SEQUENTIAL PLANNING VIA ANCHORED ROBOTIC KEYPOINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Changes in a robot's observations do not automatically imply changes in the task it is solving, but language-conditioned manipulation systems often fail to separate the two. If an object moves, for example, the locations at which actions should be executed may change while the underlying manipulation plan remains valid. We introduce Sequential Planning via Anchored Robotic Keypoints (SPARK), which separates a task's symbolic plan from its execution-time perceptual grounding, and we experimentally isolate whether a fixed symbolic plan can survive substantial changes to the scene through execution-time re-grounding. SPARK represents the task as a symbolic behavior tree whose object references are grounded to current 3D observations during execution, allowing the robot to update where actions are performed while preserving the plan. Runtime verification then enables local recovery and renewed grounding, only resorting to a new plan if necessary. To test the separation between plan and perceptual-grounding, we perform a set of manipulation experiments with a fixed plan while moving objects during execution. Under a 10 cm displacement, verification and adaptation maintain 66.7% task success, compared with 2.2% when disabled, demonstrating the importance of the underlying principle. Without task-specific model training, SPARK also achieves 72.6% mean success across six LIBERO-PRO settings and 68% across eleven physical task–embodiment settings on three robot platforms. Together, these results show that robustness does not require re-planning the task and for an important class of failures, it is sufficient to preserve the plan and update the system's perception of a changing world. Code and videos can be found at `https://longjohnsilvers22.github.io/spark_page/`

1 INTRODUCTION

General-purpose robot policies must generalize across two fundamentally different kinds of change: changes that alter what the robot should do, and changes that alter only where the same task must be executed. If a bowl is moved across a table, the robot's observations and required motions change, but the task "put the bowl on the plate" often does not. A capable manipulation system should preserve the task structure that remains valid while updating only the information that has become stale. This distinction is particularly important in open-world robotics, where the geometry of a scene can vary continuously even when the underlying task does not. Current vision-language-action models (VLAs) do not explicitly impose this separation. End-to-end models such as OpenVLA, π_0, and $\pi_{0.5}$ jointly map visual and linguistic observations to actions and achieve strong results on standard manipulation benchmarks (Kim et al., 2024; Black et al., 2024; Physical Intelligence et al., 2025). Yet recent robustness evaluations suggest that this performance can depend strongly on the specific scene and task distributions encountered during training. LIBERO-PRO (Zhou et al., 2025b), for example, reports severe degradation when object placements, instructions, manipulated objects, or environments are perturbed, despite high performance under the original LIBERO evaluation (Liu et al., 2023). A complementary mechanistic study finds that motor representations in several VLAs can be spatially bound to scene coordinates rather than representing task structure independently: transferring internal activations across tasks can steer execution toward the source trajectory and location (Grant et al., 2026). Together, these results expose a basic generalization problem: a change in the world state can invalidate the actions produced by a policy even when the underlying manipulation plan remains appropriate.

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.

One response is to introduce additional reasoning at test time. Code-generation agents such as CaP-Agent0 synthesize robot programs and use multi-turn interaction, execution feedback, visual differencing, and additional model queries to recover from failures (Fu et al., 2026). Such approaches show the value of decomposing manipulation, but they raise a more basic question: when an execution fails, what actually needs to be recomputed?

We argue that robust manipulation requires distinguishing information that should persist across environmental change from information that must be continually refreshed. A changed instruction may invalidate the task plan, while a moved object may instead invalidate only the perceptual grounding of an otherwise valid plan. On the other hand, a failed grasp or insertion may reflect neither, but rather a limitation of the available execution skill. Treating all of these cases as planning failures unnecessarily couples task reasoning to the continuous variability of the physical world. Instead, a robot should revise the level of representation that the change actually invalidates. Thus, when the task structure remains valid, renewed perception and local execution should be sufficient without solving the task again.

We introduce Sequential Planning via Anchored Robotic Keypoints (**SPARK**), a manipulation architecture built around this separation. SPARK separates a task's symbolic plan from the object locations used during execution. A language model first generates a symbolic behavior tree over a fixed set of robot skills. The resulting plan then specifies which objects to act on and which actions to perform, but it does not permanently bind those objects to particular coordinates. Instead, when an action is executed, SPARK detects the referenced object and grounds it to its current 3D location. After each action, the system checks whether the intended outcome was achieved. If not, it can re-detect the scene and retry the same action using the updated object location, while leaving the rest of the plan unchanged. Only when this local recovery fails does SPARK request a new plan. In this way, a change in object position need not become a new planning problem: the task structure can remain fixed while its connection to the current scene is updated.

We evaluate this separation through controlled object-displacement experiments, LIBERO-PRO benchmarks, physical robot experiments, and ablations of planning and perception. Our contributions include:

1. **Separating symbolic plans from execution-time grounding.** Our proposed approach, SPARK, represents a manipulation task as a symbolic behavior-tree while grounding its object references to current 3D observations only when the action is executed. Using runtime post-condition checks, this allows robust execution under changes in object position without requiring the symbolic plan to be regenerated. Additionally, without task-specific model training, SPARK achieves 72.6% mean success across six LIBERO-PRO settings and 68% across eleven physical task-embodiment settings on three robot platforms (§5.1,§5.4).
2. **Controlled evidence for recovery without replanning.** We introduce a scripted displacement protocol to test whether execution can recover when objects move during a task (LIBERO-PRO-Dyn, §5.2). With identical scripted plans, verification and adaptation achieve 66.7% success under 10 cm displacement, compared with 2.2% when disabled. This isolates a class of failures in which the task structure remains valid and recovery can occur by updating the plan's grounding rather than generating a new plan.
3. **Characterizing when additional planning and perception improve execution.** Through controlled ablations, we separately evaluate additional perceptual grounding and additional plan generation. Alternative object descriptions improve success on the LIBERO-PRO Spatial and Object suites by 27.7 and 10.0 percentage points, respectively, while generating a second candidate plan improves them by 11 and 12.5 points, with no statistically resolved improvement on the Goal suite (§5.3).

2 Problem Setup

We study language-conditioned manipulation under changes in object placement and task instructions. Given a natural-language instruction and camera observations, the robot must execute actions that achieve the specified goal. SPARK uses existing models and a fixed skill library without

task-specific model training. The evaluation protocols below distinguish whether information from earlier trials of the same task is available.

An *anchored robotic keypoint* is a 3D point associated with an object or part label through open-vocabulary detection. SAM3 segments the named object or part, such as a bowl or drawer handle, and the detection is backprojected through depth into world coordinates. The label identifies the target, while its coordinates can be updated through re-detection. When an object moves and the action sequence remains appropriate, the executor can therefore reuse the program with updated locations. Instruction changes may instead require different targets or actions. The keypoint specifies a spatial reference; the skill parameters and controller determine how the robot approaches and interacts with it.

We consider changes in object placement and task instructions at the start of an episode, as well as scripted object displacement during execution. The robot responds using observations collected within the episode, without updating model weights.

We distinguish *errors in the action sequence*, *errors in object grounding*, and *limitations of the available execution skills*. A planning error selects inappropriate actions or targets; a grounding error associates a target label with an incorrect object or location; an execution limitation prevents an otherwise appropriate action from succeeding. These causes can overlap. A failed post-condition identifies the unsuccessful step but does not, by itself, establish its root cause. Our experiments examine whether additional planning or perception improves success under these different limitations (§5.3).

We also distinguish evaluation without per-task adaptation from evaluation that uses information collected on earlier trials of the same task. In the former, no task-specific examples or memories from previous trials are available during evaluation. In the latter, earlier trials can inform subsequent execution through stored programs, examples, or other task-specific information. The headline SPARK evaluation disables retrieval and writeback of successful behavior trees; the plan-sampling experiment uses a live library and is identified separately. Access to simulator state or success signals is a separate protocol distinction and is disclosed in the experimental setup.

This paper asks one question: what manipulation capability can be achieved by combining structured programs, runtime checks, and existing foundation models without task-specific model training? SPARK does not use an end-to-end VLA executor, but its performance still depends on pretrained components, the skill library, and platform-specific control calibration.

3 Related Work

LLM-to-skill planning and BT synthesis. SayCan (Ahn et al., 2022) sequences pretrained skills from a flat menu. SPARK instead constrains the interface to a typed grammar and emits one behavior tree per task, executed with no learned policy in the loop. Prior LLM-to-BT systems established typed-grammar, single-call BT generation (Chen et al., 2024), and Contract-Grounded BT synthesis (Salfity et al., 2026) validates generated trees against robot-side contracts before deployment. SPARK adds the runtime half: perception-grounded post-conditions after every primitive, followed by LLM-free re-grounding when a check fails.

VLA fragility and learned executors. OpenVLA (Kim et al., 2024), π_0 (Black et al., 2024), and $\pi_{0.5}$ (Physical Intelligence et al., 2025) clear standard LIBERO and collapse under LIBERO-PRO position and task shift (§1), and MolmoAct2 (Fang et al., 2026) (97.2% on unperturbed LIBERO) averages 18.6% on the six perturbed cells, consistent with memorized poses.

Code-generation agents and recovery. Code-as-Policies (Liang et al., 2023) composes Python over robot APIs, and ProgPrompt (Singh et al., 2023) writes assert-style precondition checks into generated programs; SPARK instead checks primitive post-conditions against a re-detected scene at execution time. Behavior trees have carried postcondition-precondition-action templates since Colledanchise & Ögren (2018), and Code-as-Monitor (Zhou et al., 2025a) compiles constraints into visual programs that watch for failure. Our claim is narrower: typing the check to the primitive that declares it makes each failure attributable to one step. CaP-Agent0 (Fu et al., 2026) extends the line to multi-turn synthesis with a three-model ensemble and visual differencing. ASPIRE (Lu et al.,

2026) compounds a frozen coding agent's repairs into a skill library, learning on development seeds disjoint from the seeds it is scored on, the per-task-adapted regime of §2. ReKep (Huang et al., 2024) shares the keypoint-grounded, training-free structure but re-optimizes an implicit trajectory at $\sim$10 Hz for one robot.

Runtime verification and recovery. ANCHOR (Jiang et al., 2026) and U-LAG (J H & Muraleedharan, 2025) add runtime checks above a fixed executor; U-LAG pays a model call for the verdict, ANCHOR computes it from calibrated geometry, and SPARK reads it from typed predicates. Harness VLA (Zhang et al., 2026c) lifts semantic grounding and non-contact execution to an agentic planner while retaining a frozen VLA for contact-rich manipulation, and reports the strongest LIBERO-PRO result we know of: 82.4% with a Claude Code planner and 72.1% with Codex. That headline is few-shot: the agent bootstraps each task on a reference seed, stores the successful primitive trace, and re-grounds it at evaluation, the per-task-adapted regime of §2. Its zero-shot regime withholds that memory and still reaches 79.0 on goal instruction-redirection against 31.0 on position-swap; the spread measures what a frozen contact-rich policy supplies and analytic primitives do not. Its executor is a $\pi_{0.5}$-SFT checkpoint trained on robot trajectories, so the comparison turns on training-free execution against pretrained execution.

Perturbation benchmarks and test-time compute. LIBERO-PRO (Zhou et al., 2025b) perturbs only the initial state. Static perturbation separates methods weakly, so we also script displacement inside the episode. CoRe (Zhang et al., 2026b) first scripts phase-indexed in-episode perturbation and recovers a frozen policy at inference time with no failure data; we use the same axis to compare a typed post-condition with a learned recovery channel. RoboMonkey (Kwok et al., 2025) samples at the action layer and CaP-Agent0 at the plan layer; SPARK aims at perception, where the shift moves what the words point to while the plan stays valid (§5.3). Concurrently, RATS (Zhang et al., 2026a) spends its budget before the task arrives, distilling $\sim$30M tokens of self-directed play into a retrievable skill library, and still writes a program per trial at test time; SPARK writes none, so its hundredth task costs what its first did in test-time calls.

4 METHOD

SPARK has four components: SAM3 grounds the scene from the platform cameras, Gemini composes a typed behavior tree (BT) over a library of composable skills, an inverse-kinematics (IK) robot controller executes each action, and a tiered recovery layer handles failed execution checks (Figure 1). The behavior tree acts as a music score: primitive actions are the notes, their parameters specify how they are performed, and arm assignments coordinate the parts in single arm and bimanual tasks. The robot follows this score while adapting execution through perception. Specifically, each action refers to an object by label, which is resolved to a current 3D position when the action executes. Updated detections therefore allow the robot to retry an action without rewriting the score; a new plan is requested only after the execution attempt is exhausted.

Multi-camera perception. Single-arm platforms (UR10e, FR3) use a fixed bird's-eye camera, a fixed side camera, and a wrist camera, all RGB-D at 640×480; the bimanual workcell replaces the side camera with a rear camera plus a wrist camera per arm. Simulation uses LIBERO-PRO's default rig (`agentview` and `robot0_eye_in_hand`), with no custom placement. SAM3 (Carion et al., 2025), an open-vocabulary segmentation model, produces masks from text prompts. The mask centroid and median depth are backprojected into world coordinates using the camera calibration. The resulting detection map associates each object or part label with a 3D location. During re-detection, these locations are updated while the behavior tree continues to reference the same labels.

Behavior-tree planning. Given the annotated image (SAM3 masks and labels), detected 3D keypoints, and task instruction, Gemini emits a YAML BT in one call, at temperature 0 in simulation and 0.3 on hardware (per-program model versions in the Appendix). Each action node specifies a registered skill and its parameters, and the tree is parsed and type-checked before execution. Five base primitives (`move_to_keypoint`, `move_relative`, `grasp`, `release`, `wait`) cover basic motion operations. More than thirty registered skills additionally provide calibrated closed-loop routines, including force-feedback descent, contact detection, and local retry logic. Gemini selects and sequences these skills and supplies their parameters; the executor implements the underlying

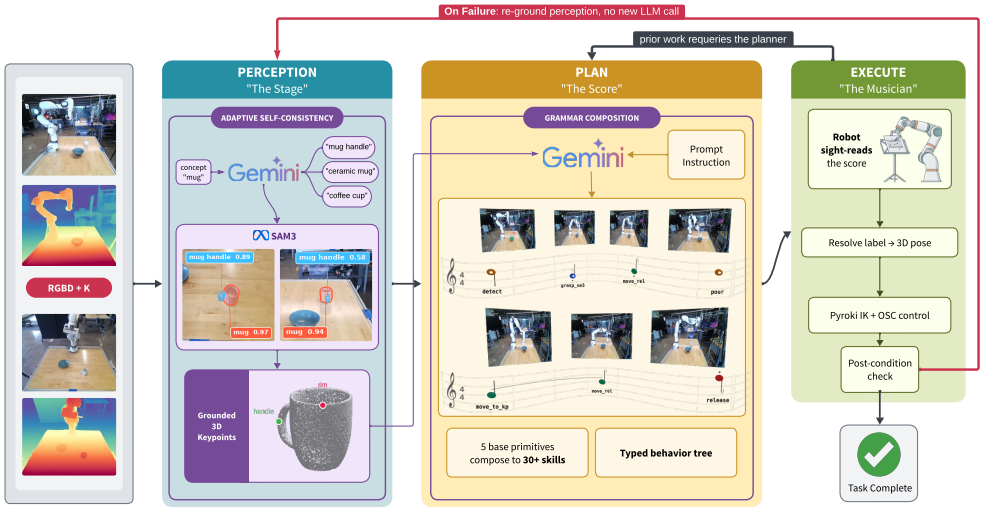


Figure 1: SPARK separates the task's score from its execution. Left: perception locates objects as 3D keypoints; the adaptive configuration uses an additional Gemini call to propose alternative object descriptions. Center: one language-model call writes the score, a behavior tree whose notes are skills from a fixed library. Right: the robot executes these skills using current object locations and checks each action's post-condition. Failed checks trigger local correction or renewed detection along the red recovery path, allowing execution to continue without another planning call. A new plan is requested only after the execution attempt is exhausted.

control. This interface keeps coordinate transformations and low-level control inside the executor, reducing the implementation details that code-generation systems like CaP-Agent0 must generate. The skills were written with Claude Code against the typed contract at build time and reviewed by the authors before entering the registry, which is fixed at the pinned commit of each evaluation, and no skill was added or edited during any reported run. This places SPARK with Harness VLA (Zhang et al., 2026c) and Contract-Grounded BT synthesis (Salfity et al., 2026), which also write no code at test time; CaP-Agent0 writes a program per trial, RATS writes one per trial from a library filled during play, and ASPIRE (Lu et al., 2026) writes and repairs one program per task on development seeds before evaluation (§2).

Successful BTs can be stored and supplied as in-context examples in subsequent trials, without updating model weights. This option is disabled in the headline evaluation and enabled in the plan-sampling experiment §5; each experiment specifies its library configuration.

Tiered recovery. Each skill evaluates a post-condition using its available observations or execution signals. A failed check identifies the unsuccessful step and triggers a three-tier recovery procedure: (1) an in-place correction, such as a small motion and grasp adjustment; (2) perception re-grounding, which retracts the arm by 10 cm, re-runs SAM3, and retries the same plan using updated object locations; (3) plan regeneration through a new Gemini call after the execution attempt is exhausted. The first two tiers preserve the existing behavior tree and require no additional LLM call. This ordering prioritizes local correction before changing the action sequence; it does not assume that a failed check uniquely identifies the underlying cause.

Alternative-prompt perception. A single SAM3 prompt often misses or produces an incomplete mask, which is identified as the limiting factor on overall success rate in our failure logs. We therefore apply self-consistency (Wang et al., 2023) to perceptual grounding: one of several prompts usually recovers the object, worth $+27.7$ points on the spatial suite and $+10.0$ on object in the matched ablation (§5.3). One additional Gemini call per trial proposes three alternative prompts ($K=3$) per object, varying color, shape, and material descriptors. SAM3 evaluates these descriptions, and the system selects the highest-confidence detection for each object, with conflicts resolved during planning. Recovery reruns detection using the existing prompt set rather than requesting new

Table 1: LIBERO-PRO success rates (%) over six position-and-task settings. Pos changes initial object placements; Task changes instructions, targets, relations, or actions depending on the suite. The SPARK row uses 10 tasks × 50 trials per setting, stored successful programs disabled, no BDDL hints, and the simulator-access protocol in the setup. OpenVLA, π_0, $\pi_{0.5}$, and the original CaP-Agent0 rows are reproduced from Fu et al. (2026); CaP-Agent0 (RATS eval) and RATS are from Zhang et al. (2026a); MolmoAct2 is our evaluation of the released LIBERO checkpoint. Intervals are 95% confidence intervals estimated by bootstrap resampling over tasks. Bold best, underline second-best. †Three goal tasks read model part poses; excluding them the unassisted mean is 71.7%. ‡The configuration every earlier number in this program used: the executor reads the simulator's success predicate mid-episode; the unassisted row removes every such read, same commit and model.

	Object		Goal†		Spatial		
Method	Pos	Task	Pos	Task	Pos	Task	Mean
OpenVLA (Kim et al., 2024)	0	0	0	0	0	0	0.0
π_0 (Black et al., 2024)	0	0	0	0	0	0	0.0
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	17	1	38	0	20	1	12.8
MolmoAct2 (Fang et al., 2026)	47.2	0.0	29.0	12.0	23.0	0.4	18.6
CaP-Agent0 (Fu et al., 2026)	22	18	26	17	12	14	18.2
CaP-Agent0 (RATS eval) (Zhang et al., 2026a)	27.0	31.0	29.0	16.0	13.0	23.0	23.2
RATS (Zhang et al., 2026a)	<u>61.0</u>	<u>63.0</u>	<u>43.0</u>	<u>36.0</u>	29.0	31.0	<u>43.8</u>
SPARK (**Ours**), unassisted	**85.0**	**85.6**	**59.6**	**60.2**	**72.8**	**72.6**	**72.6**
95% CI	[75, 94]	[77, 94]	[35, 83]	[36, 83]	[54, 87]	[53, 88]	
SPARK, verifier-assisted‡	84.8	86.2	62.6	61.2	74.8	71.4	73.5

descriptions. Physical experiments use single-prompt detection, so they evaluate the planning and recovery framework without this additional prompt-generation step (§5.4).

5 Experiments

Setup. We evaluate SPARK on LIBERO-PRO and CaP-Bench, under object displacement during execution, and on three physical robot platforms. Separate ablation studies examine alternative detection descriptions and additional candidate plans. The LIBERO-PRO evaluation reported in Table 1 uses the six position-and-task settings of Zhou et al. (2025b) as evaluated by Fu et al. (2026): three suites (object, goal, spatial), two perturbation types per suite, ten tasks per setting, and 50 trials per task. Position perturbations change initial object placements; task perturbations change instruction wording, target objects or relations, or required actions, depending on the suite. The CaP-Bench evaluation includes seven Robosuite tasks with 100 trials per task. Across experiments, SPARK uses SAM3 detection, simulator depth in simulation, hardware depth on robots, and fixed controller and IK settings per platform. Simulation follows the S2 perception protocol of CaP-Agent0: no ground-truth object lookup for detection, except where disclosed. The experiments differ in their access to simulator information and stored successful programs. During simulation, SPARK uses the simulator's success signal to end completed tasks, stop selected push and drawer actions, and decide whether to retry. Three goal tasks also use object part poses from the simulator model; excluding those tasks gives a 71.7% mean. Table 1 reports the configuration with every such read removed (`SPARK_NO_ORACLE=1`) as the headline; the assisted configuration is the second row, and the two means differ by 0.9 points. In Table 1, stored successful programs are neither retrieved nor updated. The in-process BT cache is off throughout. The ablations use different model and executor configurations from the Table 1 evaluation; comparisons are controlled within each study, and absolute success rates should not be compared across studies. Appendices give provenance, individual-task results, cost accounting, physical logs, and failure analyses.

5.1 Performance Under Position and Instruction Changes

The complete system handles initial-state and instruction changes at 72.6% mean success across the six LIBERO-PRO settings with no simulator signal read during the episode, 28.8 percentage points above RATS at 43.8% (Table 1). The spatial-suite mean is 73.1%, and the goal-suite mean

Table 2: Success under object displacement during execution. Each entry reports successful trials out of 45, covering three tasks, three displacement phases, and five initial states. Objects move over 40 simulator steps. Both conditions execute identical scripted plans, with verification and adaptation enabled or disabled together. Full results appear in Appendix §M, Table 16.

Verification and adaptation	0 cm	2 cm	5 cm	10 cm
Enabled	39/45	37/45	35/45	30/45
Disabled	33/45	26/45	7/45	1/45

is 61.9%. These numbers show system-level performance, they do not isolate the contribution of program structure from perception quality, controller tuning, or simulator access. No task in either goal setting scores zero across fifty trials, but the remaining losses concentrate in plate-push and rack-insertion tasks (Appendix §J, Table 15).

We also evaluate SPARK on seven CaP-Bench tasks. SPARK averages 63.4% success, compared with 68.3% for CaP-Agent0's three-model ensemble (Appendix §C, Table 9). SPARK succeeds in 100% of Lift and CubeRestack trials and 97% of Stack trials, while success falls to 60% for Wipe, 63% for TwoArmLift, and 0% for NutAssemblySquare. A separate ablation tests whether the planner can complete tasks when a dedicated skill is removed. Using only base primitives, stacking succeeds in 10/10 trials and wiping in 0/20 trials (Appendix §E, Table 11). The wiping routine requires force and coverage control provided by the dedicated skill.

5.2 Recovering When Objects Move

Execution checks and recovery preserve success after object displacement without generating a new plan. We execute identical scripted plans with verification and adaptation enabled or disabled, using three object-suite tasks, three injection phases, and five initial states, for 45 trials at each displacement magnitude. The object slides over 40 simulator steps by 0–10 cm on the same seeded schedule for both conditions; the planner is not in the loop.

At 10 cm, success is 30/45 (66.7%) with verification and adaptation, compared with 1/45 (2.2%) when both are disabled (Table 2). At 0 cm the corresponding counts are 39/45 and 33/45. The difference increases from six successful trials at 0 cm to twenty-nine at 10 cm. Because the plans are fixed, these results show that execution checks and recovery can preserve success while retaining the same plan. Verification and adaptation change together, and the study does not test language-model plan generation. Results for instantaneous displacement, including the VLA comparisons, appear separately in Appendix §M, Table 16. Appendix §M also reports detection and recovery latencies for both displacement protocols.

5.3 When Additional Perception or Planning Helps

Alternative detection descriptions. Alternative detection prompts help spatial and object tasks, but can reduce success on goal-task changes. The planner and controller are held fixed. The detection stage uses either one description per object or several alternative descriptions generated once per trial. The system selects detections using SAM3 confidence scores. The object and spatial columns in Table 3 are two-setting means; the goal-task column is one perturbation setting.

Table 3: Effect of alternative object descriptions on task success (%). "Single description" is the task-language-only S2 protocol; "Alternative descriptions" adds prompt variants. Object and Spatial average position and task perturbations. Goal-task reports the task perturbation setting only. Overall averages all six settings, including the goal-position result in Appendix §L. Each underlying setting includes ten tasks with 50 trials per task. Differences are in percentage points.

Config	Object mean	Spatial mean	Goal-task	Overall
Single description	29.9	36.5	22.4	31.9
Alternative descriptions	39.9	64.2	14.0	43.7
Δ	+10.0	+27.7	−8.4	+11.8

Table 4: Effect of generating two candidate plans on task success (%). Each condition includes 600 trials across three suites and two perturbation types. The second condition selects between two candidates using success checks during trial executions in simulation. Intervals are 95% confidence intervals estimated by bootstrap resampling over tasks.

	Spatial		Object		Goal		
Condition	Pos	Task	Pos	Task	Pos	Task	Mean
Plan once	43.0	82.0	57.0	63.0	37.0	25.0	51.2
95% CI	[22, 63]	[67, 95]	[35, 77]	[39, 86]	[14, 61]	[5, 48]	[42, 60]
Two plans, oracle selection	60.0	87.0	67.0	78.0	39.0	26.0	59.5
95% CI	[35, 83]	[72, 99]	[39, 91]	[58, 93]	[14, 66]	[5, 50]	[50, 68]



Figure 2: Examples of tool use, folding, and object manipulation on UR10e, Franka FR3, and bimanual Franka platforms. Appendix §H reports success rates and 95% Wilson confidence intervals for all eleven task and platform combinations.

Adding alternative descriptions raises spatial success from 36.5% to 64.2% (+27.7 points) and object success from 29.9% to 39.9% (+10.0 points). Goal-task success falls from 22.4% to 14.0%. The decrease on goal-task perturbations shows that additional descriptions can also reduce task success.

Additional plan candidates. Generating a second candidate plan improves spatial and object success, but does not yield a statistically resolved improvement on the goal suite. Each condition runs 600 trials, for 1,200 total: ten tasks, ten initial states, two perturbations, and three suites. The baseline plans once per trial. The sampling condition generates two candidate plans and selects one using simulator success checks during trial executions in simulation, an oracle selector unavailable as such on hardware. Successful programs from earlier trials are available to later trials in this study. Although task order is identical, the two conditions may accumulate different stored examples.

Mean success rises from 51.2% to 59.5% (Table 4). The spatial mean rises from 62.5% to 73.5% (+11.0 points; 95% confidence interval $[+6.0, +17.0]$), and the object mean rises from 60.0% to 72.5% (+12.5 points; $[+0.5, +26.0]$). The estimated gain on the goal suite is 1.5 percentage points, with a 95% confidence interval from -4.5 to 9.5. Generating two plans also increases the reported LLM cost per success from \$0.257 to \$0.282 (detector cost and candidate-execution compute excluded; Appendix §F).

5.4 Physical Experiments

The same BT grammar, Gemini planning call, and SAM3 pipeline run on UR10e, Franka FR3, and a bimanual Franka setup, without task-specific model training. All physical runs use hardware depth and single-description perception; the alternative-description ablation above is not used on hardware. Across eleven task-platform combinations, each with twenty randomized trials, SPARK succeeds in 150 of 220 trials (68.2%; Appendix §H). Figure 2 illustrates representative tasks. Each platform uses its own IK solver, gripper interface, workspace limits, and contact-control parameters.

The same utensils-in-tray task runs on two platforms: UR10e succeeds in $18/20$ trials (90%, 95% Wilson interval $[70, 97]\%$), and FR3 succeeds in $16/20$ (80%, $[58, 92]\%$). This supports execution through shared planning and skill interfaces, while the small trial counts leave substantial uncertainty about platform differences. On the bimanual Franka platform, sleeve folding succeeds in $15/20$ trials, while the complete three-fold sequence succeeds in $6/20$.

6 Discussion

An object displacement can invalidate the coordinates used by an action while leaving the action sequence appropriate. SPARK makes this distinction explicit by resolving object labels during execution and attempting recovery within the existing behavior tree. With identical scripted plans, verification and adaptation achieve $30/45$ successes at $10\,\text{cm}$, compared with $1/45$ when both are disabled (Table 2; Appendix §M, Table 16). Because verification and adaptation change together, this experiment does not isolate their individual effects. It nevertheless shows that some failed executions can be recovered by correcting where an action is performed while retaining its action sequence. The study does not establish that local recovery is preferable for failures that invalidate the plan itself, nor does it compare it with replanning strategies.

Maintaining a plan's connection to the scene also depends on observations that identify relevant objects and on skills that can execute the requested actions. Alternative detection descriptions increase mean spatial-suite success by 27.7 percentage points but decrease goal-task success by 8.4 points (Table 3; Appendix §L). These task outcomes show that added descriptions have setting-dependent effects; they do not independently measure detection accuracy. Generating two candidate plans also improves some settings (Table 4), but the experiment uses simulator-based oracle selection and finds no statistically resolved improvement on the goal suite. The perception and planning studies use separate configurations and are not an equal-budget comparison. The skill ablation illustrates a different limitation: the planner reconstructs stacking from base primitives in all ten trials, whereas wiping fails in all twenty trials without the dedicated control routine (Appendix §E, Table 11). Sequences of available actions can therefore express some kinematic skills, while contact regulation and coverage require capabilities in the executor. A key question is how execution feedback should select an additional observation, candidate plan, or control routine.

Limitations. The evaluation conditions bound these conclusions. The LIBERO-PRO evaluation reads no simulator success signal during the episode; three goal tasks still take drawer-bar and knob-lever geometry from the simulator model, and excluding them lowers the mean from 72.6% to 71.7%. The ablation results apply to their respective configurations, and controller and primitive development used LIBERO-PRO tasks, so disabling stored programs during evaluation does not remove design-time benchmark exposure. The displacement study uses scripted plans and does not establish reliable grasping of continuously moving targets. Physical execution requires platform-specific tuning, and twenty trials per setting limit the precision and breadth of the hardware evidence.

7 Conclusion

SPARK separates a symbolic manipulation plan from the current object locations used to execute it. A language model generates a behavior tree over a fixed skill library, and the executor resolves object labels, checks each step, and attempts local recovery before requesting a new plan. Without task-specific model training, SPARK achieves 72.6% mean success across six LIBERO-PRO settings with no simulator signal read during the episode and 68% across eleven physical task and platform combinations, with platform-specific control tuning. These findings support treating recovery within a valid plan as an explicit part of manipulation system design. Future work should verify from onboard observations alone, strengthen contact and articulated control, and let execution feedback choose between renewed perception, local correction, and replanning.

AI use statement

In this work, we used generative AI tools (Claude and ChatGPT) for implementing and testing primitives against the typed skill contract and analysis of failure modes. We also used generative

AI tools for polishing the writing, including word refinement and grammar correction. We did not use generative AI tools for generating data, formulating mathematical claims & hypotheses, writing proofs, or interpreting results. The remaining required-disclosure tasks under the ICLR generative AI policy are not applicable to this work. We have reviewed all AI-assisted work: all AI-assisted code was tested by the authors before use, every reported number comes from runs the authors configured, executed, and inspected, and all AI-assisted text was reviewed and revised by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

Every reported number is tied to a named program (headline, plan-sampling, displacement, physical), each pinned to a git commit, model version, and protocol flags stated where the number is used and in Appendix §J, which also lists per-task success counts for every cell of Table 1. The LIBERO-PRO protocol (task lists, trials per task, perception protocol, simulator-signal access, and the unassisted configuration) is specified in §5; the LIBERO-Dyn injector, displacement schedules, and stop rules are in Appendix §M; the behavior-tree grammar and skill slots are in Appendix §D, with two complete example trees in Appendix §I and implementation settings in Appendix §A; physical task definitions, trial logs, and per-cell intervals are in Appendix §H; LLM call accounting is in Appendix §F; the CaP-Bench protocol and its reproduction record are in Appendix §C. Source code, the benchmark runners, the displacement injector, and the run configuration files will be released with the camera-ready.

References

Michael Ahn, Anthony Brohan, Noah Brown, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022. arXiv:2204.01691.

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. Robotics: Science and Systems (RSS) 2025.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, et al. SAM 3: Segment anything with concepts. *arXiv preprint arXiv:2511.16719*, 2025.

Xinglin Chen, Yishuai Cai, Yunxin Mao, et al. Integrating intent understanding and optimal behavior planning for behavior tree generation from human instructions. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 6832–6840, 2024. doi: 10.24963/ijcai.2024/755.

Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2018.

Haoquan Fang, Jiafei Duan, Donovan Clay, et al. MolmoAct2: Action reasoning models for real-world deployment. *arXiv preprint arXiv:2605.02881*, 2026.

Letian Fu, Justin Yu, Karim El-Refai, et al. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026.

Bryce Grant, Xijia Zhao, and Peng Wang. Not all features are created equal: A mechanistic study of vision-language-action models. *arXiv preprint arXiv:2603.19233*, 2026. Multimodal Intelligence Workshop @ ICLR.

Jianing Guo, Zhenhong Wu, Chang Tu, et al. On robustness of vision-language-action model against multi-modal perturbations. In *International Conference on Learning Representations (ICLR)*, 2026.

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

Anamika J H and Anujith Muraleedharan. U-LAG: Uncertainty-aware, lag-adaptive goal retargeting for robotic manipulation. *arXiv preprint arXiv:2510.02526*, 2025.

Jinhao Jiang, Shengyu Fang, Sibo Zuo, et al. ANCHOR: A physically grounded closed-loop framework for robust home-service mobile manipulation. *arXiv preprint arXiv:2604.25323*, 2026.

Chung Min Kim, Brent Yi, Hongsuk Choi, et al. PyRoki: A modular toolkit for robot kinematic optimization. *arXiv preprint arXiv:2505.03728*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, et al. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning (CoRL)*, pp. 2679–2713, 2024. arXiv:2406.09246.

Jacky Kwok, Christopher Agia, Rohan Sinha, et al. RoboMonkey: Scaling test-time sampling and verification for vision-language-action models. In *Conference on Robot Learning (CoRL)*, pp. 3200–3217, 2025. arXiv:2506.17811.

Jacky Liang, Wenlong Huang, Fei Xia, et al. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. arXiv:2209.07753.

Byeongdo Lim, Jongmin Kim, Jihwan Kim, et al. EquiGraspFlow: SE(3)-equivariant 6-DoF grasp pose generative flows. In *Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pp. 5067–5086. PMLR, 2024.

Haotong Lin, Sili Chen, Junhao Liew, et al. Depth anything 3: Recovering the visual space from any views. *arXiv preprint arXiv:2511.10647*, 2025.

Bo Liu, Yifeng Zhu, Chongkai Gao, et al. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. arXiv:2306.03310.

Hanqing Liu et al. Eva-VLA: Evaluating vision-language-action models' robustness under real-world physical variations. *arXiv preprint arXiv:2509.18953*, 2025. Submitted 23 Sep 2025, v2 15 Mar 2026; OpenVLA 90% fail on LIBERO-Long under worst-case.

Runyu Lu, Yubo Wu, Ethan Kou, et al. ASPIRE: Agentic /Skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026.

Physical Intelligence, Kevin Black, Noah Brown, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Wilbert Pumacay, Ishika Singh, Jiafei Duan, et al. THE COLOSSEUM: A benchmark for evaluating generalization for robotic manipulation. In *Robotics: Science and Systems (RSS)*, 2024. arXiv:2402.08191.

Jonathan Salfity, Robert Blake Anderson, and Mitch Pryor. Contract-grounded behavior tree synthesis via coding agents. *arXiv preprint arXiv:2607.12220*, 2026.

Ishika Singh, Valts Blukis, Arsalan Mousavian, et al. ProgPrompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. arXiv:2209.11302.

Peidong Wang, Zhiming Ma, Ying Chang, Xufang Luo, Xiaocui Yang, Shi Feng, Yuqing Yang, and Dongsheng Li. Self-evolving embodied agents via skill-harness evolution. *arXiv preprint arXiv:2608.11350*, 2026. Method named SHAPER.

Xuezhi Wang, Jason Wei, Dale Schuurmans, et al. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2203.11171.

Wenli Xiao, Jia Xie, Tonghe Zhang, et al. ENPIRE: Agentic robot policy self-improvement in the real world. *Conference on Robot Learning (CoRL)*, 2026. arXiv:2606.19980.

Yuhan Xie, Yuping Yan, Yunqi Zhao, et al. STRONG-VLA: Decoupled robustness learning for vision-language-action models under multimodal perturbations. *arXiv preprint arXiv:2604.10055*, 2026.

Junyi Zhang, Jiaxin Ge, Hanjun Yoo, et al. Playful agentic robot learning. *Conference on Robot Learning (CoRL)*, 2026a. arXiv:2606.19419.

Yanyan Zhang, Disheng Liu, Kai Ye, Chaoda Song, Xinpeng Li, Mohsen Hariri, Vikash Singh, Yu Yin, and Vipin Chaudhary. Imagining recovery: Inference-time counterfactual realignment for vision-language-action models. *arXiv preprint arXiv:2608.14822*, 2026b.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Zhengru Fang, Yi Nie, 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*, 2026c.

Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, and He Wang. Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025a.

Xueyang Zhou, Yangming Xu, Guiyao Tie, et al. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025b.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, et al. Robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

A Implementation Details

We document the per-component settings sufficient to reproduce SPARK on LIBERO-PRO and CaP-Bench. All values are the defaults used in the experiments of §5. No per-task tuning was performed beyond what is reported below. Figure 3 shows the camera rigs for the three physical platforms.

A.1 SAM3 Detection

We use the public SAM3 (Carion et al., 2025) checkpoint via its open-vocabulary text-prompted API. Detections are accepted at the model's default confidence threshold. Inside the adaptive configuration, the variant-selection rule of §4 (keep the phrasing that yields a single confident detection) supersedes hard thresholding. Prompts are short noun phrases (1–6 tokens) emitted by Gemini from the task language string. The *+BDDL names* configuration additionally uses the LIBERO canonical-name dictionary. Both the agentview (640×480) and the `robot0_eye_in_hand` wrist view (640×480) are queried with the same prompt set on each perception cycle.

A.2 Gemini (BT Generator and Prompt Synthesiser)

In simulation, the main-table behavior-tree call uses Gemini 3.7 Flash (`gemini-3.7-flash`); the matched-budget program is pinned to `gemini-3.1-pro-preview` at temperature 0 (its diversity samples at 0.7; the bare `gemini-3.1-pro` identifier no longer resolves). Physical experiments use Gemini 3.5 Flash (`gemini-3.5-flash`) at temperature 0.3. Both use `response_mime_type="application/json"` for the BT-generation call and unstructured text for the prompt-synthesis call. `max_output_tokens` is set to 2048 for BT generation and 512 for prompt synthesis; both calls fit comfortably under these caps in practice. The system prompt for BT generation specifies the primitive grammar of §4, the required JSON-of-YAML output shape, and a one-line invariant ("do not hallucinate actions outside the listed grammar"). The prompt-synthesis call asks for three color-and-shape prompts for each object and returns a JSON list of strings.

A.3 Depth Sources

Simulation evaluations use the renderer's ground-truth depth. Physical experiments use hardware time-of-flight depth from two Azure Kinect DK cameras (one sideview master, one birdview subordinate) via `pyk4a`. The wrist RealSense D435I provides supplementary stereo depth for close-range grasp refinement. DA3 (Lin et al., 2025) is available as a monocular fallback but is not used in the reported experiments. Initial multi-view experiments showed poor cross-view agreement.

A.4 Pyroki IK Settings

Orientation-constrained moves (drawer pulls, plate-rim grasps) use Pyroki (Kim et al., 2025), a JAX-based constrained 6-DOF solver, on the LIBERO platform. On the Robosuite/CaP-Bench platform the `JOINT_POSITION` controller swap Pyroki needs is unavailable (the simulator exposes a composite controller), so Cartesian moves there run on OSC; this is why the contact-rich nut insertion is bounded by OSC's pose-control-under-contact limit: full-pose control clears it (§C), but this platform's composite controller precludes the joint-position interface that control needs. We measured the `panda_hand_tcp` link to be offset by 6.9 mm in z from LIBERO's `grip_site`; this correction is applied at solve time. Default Pyroki weights are used for position (w_{pos}=1.0) and orientation (w_{ori}=0.5); joint-limit and self-collision constraints are enabled on the Franka chain. Unconstrained moves fall back to MuJoCo's Jacobian-pseudoinverse for speed.

A.5 OSC Controller

Cartesian moves use the mass-matrix operational-space controller from Robosuite (Zhu et al., 2020) with the default `OSC_POSE` action space. Translational stiffness is 300 N/m, rotational stiffness 50 Nm/rad, with critically-damped gains; settling time on a 5 cm translation is below 0.5 s. Joint-space fallback uses the Robosuite `JOINT_POSITION` controller with a per-step delta cap of 0.1 rad

enforced by the controller layer independently of the LLM. TCP-delta limits in the real-robot pipeline are 0.02 m/step.

A.6 BT EXECUTOR

Episode scoring is a single terminal `env.check_success()` read after execution ends; success is never latched mid-episode. The disclosed mid-episode control reads only ever stop execution at an achieved-success state or prune redundant retries, which is what grounds the claim that an unassisted run cannot score higher in expectation.

Each primitive returns a boolean post-condition (e.g.`grasp` verifies that the gripper finger gap is below a contact threshold). A failed post-condition triggers the recovery layer (§4), which is capped at two retries before fail-closed. Per-primitive timeouts: `move_to_keypoint` 4 s, `move_relative` 2 s, `grasp`/`release` 1 s, `insert` 6 s, `push_object` 4 s, `open_drawer` 5 s, `wipe` 10 s. A timeout triggers the recovery layer and does not terminate the trial. Consecutive Cartesian waypoints within a plan are concatenated into a single Ruckig time-optimal trajectory (*legato* execution) in place of separate point-to-point moves, so a multi-waypoint motion runs as one smooth, continuous path through the waypoints.

B LIBERO-PRO PER-TASK RESULTS (ABLATION BUILD)

These are the per-task vectors of the ablation build, the earlier build on which the perception ablation and the plan-sampling arms ran, whose suite means the perception-sourcing ablation table reports; the headline-build vectors for Table 1 are in the run-provenance section later in this appendix.

Figure 4 shows representative scenes from each LIBERO-PRO suite. Tables 5–7 give per-task SPARK (adaptive) success rates across the spatial, object, and goal suites of LIBERO-PRO (50 trials per task), and Figure 5 contrasts them with MolmoAct2.

Table 8 gives the corresponding MolmoAct2-LIBERO rates from our evaluation, as no published LIBERO-PRO results exist for MolmoAct2.

Table 5: SPARK adaptive: spatial suite per-task success rates (%, 50 trials).

Task	Pos	Task
between plate & ramekin	64	98
next to ramekin	100	72
from table center	86	40
on cookie box	42	74
in top drawer	0	70
on ramekin	94	66
next to cookie box	70	82
on stove	20	56
next to plate	28	90
on wooden cabinet	56	76
Mean	56.0	72.4

C CAP-BENCH PER-TASK DETAILS

We document the BT used for each of the seven CaP-Bench tasks, the number of trials, the success criterion, and the dominant failure mode observed. All SPARK and CaP-Agent0 numbers in this section use 100 trials per task; RATS numbers use its 50-trial RoboSuite skill-transfer evaluation. The SPARK column comes from the CaP-Bench build of the runner (`run_spark_robosuite` with Gemini 3 Flash preview, robosuite 1.5.2, and the Pyroki joint-space descent), which predates the headline build and is reported as measured on the CaP-Bench build. A re-run on the headline build and model (Gemini 3.7 Flash, 100 trials per task) reproduces the single-arm tasks within three points (Lift 100, Stack 97, CubeRestack 98, NutAssemblySquare 0) and Wipe at 47 in an

Table 6: SPARK adaptive: object suite per-task success rates (%, 50 trials).

Task	Pos	Task
alphabet soup	18	30
cream cheese	28	40
salad dressing	16	88
bbq sauce	80	18
ketchup	4	50
tomato sauce	70	92
butter	34	8
milk	88	6
chocolate pudding	58	32
orange juice	38	0
Mean	43.4	36.4

Table 7: SPARK adaptive: goal suite per-task success rates (%, 50 trials).

Task	Pos	Task
open middle drawer	0	0
bowl on stove	100	0
wine bottle on cabinet	42	12
bowl in top drawer	0	0
bowl on cabinet	82	0
push plate to stove	0	40
cream cheese in bowl	56	14
turn on stove	10	2
bowl on plate	98	14
wine bottle on rack	12	58
Mean	40.0	14.0

environment rebuilt with robosuite 1.5.2 and Pyroki (5 without Pyroki, where every descent trips the joint-limit termination); the two bimanual rows did not reproduce in that rebuilt environment (TwoArmLift 1, TwoArmHandover 48), which we attribute to the composite-controller configuration and leave as the CaP-Bench build's numbers with this note.

Lift (100%) uses a single-call BT of the form `move_to_keypoint(cube, offset_z=0.05)` → `grasp` → `move_relative(dz=0.15)`, with the cube clearing the table by > 4 cm as the success criterion; no failures were observed at 100 trials.

Stack (97%) chains `move_to_keypoint(cube_A)` → `grasp` → `move_to_keypoint(cube_B, offset_z=0.08)` → `release`, with success requiring cube A to rest on cube B with the contact normal upward. Three failures: two from a near-edge release that toppled the stack, one from a SAM3 detection swap between identically-colored cubes.

CubeRestack (100%) reuses the Stack BT but adds a precomputed pre-condition that the source cube is currently on a third cube, so the success criterion is the additional post-condition that cube A ends on cube B regardless of starting configuration; no failures were observed.

Wipe (60%): Gemini selects the `constrained_scrub` skill, which oscillates a normal force across the SAM3-detected dirt region. The dominant failure mode is the operational-space controller tripping robosuite-Wipe's joint-limit early-termination during the descent: the pose-delta descent passes through a near-limit configuration, so the episode terminates before the pad reaches the table and no markers are wiped.

NutAssemblySquare (0%) is the one task the base grammar cannot complete, and it isolates the limit. Perception is solved: Gemini's adaptive prompts let SAM3 detect the small square nut (e.g. "brass block with hole") and select the square peg by shape at 100% peg-selection accuracy (the

Table 8: MolmoAct2-LIBERO per-task success rates (%, 50 trials). No published LIBERO-PRO results exist; these are from our evaluation using the public MolmoAct2 checkpoint.

	Spatial		Object		Goal	
Task	Pos	Task	Pos	Task	Pos	Task
T0	0	2	100	0	0	2
T1	66	0	66	0	72	0
T2	28	0	20	0	0	0
T3	2	0	12	0	60	0
T4	0	0	58	0	2	0
T5	20	0	68	0	0	18
T6	0	0	0	0	8	0
T7	26	0	0	0	100	100
T8	88	2	98	0	48	0
T9	0	0	50	0	0	0
Mean	23.0	0.4	47.2	0.0	29.0	12.0

Table 9: CaP-Bench results (%). CaP-Agent0 numbers from Fu et al. (2026) (its full 3-model ensemble); RATS from Zhang et al. (2026a) (its "CaP-Agent0 + RATs skills" transfer column, 50 trials/task). SPARK and CaP-Agent0 use 100 trials/task. NutAssemblySquare (CaP-Agent0's Peg Insert) is 0% for all three; SPARK solves the perception, but threading needs pose control under contact that the platform's OSC layer cannot hold (see text).

Task	CaP-Agent0 (3M)	RATS	SPARK
Lift	97	84	**100**
Stack	**98**	60	97
CubeRestack	89	46	**100**
Wipe	**100**	**100**	60
NutAssemblySquare	0	0	0
TwoArmLift	**74**	34	63
TwoArmHandover	20	20	**24**

"cylinder" prompt anchors the round distractor, and the square peg is the post farthest from it), and a general perception-driven grasp primitive orients the gripper to the object's principal axis without affecting other tasks. The gap is the motion layer: the OSC controller cannot hold the aligned wrist orientation through the saturated table-height descent, so the nut is grasped and carried in 26% of trials and seated in none; the same descent instability costs the remaining trials the grasp. Threading needs deterministic full-pose (6-DOF or joint-position) control, which cleared roughly 90% in a probe outside the benchmark's controller but lies outside the grammar's OSC motion layer. The gap is pose control under contact; perception, primitive design, and kinematics all clear this task, and both SPARK and CaP-Agent0 report 0%.

TwoArmLift (63%, 100 trials) is composed by Gemini as a bimanual lift: a `parallel` node runs simultaneous `pick_with_arm(left, handle_left)` and `pick_with_arm(right, handle_right)` branches joined by a `sync_barrier`, followed by a coordinated `bimanual_lift`. Each arm grasps top-down with the wrist yaw aligned to the handle bar, width-gates the grip, and retries independently while the other arm holds; the `sync_barrier` raises the pot only after both arms confirm, so a missed grasp aborts with the pot still in place. Adaptive perception's centroid-merging strategy collapses the two handle detections into one centroid. We suppress merging here and use per-arm prompts for the left and right handles. The dominant failure mode is a single-arm grasp miss from SAM3 handle-centroid noise on the thin handle bar: after the per-arm retries the barrier correctly refuses to lift on one arm, and the pot stays put.

TwoArmHandover (24%) is composed by Gemini as a typed `handoff` primitive: arm A grasps, moves to a meeting point, arm B grasps, then arm A releases (the never-unsupported timing is enforced by the skill, releasing only after arm B confirms its grasp). Arm A's grasp uses the same

width-gated retry as the lift, sweeping wrist azimuth and descent depth after a missed first attempt; this raises arm A's grasp success to $\sim$80%. The dominant remaining failure is arm B's cross-table top-down grasp on the now-vertical handle, which is loose and brittle: the hammer is occasionally dropped during arm B's lift or grasped off the handle, so the transfer does not complete.

SPARK trails where mid-execution feedback pays, on Wipe (60% vs. 100) and TwoArmLift (63% vs. 74): success there is observational (whether the spill is gone or the pot has lifted clear of the table is readable only from the image after acting), which CaP-Agent0's per-turn visual differencing tracks and our single-shot plan commits before seeing; the gap is architectural on TwoArmLift, where a turn-level observation gate that re-checks the success condition mid-execution would recover most of it; the Wipe deficit is bounded by the joint-limit early termination described above. The bimanual tasks are hard for every method: RATS, with its play-distilled skill library, reports 34% on TwoArmLift and 20% on TwoArmHandover; SPARK composes both bimanual skills from primitives with no offline skill phase, exceeding RATS on the lift (63% vs. 34%) and posting the best handover of any method (24%, vs. 20 for both CaP-Agent0 and RATS).

D BT Grammar Reference

Figure 6 summarizes the primitive library. A sample of the YAML BT DSL is given below in BNF-like form. A program is a `task` string and a `tree` whose root is a `sequence` (or `parallel`) node of typed primitive invocations. The grammar is parsed and type-checked before any actuator command issues; the slot listing in Table 10 is abridged, and the registry declares the full slot set the checker admits (including `force`, `offset_x`/`offset_y`, and `scrub_radius_m`). Malformed input triggers the recovery layer of §4. Each primitive returns a boolean post-condition. A false return triggers a retract-and-re-render cycle that re-grounds perception without re-querying the LLM. The grammar surface does not change across recovery rounds.

```
<program>      ::= "task:" <text> NL "tree:" NL <node>
<node>         ::= "type:" <ctrl> NL "children:" NL <child>+
<ctrl>         ::= "sequence" | "parallel"
<child>        ::= "- type:" <prim_id> NL "  params:" NL <arg>*
                 | "- " <node>
<prim_id>      ::= "move_to_keypoint" | "move_relative" | "grasp"
                 | "release" | "wait" | "grasp_se3" | "insert"
                 | "push_object" | "open_drawer" | "pour"
                 | "sweep" | "wipe" | "constrained_scrub"
                 | "pull" | "drag" | "stack" | "screw"
                 | "pick_with_arm" | "sync_barrier" | "handoff" | ...
<arg>          ::= "    " <slot> ": " <value> NL
```

Per-primitive slot signatures and defaults are listed in Table 10.

E Skill-Emergence Ablation

The method section claims that five base primitives (`move_to_keypoint`, `move_relative`, `grasp`, `release`, `wait`) cover the position-controlled motions and that the grammar composes them into multi-step skills with no task-specific code. We test this directly with two reproducible hooks: an environment switch that removes a named composed skill from the grammar shown to Gemini, so the behavior tree cannot call it, and an in-context instruction to express the motion as base primitives before emitting the tree. We then run CaP-Bench tasks with the relevant composed skill withheld and report the success of the base-only plan (Table 11). Executor logs confirm the constraint held: base-only runs emit zero nodes of the withheld skill.

For the kinematic pick-and-place tasks the planner composes a fully executable multi-step skill from the base primitives at the same success as the dedicated path. A withheld-`stack` plan becomes `move_to_keypoint` $\rightarrow$ `grasp` $\rightarrow$ `move_relative` $\rightarrow$ `move_to_keypoint` (with a place offset) $\rightarrow$ `release`, with no task-specific code. This is the measured form of the first contribution.

Wiping marks the boundary. The planner reliably composes a structurally valid base-primitive plan, a boustrophedon `move_relative` raster pressed onto the surface, but it scores 0% where the composed `constrained_scrub` reaches 60% (Appendix C). The composed skill carries closed-

Table 10: Typed slots for each SPARK primitive. Required slots are marked with ($\star$). Five base primitives (`move_to_keypoint`, `move_relative`, `grasp`, `release`, `wait`) compose into the manipulation and tool-use skills below; the table lists these five and a representative subset of the more than thirty registered skills. The scalar slot values (offsets, forces, angles, durations such as `wait`'s) are emitted by Gemini (temperature 0 in simulation, 0.3 on hardware): the keypoint label fixes only the spatial anchor, so the planner sets the scalar values directly.

Primitive	Slots	Default
`move_to_keypoint`	`keypoint_label`*, `offset_z`	`offset_z`$= 0.10$
`move_relative`	`dx`, `dy`, `dz`	all $= 0$
`grasp`	`force`	`force`$= 50$
`release`	–	–
`wait`	`duration`	`duration`$= 0.5$
`grasp_se3`	`keypoint_label`*, `strategy`, `target_width`	`strategy`=top_down
`insert`	`keypoint_label`*	–
`push_object`	`keypoint_label`*, `direction`*, `distance`	`distance`$= 0.10$
`open_drawer`	`keypoint_label`*	–
`pour`	`target_label`*, `pour_angle`	`pour_angle`$= 1.57$
`sweep`	`keypoint_label`*, `direction`, `distance`	`distance`$= 0.15$
`wipe`	`keypoint_label`*	–
`constrained_scrub`	`target_label`*, `force_n`, `n_cycles`, `pattern`	`force_n`$= 8$
`pull`	`keypoint_label`*, `distance`	`distance`$= 0.10$
`drag`	`keypoint_label`*, `target_label`*	–
`stack`	`keypoint_label`*, `target_label`*	–
`screw`	`keypoint_label`*, `angle`	`angle`$= 90$

Table 11: Skill-emergence ablation. With the composed skill removed from the grammar, the planner must build the task from the five base primitives. Pick-and-place reconstructs at full success; wiping composes a structurally valid plan that fails the task's force-and-coverage success check.

Task	Composed skill withheld	Base-only success (%)	Trials
Lift	*(none; already base)*	100	10
Stack	`stack`	100	10
Wipe	`constrained_scrub`	0	20

loop control that a static raster cannot recover from language alone: force-regulated descent to contact, a raster sized to the perceived dirt region, and observation-gated re-wiping. Base primitives therefore compose multi-step kinematic skills with no task-specific code, while contact skills that depend on force and coverage control still rely on a calibrated composed primitive, the role of the more than thirty typed skills.

F Compute Budget

We give a per-trial cost breakdown for each evaluated system. Dollar costs use the public Gemini and frontier-model API list prices current when the runs were made; the per-model rates appear inline below. All SPARK costs below come from per-call usage metadata logged by the planner. The logged population is 7,946 calls over 2,872 trials, a 2.77-per-trial blend across the simulated programs that log per-call usage metadata; it is not the union of every reported sweep, and the per-program means below are the reliable figures. In simulation with adaptive self-consistency, SPARK issues a multimodal variant-generation call and a multimodal BT-generation call per nominal plan, and the adaptive program averages 2.09 calls per trial once recovery replans and label-validator retries are counted. The BT-generation call sends a mean of 19,874 input tokens: the typed grammar, per-primitive parameter guidance, spatial-reasoning instructions, detection context, the recovery-grammar section, and few-shot library examples. Billed output is dominated by reasoning tokens, which average 3,323 per BT-generation call against a 413-token tree, charged together at the output rate. The reasoning-token distribution is right-skewed (median 1,665, mean 3,580 over all 7,946 logged calls), so a median-based estimate understates the token bill by about a factor of

two. The matched-budget arms' totals: the plan-once arm spent \$78.89 over its 600 trials for 307 successes, and the sampling arm \$100.75 over 600 for 357. At Gemini 3.1 Pro pricing (\$2.00/1M input, \$12.00/1M output inclusive of reasoning tokens), the mean call costs \$0.085, so

$$C_{SPARK}^{\text{sim}} \approx \$0.18 \text{ per trial.}$$

An earlier version of this appendix reported \$0.048 per trial from an assumed 7,400-token prompt and 1,700 reasoning tokens. Both assumptions were made before the typed grammar and recovery sections grew the prompt, and the measured figure supersedes them. On physical hardware, Gemini 3.5 Flash (\$1.50/1M input, \$9.00/1M output, \$0.15/1M cached) makes a single BT-generation call, and at the same token profile with the measured 62% implicit-cache coverage this is $\sim$\$0.05 per trial.

CaP-Agent0 in the 3M ensemble setting issues 9 candidate queries per turn (3$\times$ each of GPT-5.2, Claude Opus 4.5, and Gemini-3-Pro) across multiple turns, plus a visual-differencing module adding $\sim$10 further calls (Fu et al., 2026). Fu et al. (2026) report this call structure but not a per-trial cost. Applying public 2026 pricing for those three premium reasoning models to the documented call counts puts the per-trial LLM cost on the order of \$1, roughly 6$\times$ SPARK's measured \$0.18.

G Failure-Mode Analysis

Spatial-Pos. The dominant mode is wrist-camera self-occlusion: the object sits within the gripper-closed occlusion zone, so adaptive's wrist-mask check rejects a valid detection. Pyroki IK also fails on extreme reach poses when the orientation constraint over-restricts the null-space. Recovery re-renders the same prompt set, so a deterministic misdetection persists across retries.

Spatial-Task. Shares the Spatial-Pos modes. Additionally, the spatial rewrite points at a second identical-looking instance, so one prompt matches both (e.g., "black bowl" matches the target and the distractor bowl).

Object-Pos. The 30 px centroid merge rule merges two physically distinct objects that look similar. Pre-grasp clearance is insufficient for tall objects, causing gripper collision on descent. The place pose underestimates the destination container's rim height.

Object-Task. Adaptive's three variants for a renamed object split one concept across multiple SAM3 detections. The variant-generation call occasionally returns phrases that no longer match the original semantics.

Goal-Pos. The destination keypoint z depth is biased by transparent or reflective surfaces (glass plate, polished stove). The BT sequence is correct but `insert` times out before alignment converges.

Goal-Task. Adaptive variant generation produces noisier prompts than the fair baseline for rewritten tasks. Recovery retries do not change the SAM3 prompt set, so deterministic detection failures persist.

Adversarial robustness is out of scope for this paper; three recent robustness benchmarks span 3–28 perturbation types across visual, linguistic, environmental, and action modalities (Xie et al., 2026; Liu et al., 2025; Guo et al., 2026) and are the natural next evaluation of the failure modes above.

H Physical Experiments: Per-Trial Logs

Wilson 95% intervals for the eleven cells at n=20: utensils-in-bowl 55 [34, 74]; utensils-in-tray 90 [70, 97] (UR10e) and 80 [58, 92] (FR3); plushie 100 [84, 100]; stack 65 [43, 82]; sponge-wash 100 [84, 100]; mug pour 65 [43, 82]; sweep 55 [34, 74]; t-shirt fold 50 [30, 70] (FR3) and 30 [15, 52] (bimanual); silverware 60 [39, 78].

Failure modes across these trials are analysed in Appendix §G; two complete behavior trees from these platforms are in Appendix §I. Table 12 lists the eleven cells.physical table restored here

Table 12: Physical success rates (%, 20 trials per task, objects randomly placed and rotated per trial); Wilson 95% intervals for each cell are given in the text above.

Embodiment	Task	Success (%)
UR10e	Utensils in bowl	55
UR10e	Utensils in tray	90
UR10e	Plushie in bowl	**100**
UR10e	Stack blocks	65
FR3	Utensils in tray	80
FR3	Sponge-wash plate	**100**
FR3	Mug pour	65
FR3	Sweep to dustpan	55
FR3	T-shirt fold	50
Bimanual	T-shirt fold	30
Bimanual	Silverware sort	60
Mean (11 cells)		68

Nine tasks span three embodiments (eleven task-embodiment cells total). Objects and placements are randomized per trial, and some tasks swap object categories between runs (different plushie characters, different utensil sets). All runs use the same BT grammar, SAM3 perception, and Gemini planner. Only the IK solver, gripper interface, and workspace bounds differ.

UR10e + Robotiq 2F-85

Utensils in bowl (55%, 11/20). Once a grasp fails, the robot often re-targets the same dropped utensil and fails again. SAM3 classifies fork handles as knives at 0.94 confidence (birdview), causing incorrect tray assignment. The scene is manually reset between trials. The executor can also invert this placement, grasping each utensil back out of the bowl and releasing it on the table, which is the ENPIRE-style automatic reset (Xiao et al., 2026) that would let collection run unattended; the twenty scored trials used manual resets. Occasionally the system self-corrects a previous mistake.

Utensils in tray (90%, 18/20). Trials counted only when the robot yaw-aligned the utensil with the tray direction and correctly separated utensils into distinct trays.

Plushie in bowl (100%, 20/20). Saturated across varied plushie characters (Kirby, Waddle Dee). The top-down grasp primitive is robust to the soft deformable geometry.

Stack blocks (65%, 13/20). Wide golden blocks are harder to pick: the gripper aperture barely accommodates them, and wrist-camera refinement introduces placement offsets during stacking (Figure 7e).

Franka FR3

Utensils in tray (80%, 16/20). The dominant failure is SAM3 labeling a spoon handle as a knife, causing placement into the wrong tray. Cross-embodiment comparison: UR10e scores 90% on the same task, suggesting the wider Robotiq jaw accommodates utensil geometry better.

Sponge-wash plate (100%, 20/20). Saturated. The composed skill (grasp_se3 $\rightarrow$ tool_grip_pose $\rightarrow$ constrained_scrub) executes reliably on the FR3 workspace.

Mug pour (65%, 13/20). Two failure modes: (1) inaccurate handle localization from depth noise at the handle's narrow profile, causing the gripper to miss; (2) Gemini occasionally selects `tilt_wrist` (roll) in place of `pour` (pitch), spilling liquid sideways.

Sweep to dustpan (55%, 11/20). Failures stem from both brush grasp location (too high on the bristles) and maintaining a stable sweep direction toward the dustpan. A bimanual setup where one arm holds and angles the dustpan would improve repeatability.

T-shirt fold (50%, 10/20). Zero successful runs on black or grey shirts: SAM3 returns inconsistent hem labels on dark fabric; sleeve folding still succeeds. Coordinate filtering could mitigate this;

it collapses when objects are oriented. Light-colored shirts fold reliably when sleeve detection succeeds.

BIMANUAL FRANKA (PANDA + FR3)

T-shirt fold (30%, $6/20$). Sleeve folding succeeds at 75% ($15/20$), but the full three-fold sequence (two sleeves plus hem) fails on workspace limits when reaching for the hem. Incorrect primitive selection also contributes: the planner occasionally chains incompatible arm assignments.

Silverware sort (60%, $12/20$). Regression from poor multi-object localization: robots do not always pursue objects in their respective workspace halves, causing arm collisions or unreachable targets. Adding an additional external camera would help here.

Grasp strategy on thin objects. The default top-down grasp sets the gripper yaw from the object mask's oriented bounding box (its principal axis). A flat tray yields a clean box, but a thin knife or spoon resolves to a narrow sliver whose principal axis is unstable and flips between frames (Figure 7f), so early utensil grasps approached across the blade or missed. Silverware tasks therefore use `grasp_se3`, which selects a full SE(3) grasp pose from an SE(3)-equivariant grasp generator (Lim et al., 2024) and reaches it with Pyroki's constrained IK (Kim et al., 2025), in place of a mask-derived yaw.

I EXAMPLE BEHAVIOR TREES

The behavior trees below are emitted by Gemini from the SAM3 detections of real physical scenes; parameter values are exactly as returned by the model (YAML formatting is compacted for space). The single-arm scrub is taken from an executed FR3 run; the bimanual example is Gemini's bimanual-mode plan for the physical silverware-sort scene.

SINGLE-ARM: SCRUB A PLATE (FR3)

Given the base primitives plus the `constrained_scrub` skill, Gemini grasps the sponge, runs a force-limited circular scrub over the plate, and returns the sponge. The scrub's force, cycle count, radius, and pattern are emitted by the planner.

```
task: Pick up the sponge, scrub the plate in a
      circular pattern, and return the sponge
tree:
  type: sequence
  children:
  - type: grasp_se3
    params:
      keypoint_label: sponge
      strategy: top_down
      force: 25
      target_width: 0.035
  - type: move_relative
    params:
      dx: 0.0
      dy: 0.0
      dz: 0.1
  - type: constrained_scrub
    params:
      target_label: plate
      force_n: 8.0
      n_cycles: 4
      scrub_radius_m: 0.04
      pattern: circle
  - type: move_relative
```

```
    params:
      dx: 0.0
      dy: 0.0
      dz: 0.1
  - type: move_to_keypoint
    params:
      keypoint_label: sponge
      offset_x: 0.0
      offset_y: 0.0
      offset_z: 0.02
  - type: release
```

BIMANUAL: SORT SILVERWARE (PANDA + FR3)

The `parallel` node runs two arm-tagged subtrees that join at an implicit barrier: the left arm picks the knife while the right arm picks the fork, then both place into the tray. Every leaf carries an `arm` tag.

```
task: place the fork and the knife into the
      tray, using both arms
tree:
  type: sequence
  children:
  - type: parallel
    children:
    - type: sequence
      children:
      - type: pick_with_arm
        params: {arm: left, keypoint_label: knife,
                 target_width: 0.01, force: 40,
                 prefer_angled: false}
      - type: move_relative
        params: {dx: 0.0, dy: 0.0, dz: 0.2}
    - type: sequence
      children:
      - type: pick_with_arm
        params: {arm: right, keypoint_label: fork,
                 target_width: 0.01, force: 40,
                 prefer_angled: false}
      - type: move_relative
        params: {dx: 0.0, dy: 0.0, dz: 0.2}
  - type: parallel
    children:
    - type: sequence
      children:
      - type: move_to_keypoint_arm
        params: {arm: left, keypoint_label: tray,
                 offset_x: 0.0, offset_y: 0.05,
                 offset_z: 0.08}
      - type: place_with_arm
        params: {arm: left, tilt_angle: 0.0}
    - type: sequence
      children:
      - type: move_to_keypoint_arm
        params: {arm: right, keypoint_label: tray,
                 offset_x: 0.0, offset_y: -0.05,
                 offset_z: 0.08}
      - type: place_with_arm
        params: {arm: right, tilt_angle: 0.0}
```

J RUN PROVENANCE AND PER-TASK RESULTS FOR TABLE 1

Three builds of the runner appear in this appendix, and the released code tags each: the headline build (Table 1 and every re-measured row), the earlier ablation build (perception-sourcing ablation and plan-sampling arms), and the CaP-Bench build (Appendix §C) The headline row of Table 1 comes from one sweep of the headline build: runner `run_spark_libero_pro_fair` with per-task checkpointing, Gemini 3.7 Flash (`gemini-3.7-flash`) for planning and prompt proposal, frozen cross-run library (`SPARK_BT_FROZEN=1`), no BDDL hints, in-process cache off, ten tasks per cell at fifty trials, initial state i for trial i. The ablation-build rows come from the archived run logs documented in the repository's provenance audit. Three goal tasks (both drawer tasks and `turn_on_the_stove`) use model-supplied part poses; excluding them, the assisted goal cells are 59.4 and 58.6 and the assisted macro is 72.5; the unassisted goal cells are 56.9 and 57.1 and the unassisted macro is 71.7.unassisted provenance paragraph + per-task table follow Mid-episode success-predicate access applies to both builds and is disclosed in §5.1. The unassisted headline row ran the same runner and build, with the gate flag `SPARK_NO_ORACLE=1` as the only code change, which gates every mid-episode read of the success predicate (goal short-circuit, push and drawer stop-polls, retry gating, settle guard) while scoring stays the single terminal read; fifty trials per task, same model, frozen library, no BDDL hints. Its per-task counts are in Table **??**; the assisted per-task tables that follow are the earlier, assisted sweep.

Table 13: Per-task successes of 50 trials, object suite, the re-measured Adaptive row (commit-pinned, frozen library, no BDDL hints, Gemini 3.7 Flash). Cell means: position 84.8, task 86.2.

Task	Pos	Task
pick_up_the_alphabet_soup_and_place_it_in_t	39	42
pick_up_the_cream_cheese_and_place_it_in_th	35	41
pick_up_the_salad_dressing_and_place_it_in_	49	48
pick_up_the_bbq_sauce_and_place_it_in_the_	49	48
pick_up_the_ketchup_and_place_it_in_the_bas	43	42
pick_up_the_tomato_sauce_and_place_it_in_th	50	50
pick_up_the_butter_and_place_it_in_the_bask	27	27
pick_up_the_milk_and_place_it_in_the_basket	48	48
pick_up_the_chocolate_pudding_and_place_it_i	34	36
pick_up_the_orange_juice_and_place_it_in_th	50	49

Table 14: Per-task successes of 50 trials, spatial suite, the re-measured Adaptive row (commit-pinned, frozen library, no BDDL hints, Gemini 3.7 Flash). Cell means: position 74.8, task 71.4.

Task	Pos	Task
pick_up_the_black_bowl_between_the_plate_and	32	32
pick_up_the_black_bowl_next_to_the_ramekin_	35	34
pick_up_the_black_bowl_from_table_center_and	50	47
pick_up_the_black_bowl_on_the_cookie_box_an	48	46
pick_up_the_black_bowl_in_the_top_drawer_of	0	0
pick_up_the_black_bowl_on_the_ramekin_and_p	46	44
pick_up_the_black_bowl_next_to_the_cookie_b	44	36
pick_up_the_black_bowl_on_the_stove_and_pla	44	44
pick_up_the_black_bowl_next_to_the_plate_an	27	25
pick_up_the_black_bowl_on_the_wooden_cabinet	48	49

K DEPLOYABLE SELECTOR ROWS FOR THE MATCHED-BUDGET PROGRAM

Ten-state subset provenance for §5.3: the paired arms ran on the April build at ten initial states per task, and the ten-state subset runs high on every suite (spatial task 82.0 against 72.4 at fifty trials; goal task 25.0 against 14.0). The two deployable selector arms below ran at later commits than the paired arms of §5.3 and are reported here with that caveat. On the spatial suite: kinematic-gate selection 45.0 position / 80.0 task; trace-model selection 55.0 / 72.0.

The kinematic-gate arm replaces the oracle with a deployable feasibility gate on the spatial suite: after three defect classes in the gate were fixed, it passed every candidate, chose the first, and scored 62.5 on the suite (the mean of its 45.0 and 80.0 cells), matching its paired baseline's 62.5 at that commit. A geometric veto rule contributes a safety floor without a selection signal, and each fix cost

Table 15: Per-task successes of 50 trials, goal suite, the re-measured Adaptive row (commit-pinned, frozen library, no BDDL hints, Gemini 3.7 Flash). Cell means: position 62.6, task 61.2.

Task	Pos	Task
open_the_middle_drawer_of_the_cabinet	49	48
put_the_bowl_on_the_stove	50	50
put_the_wine_bottle_on_top_of_the_cabinet	22	21
open_the_top_drawer_and_put_the_bowl_inside	15	12
put_the_bowl_on_top_of_the_cabinet	49	49
push_the_plate_to_the_front_of_the_stove	2	2
put_the_cream_cheese_in_the_bowl	27	26
turn_on_the_stove	41	41
put_the_bowl_on_the_plate	50	50
put_the_wine_bottle_on_the_rack	8	7

a measured $\sim\$40$ iteration that the oracle row never pays. The trace-model arm closes most of the gap with no oracle: we mine every recorded trial into (plan, grounding, outcome) rows, each label supplied by the predicate verifier at no annotation cost, and fit a logistic model that ranks candidates before execution. Offline it picks the surviving plan of a held-out cross-run pair 72.5% of the time (n=142; random 50%, the short-plan heuristic 43.0%); live it overrides planner order on 43 of 200 trials and lifts the position cell from 43.0 to 55.0, recovering most of the oracle's 60.0 at $\sim$10 ms of CPU per decision, while the task cell falls from 82.0 to 72.0 exactly as the offline pair accuracy predicted (0.575 task, 0.784 position). The mechanism is a position-grounding ranker, and we report it as such.

Library-ablation mechanisms. The library-ablation pair (65.8 live, 64.8 frozen) ran at a September commit on the three suites at ten tasks by ten states under both perturbations, paired on task index; its level sits above the April-build matched-budget arms because the executor had received the drawer, knob, and grasp repairs by then. The two largest per-task movers pull in opposite directions. A stale stove exemplar from an older executor suppresses the correct `turn_knob` plan the planner writes unaided ($7/10$ frozen falls to $0/10$ live), and a stored label correction rescues a detection mismatch fresh planning fumbles ($0/10$ frozen rises to $7/10$ live). Their opposite signs are why the macro moves one point while individual tasks move seven of ten trials. SHAPER (Wang et al., 2026) reports the same direction as the matched-budget null independently on VLABench, where two test-time aggregation schemes score 21.4 and 22.0 against 23.3 for direct execution.

L Perception-Sourcing Ablation (April Build)

Per-task vectors behind these suite means are in Appendix §B. Planner and controller fixed; rows mutually comparable; absolute level superseded by the headline row. *Fair*: task language only, matching CaP-Agent0. *+BDDL*: adds canonical object names from the tasks' Behavior Domain Definition Language (BDDL) files. *Adaptive*: alternative prompts with SAM3 self-selection.

	Object		Goal		Spatial		
Config	Pos	Task	Pos	Task	Pos	Task	Mean
Fair	36.4	23.4	36.4	22.4	30.0	43.0	31.9
+BDDL	35.0	24.0	36.2	19.4	29.2	43.4	31.2
Adaptive	43.4	36.4	40.0	14.0	56.0	72.4	43.7

M LIBERO-Dyn: Displacement Inside the Episode

LIBERO-PRO and LIBERO-Plus perturb the initial state and then leave the episode undisturbed, so a policy that memorized a trajectory still acts on a static scene. We define LIBERO-Dyn, a protocol that moves an object while the robot acts; THE COLOSSEUM (Pumacay et al., 2024) is the precedent for the form, a perturbation protocol layered onto an existing benchmark with that benchmark's tasks and assets kept. An injector slides the target object across the table at one of three phases (after planning and before the first motion, at half the initial approach distance, immediately

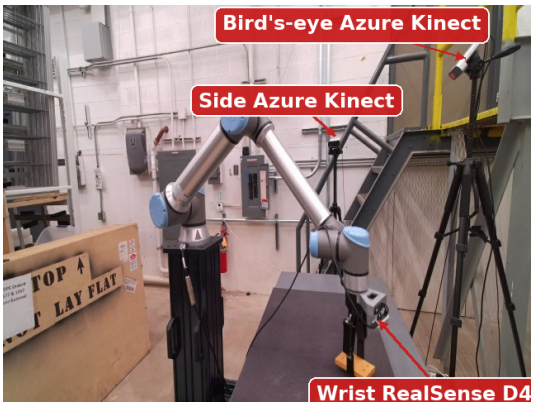
Table 16: LIBERO-Dyn successes of 45 trials per cell (three object-suite tasks × three injection phases × five initial states), identical displacement schedules within each block. The two SPARK arms execute the same scripted plans, so the planner is out of the loop and they differ only in verification and adaptation. The continuous block slides the object over 40 steps at the commit that carries the binder fix described below; the instantaneous block is the original one-step injector, at the commit of the first sweep. The VLA rows are LIBERO-finetuned checkpoints under the instantaneous injector; their 0 cm controls (OpenVLA 86.7% vs. its published ∼88.4% object-suite average; $\pi_{0.5}$ 93.3%, matching prior local baselines) rule out a strawman. Recov. counts logged recovery events over the perturbed cells.

Arm	0 cm	2 cm	5 cm	10 cm	Recov.
Continuous slide (40 steps)					
SPARK, verify and adapt	39/45	37/45	35/45	30/45	185
SPARK, no verification	33/45	26/45	7/45	1/45	0
Instantaneous displacement (one state write)					
SPARK, verify and adapt	36/45	19/45	23/45	12/45	135
SPARK, no verification	27/45	21/45	1/45	0/45	0
OpenVLA (finetuned)	39/45	25/45	4/45	0/45	0
$\pi_{0.5}$ (finetuned)	42/45	38/45	17/45	5/45	0

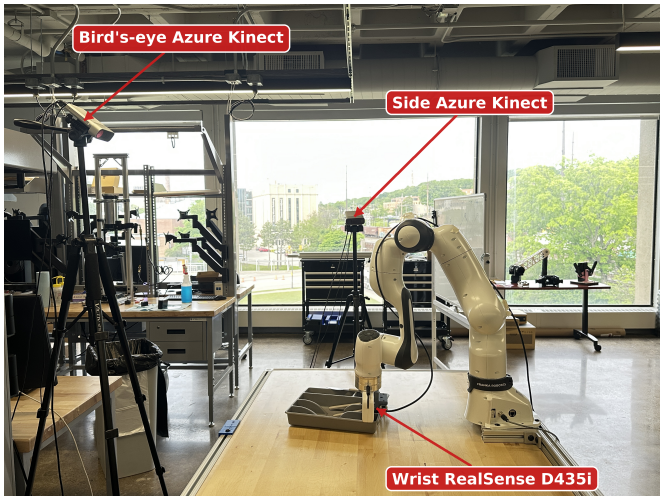
before the gripper closes), spreading the displacement over 40 consecutive simulator steps so the object is visibly in motion while the robot acts; the magnitude sweep is 0, 2, 5, and 10 cm, with a fixed table of eight compass headings selected by a seed derived from the task identifier and trial index, so every method meets an identical schedule of interventions. The original injector applied the whole displacement in one state write; we keep that instantaneous variant as a labeled block of Table 16 because the VLA rows were measured under it, and a slide is the harder and more physical case (an optional multi-object mode moves distractors on staggered schedules with the simulator's contact check rejecting any step that would interpenetrate; it is not used in the table). We report success, recoveries, detection latency (injection to first flag), and adaptation latency (flag to resumed nominal execution). The 0 cm column is a mandatory control, because an arm can score well under displacement and still be broken on the undisturbed scene: our own adaptive arm scored 9/45 on the controls while its perturbed cells looked healthy, its re-detections adopting a worse object centroid than the plan-time binding, and it scores 36/45 after the binding fix (Table 16).

Under a slide the verify-and-adapt arm gives up 9 points from 2 to 10 cm (37/45, 35/45, 30/45) while the no-verification arm collapses from 26/45 to 7/45 to 1/45: the gap between the two arms widens with the displacement, which is the signature of a check that pays more the further the world moves. A slide is also easier to follow than a jump: the camera scene diff sees the object leave its bound position and the in-flight retarget re-aims within the same control step, so 44 of the 102 perturbed successes complete with no detection flag at all, the motion absorbed by retargeting before any threshold trips. The cost lands at the grasp: an object still sliding when the gripper arrives is occluded by the gripper itself, the pre-close diff cannot see the residual offset, and the grasp-telemetry vote that catches the empty close fires later than under a jump (median 4.6 s against 2.4 s), which is the recovery path behind most of the 185 logged events. The 0 cm control moved from 36/45 to 29/45 between the two sweeps, and the column earned its place a second time: two repeated controls at the current commit gave 29 and 30, so the shift was not variance, and a bisection over every commit since the first sweep found no bad commit, because the first sweep's own code scores the same today. The cause was a ghost: the detector now emits a phantom "salad dressing bottle" at confidence 0.030, a hair above the 0.03 detection threshold, and the scripted-plan binder's confidence-free fallback pass bound it because the real bottle's phrase shared no word with the task hint, so the plan grasped empty table and every perception rebind re-adopted the same ghost (0/15 and 1/15 on that task; the other two tasks were unaffected). A 0.1 confidence floor on that fallback pass restores the task to 10/15, and the continuous block of Table 16 is measured at that commit, where the control reads 39/45. The instantaneous block tells the same story with a sharper cliff: the arm without verification holds at 2 cm and collapses at 5 and 10 cm, and the verify-and-adapt arm keeps 23/45 at 5 cm and 12/45 at 10 cm; at 2 cm it scores 19/45, below both VLA rows and its own no-verification arm, because a 2 cm jump usually stays inside gripper tolerance, where a triggered

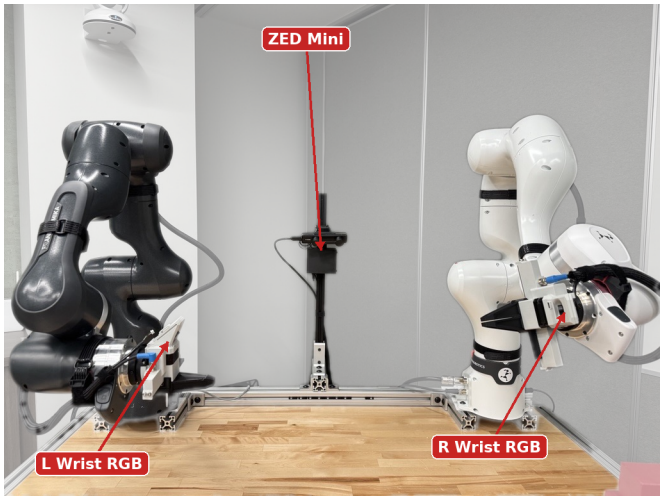
re-detection costs more than it returns. The VLA rows sharpen both sides of that comparison. $\pi_{0.5}$ wins the 2 cm cell outright ($38/45$, the best of any arm): its chunked closed loop absorbs small displacement. The claim is therefore that typed verification and VLA closed loops share a cliff: both collapse past the gripper's tolerance ($17/45$ and $4/45$ at 5 cm, $5/45$ and $0/45$ at 10 cm) with no signal that they failed: detection latency, adaptation latency, and recovery are structurally absent for both policies, not zero, because neither has a failure flag or a replan channel. $\pi_{0.5}$ also fails the post-planning 5–10 cm cells even though that displacement lands before its first observation ($5/15$ and $1/15$): the displaced pose sits outside its training placement distribution, so distribution shift drives the brittleness beyond mid-episode surprise alone. SPARK's latencies split by detector and by adapter, each pair an order of magnitude apart, so we do not report blended medians. Under the instantaneous injector, detection is 0.34 s for the camera scene diff (n=66) against 2.39 s for the grasp-telemetry vote (n=64), and adaptation is 0.000 s for in-flight retargeting, which resumes within the same control step (n=50), against 2.29–2.40 s for a recovery subtree that retracts before re-detecting (n=77). Under the slide the same split holds with the telemetry path slower: scene diff 0.53 s (n=40), telemetry 4.62 s (n=51), retarget 0.000 s (n=41), recovery 3.52–3.85 s (n=45). Two of the 180 adaptive trials report a satisfied post-condition on a failed trial: the object landed within 10 cm of the place target in xy but outside the container, which an xy predicate cannot see.



(a) UR10e + Robotiq 2F-85



(b) FR3 + Franka Hand



(c) Bimanual workcell (Panda left, FR3 right, MSG compliant grippers)

Figure 3: Camera rigs for the three physical platforms, with each camera labeled in-image. The single-arm platforms (a, b) use a bird's-eye and a side Azure Kinect DK plus a wrist RealSense D435i. The bimanual workcell (c) replaces the side camera with a single rear ZED Mini facing the workspace and adds one wrist OV9732 camera per arm. In simulation, LIBERO-PRO fixes the rig to the standard `agentview` third-person camera and the `robot0_eye_in_hand` wrist camera; we do not modify camera placement.

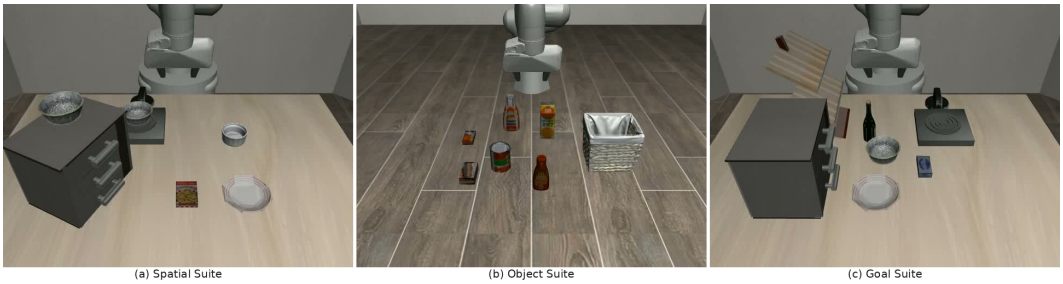


Figure 4: Representative LIBERO-PRO simulation environments. (a) Spatial suite: bowl pick-and-place with position perturbation across kitchen fixtures. (b) Object suite: grocery items into a basket with object substitution. (c) Goal suite: multi-step tasks involving drawers, stove, and cabinet with goal redefinition.

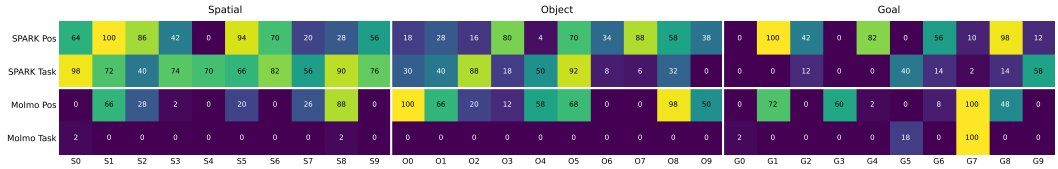


Figure 5: Per-task heatmap of position and task success rates (%) across three LIBERO-PRO suites. SPARK outperforms MolmoAct2 on task-level perturbation across all three suites; MolmoAct2 achieves higher position accuracy on specific object-suite tasks but scores near zero on task perturbation.

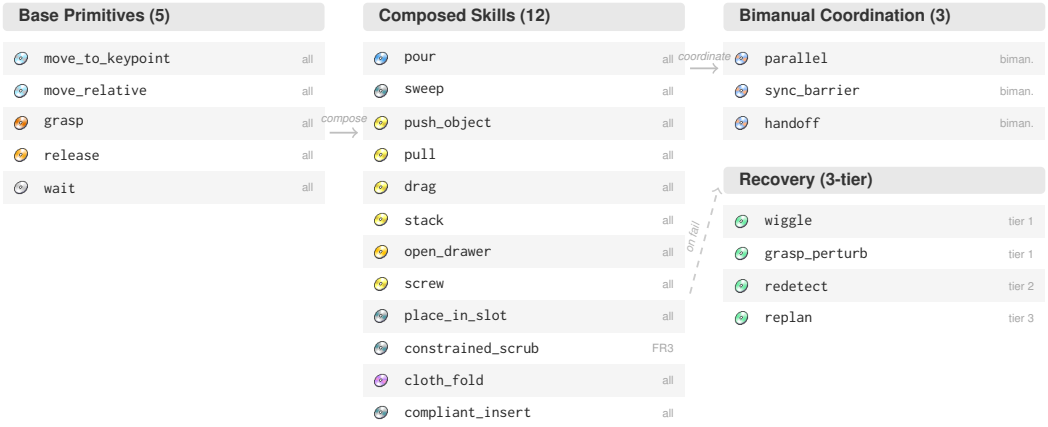


Figure 6: The SPARK primitive library. Five base primitives (left) compose into the manipulation and tool-use skills (center) and bimanual coordination nodes (right). The three-tier recovery hierarchy escalates from in-place perturbation (`wiggle`, `grasp_perturb`) to perception re-grounding (`redetect`) to plan regeneration (`replan`); within an execution attempt only the first two tiers run; plan regeneration is the between-attempt last resort.

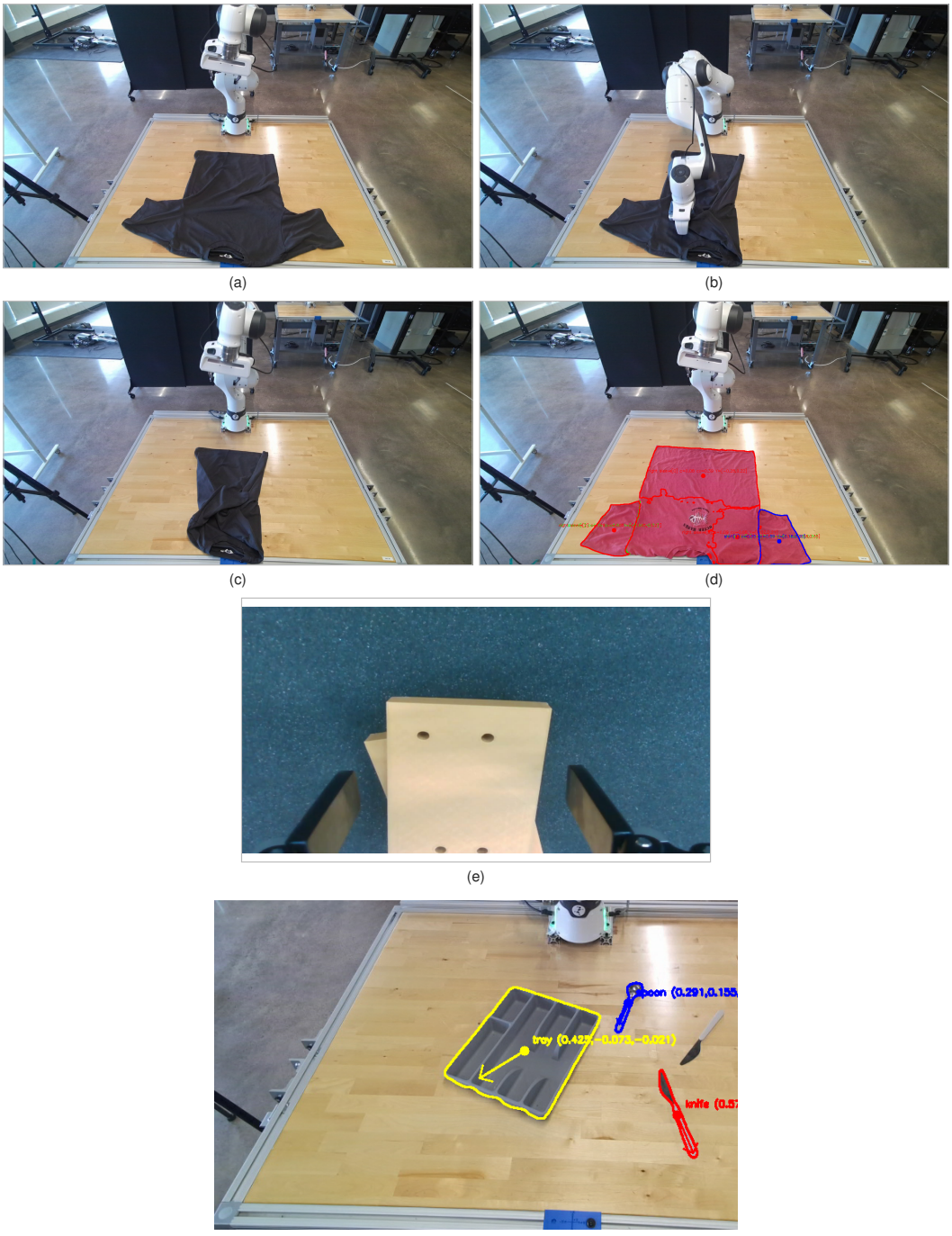


Figure 7: Perception-driven failure modes. (a)–(d) Garment folding: (a) dark shirt laid flat, (b) sleeves folded inward successfully, (c) hem fold fails because SAM3 cannot resolve the hem boundary on dark fabric, (d) Red shirt masks showing sleeve masks that bleed into the body region with ambiguous left/right labeling. (e) Block stacking: wrist-camera refinement introduces a lateral offset on wide gold blocks, causing the gripper to grasp off-center and misalign the stack. (f) Utensil-scene grounding: the tray major axis from the OBB is mis-aligned causing poor slot placement with the utensils, so silverware tasks use `grasp_se3` in place of an OBB-derived gripper yaw.

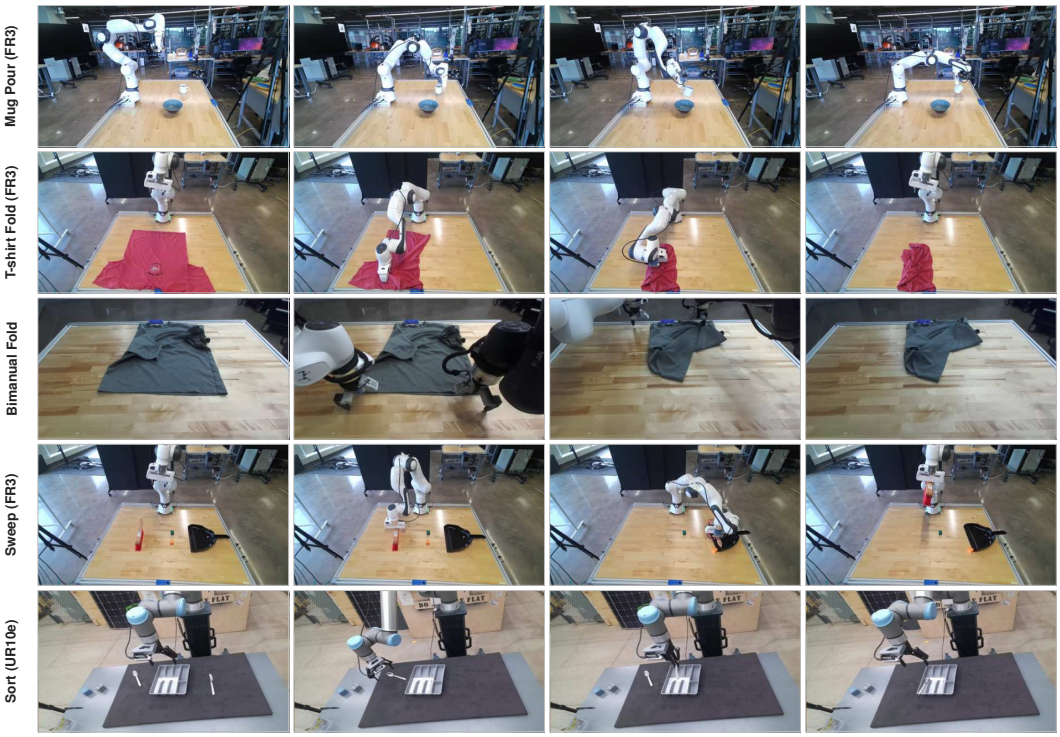


Figure 8: Physical execution sequences. Each row shows four keyframes of a single task: mug pour (FR3), t-shirt fold (FR3), bimanual fold (Panda + FR3), sweep to dustpan (FR3), and silverware sort (Panda + FR3). The same typed BT grammar and SAM3 pipeline run across all three platforms with no task-specific training.