, Yi-Xuan Deng, Jia-Peng Zhang, Yongming Rao, and Shi-Min Hu. Transferring the intelligence of vlms to robotic control. *Technical Report*, 2026. URL https://robodawn.top/RoboDawn.pdf. Technical report PDF from the project page. Accessed September 21, 2026.

Hengyuan Hu, Priya Sundaresan, Jensen Gao, and Dorsa Sadigh. Via: Visual interface agent for robot control, 2026. URL https://arxiv.org/abs/2607.11119.

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. In *Conference on Robot Learning*, 2022. URL https://arxiv.org/abs/2207.05608.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 540–562. PMLR, 2023. URL https://proceedings.mlr.press/v229/huang23b.html.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. ReKep: Spatio-Temporal Reasoning of Relational Keypoint Constraints for Robotic Manipulation. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4573–4602. PMLR, 2025. URL https://proceedings.mlr.press/v270/huang25g.html.

Yuzhi Huang, Jie Wu, Weijue Bu, Ziyi Xiong, Gaoyang Jiang, Ye Li, Kangye Ji, Shuzhao Xie, Yue Huang, Chenglei Wu, Jingyan Jiang, and Zhi Wang. RoboStream: Weaving Spatio-Temporal Reasoning with Memory in Vision-Language Models for Robotics, 2026. URL https://arxiv.org/abs/2603.12939.

Yuka Iwanaga, Masayoshi Tsuchinaga, Kosei Tanada, Yuji Nakamura, Takemitsu Mori, and Takashi Yamamoto. Sketch Interface for Teleoperation of Mobile Manipulator to Enable Intuitive and Intended Operation: A Proof of Concept, 2025. URL https://arxiv.org/abs/2505.13931.

Mengzhao Jia, Yang Lin, Xixin Zhang, Zhihan Zhang, Xiaobai Liu, and Meng Jiang. Agent as Policy for Robotic Manipulation, 2026. URL https://arxiv.org/abs/2609.12541.

Kamal Kant and Steven W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72–89, 1986.

Andrej Karpathy. There's a new kind of coding I call "vibe coding". Post on X, February 2025. URL https://x.com/karpathy/status/1886192184808149383. Posted February 2, 2025. Accessed September 22, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Raymond Yu, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, Abhishek Gupta, and Ankit Goyal. HAMSTER: Hierarchical Action Models For Open-World Robot Manipulation, 2025. URL https://arxiv.org/abs/2502.05485.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: Language Model Programs for Embodied Control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://doi.org/10.1109/ICRA48891.2023.10160591.

Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. MOKA: Open-World Robotic Manipulation through Mark-Based Visual Prompting, 2024. URL https://arxiv.org/abs/2403.03174.

Yijun Liu, Yuwei Liu, Yuan Meng, Jieheng Zhang, Yuwei Zhou, Ye Li, Jiacheng Jiang, Kangye Ji, Shijia Ge, Zhi Wang, and Wenwu Zhu. Spatial Policy: Guiding Visuomotor Robotic Manipulation with Spatial-Aware Modeling and Reasoning, 2025. URL https://arxiv.org/abs/2508.15874.

Yuke Liu. RoboJEV: Two-Stage JEV Control of a Franka Panda in MuJoCo. GitHub repository, Apache-2.0, 2026. URL https://github.com/lykycy123/RoboJEV. Accessed September 21, 2026.

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 "Jim" Fan, and Guanzhi Wang. ASPIRE: Agentic /Skills Discovery for Robotics, 2026. URL https://arxiv.org/abs/2607.00272.

Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, Quan Vuong, Tingnan Zhang, Tsang-Wei Edward Lee, Kuang-Huei Lee, Peng Xu, Sean Kirmani, Yuke Zhu, Andy Zeng, Karol Hausman, Nicolas Heess, Chelsea Finn, Sergey Levine, and Brian Ichter. PIVOT: Iterative Visual Prompting Elicits Actionable Knowledge for VLMs, 2024. URL https://arxiv.org/abs/2402.07872.

OpenAI. Gpt-6 astra: A new generation of intelligence. Official model announcement, 2026. URL https://openai.com/index/gpt-6-astra/.

Hung Pham and Quang-Cuong Pham. A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Transactions on Robotics*, 34(3):645–659, 2018.

Thomas Ricouard. Architectural visualization with astra. OpenAI Developers case study, 2026. URL https://developers.openai.com/blog/architectural-visualization-with-astra.

Robocurve. GPT-6 Astra on robotic manipulation. Independent experimental report, September 2026. URL https://openai.robocurve.org/gpt-6-astra/. Published September 4, 2026. Accessed September 15, 2026.

RoboDojo Team. RoboDojo-Sim public leaderboard: per-task success rates. Leaderboard snapshot dated 2026-09-23, 2026. URL https://robodojo-benchmark.com/leaderboard. Accessed September 23, 2026.

Jiayi Su, Yixin Zheng, Mi Yan, Li Yi, Zhizheng Zhang, and He Wang. GPT 6 Astra as an embodied policy. Technical report and code, 2026. URL https://github.com/anonymous-report-421/eval-of-gpt-6-astra-as-policy. Companion report. Accessed September 15, 2026.

Priya Sundaresan, Hengyuan Hu, Quan Vuong, Jeannette Bohg, and Dorsa Sadigh. What's the Move? Hybrid Imitation Learning via Salient Points, 2024. URL https://arxiv.org/abs/2412.05426.

Huajie Tan, Peterson Co, Yijie Xu, Shanyu Rong, Yuheng Ji, Cheng Chi, Xiansheng Chen, Qiongyu Zhang, Zhongxia Zhao, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. ActionSketcher: From Reasoning to Action via Visual Sketches for Long-Horizon Robotic Manipulation, 2026. URL https://arxiv.org/abs/2601.01618.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2305.16291.

Maggie Wang, Lars Osterberg, Stephen Tian, Ola Shorinwa, Jiajun Wu, and Mac Schwager. InSight: Self-Guided Skill Acquisition via Steerable VLAs, 2026. URL https://arxiv.org/abs/2606.24884.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian "Max" Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi "Jim" Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic Robot Policy Self-Improvement in the Real World, 2026. URL https://arxiv.org/abs/2606.19980.

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, 2024. URL https://arxiv.org/abs/2404.07972.

Haoqi Yuan, Zhixuan Liang, Anzhe Chen, Ye Wang, Haoyang Li, Pei Lin, Yiyang Huang, Zixing Lei, Tong Zhang, Jiazhao Zhang, Jie Zhang, Jingyang Fan, Gengze Zhou, Qihang Peng, Chenxu Lv, Xiaoyue Chen, An Yang, Fei Huang, Junyang Lin, Dayiheng Liu, Jingren Zhou, Chenfei Wu, and Xiong-Hui Chen. Qwen-RobotManip Technical Report: Alignment Unlocks Scale for Robotic Manipulation Foundation Models, 2026. URL https://arxiv.org/abs/2606.17846.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung, Hao Peng, and Heng Ji. CRAFT: Customizing LLMs by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.17428.

Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation, 2018. URL https://arxiv.org/abs/1710.04615.

Wenbo Zhang, Kaixuan Wang, Yutao Ouyang, Xiaoyu Huang, Liyang Li, Kailun Su, Weiyang Jin, Wenhao Chai, Haotian Liang, Zhiyang Dou, Yue Chen, and Tianxing Chen. An Unexpected Robot Policy: Early Evaluations of GPT-6 Astra on RoboDojo and Beyond. RoboDojo project report, 2026. URL https://robodojo-benchmark.com/report/gpt-6-astra-eval. Site snapshot dated 2026-09-21. Accessed September 22, 2026.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware, 2023. URL https://arxiv.org/abs/2304.13705.

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, 2024. URL https://arxiv.org/abs/2307.13854.