Generalized Planning Interfaces for Agentic Robotics

Anonymous Authors

Abstract: Agentic manipulation requires reasoning over semantic task structure and continuous robot motion. Existing robot interfaces determine which decisions an agent can express and which it must resolve explicitly. High-level skill interfaces restrict task coverage, while detailed low-level motion generation imposes substantial inference cost and is more prone to geometric errors. We introduce generalized planning interfaces (GPIs), which let agents specify what a robot should achieve and how it should act, at any level of abstraction. Agents flexibly combine declarative goals and motion constraints with imperative skill sequences and fully specified trajectories. We instantiate a GPI with SPROUT (Sampling and Pooled Refinements of Underspecified Trajectories). SPROUT interleaves spatiotemporal constraint-graph synthesis with sampling and optimization, preserving agent-defined goals and constraints, while supplying the physical variables and constraints needed for execution. Compatible numerical queries are pooled across heterogeneous graphs for GPU execution. Across eight simulated tasks with three scenes each, SPROUT achieves a 29 percentage point increase in success, compared with less flexible agent-robot interfaces. SPROUT enables agents to solve complex multi-step manipulation tasks, reasoning at the level of abstraction required for the task. Additional information and videos are shown on our website: https://sprout-project-d81650.pages.dev/

Keywords: Agents, Planning, Manipulation

1 Introduction

General-purpose robots have to translate semantic task requirements into physical actions across a wide range of manipulation problems. Language and Vision-Language Agents can reason about these requirements, but solving a task also depends on how they communicate their decisions to the robot. For this, a robot *interface* or API is required. The interfaces between the agent and these robot systems determines which control decisions the agent can express and, consequently, which tasks it can solve. For example approaches like TiPToP [1] translate instructions into symbolic goals for a downstream task-and-motion-planning (TAMP) solver, producing a robot trajectory. Code as Policies (CaP) [2] and CaP-X [3] expose primitives via a python API that agents compose through code. Such interfaces support reasoning within predefined abstractions, but restrict the class of tasks agents can address.

For different tasks, different agent-robot interfaces are better. For example, placing an apple on a plate can be easily specified by a declarative relational goal. TAMP-style approaches are efficient for such tasks, delegating geometric reasoning to a dedicated model-based planning system. However, carrying a full mug requires additional geometric constraints on its grasp and orientation during transport that may be outside the planning domain. As such, at the cost of additional agent reasoning, a CaP-style agent could synthesize a solution. If the agent is instead asked to draw a particular picture, this requires the agent to directly produce end-effector/tool trajectories, imposing a substantial reasoning cost and latency, and this often introduces geometric errors.

Often, for complex and realistic tasks e.g. *Draw a big smiley face on the whiteboard* (see Fig. 4.7), a mixed level of abstraction is best. Efficient agentic robotics therefore requires choosing an interface at the right level of abstraction for the particular task, and being able to *flexibly* combine and move between them. We introduce Generalized Planning Interfaces (GPIs), a class of agent-robot interfaces for flexibly specifying behavior at arbitrary levels of abstraction. Building on hybrid

imperative-declarative planning [4], these interfaces accept objectives, constraints and prescribed actions within the same agent request. At the symbolic/task level, an agent can specify a semantic relational goal to achieve e.g. *clear(whiteboard)*, as well as order subtasks. At the geometric/motion level, it can impose an orientation constraint, or supply waypoints, joint configurations, or even a complete pen trajectory to draw the smiley face. Abstraction level and commands along the declarative-imperative spectrum can therefore vary independently and be flexibly combined, allowing the agent to quickly define behaviors for solving such tasks.

We instantiate a GPI with **SPROUT** (**S**ampling and **P**ooled **R**efinements **o**f **U**nderspecified **T**rajectories). SPROUT represents each request as a partial spatio-temporal constraint graph. The graph relates semantic and geometric variables across abstraction levels to constrain the robot trajectory and define behavior. SPROUT graph growth involves synthesizing completed candidate constraint graphs from partial ones, supplying the remaining unresolved structure. Then, the SPROUT solver determines the remaining variable values to produce a trajectory satisfying the agent's specifications *and* the robot's physical constraints.

The amount the Agent specifies the motion determines how much search remains for the compiler and solver. Additional agent decisions about skills, placements or waypoints constrain graph synthesis, reducing the number of candidate graphs. Whereas, a weak declarative objective leaves more alternatives to explore. GPI's therefore expose a trade-off between computation spent on agentic reasoning and computation spent on graph synthesis and solving. The computationally rational allocation varies across tasks and subtasks: some benefit from precise agent guidance, while others are solved more efficiently by delegating choices to the model-based solver. SPROUT employs GPU-accelerated constraint-graph solving to efficiently resolve the choices left unspecified by the agent.

We evaluate agents using SPROUT on 8 tasks requiring agentic and physical reasoning at multiple abstraction levels, including constrained manipulation, tool use, and tasks beyond pick-and-place with agent-specified motions. The system achieves 67% overall success, compared with 17% for end-effector control and 38% for both skill-plus-end-effector API and task-and-motion-planning (TAMP) interfaces. It completes six of the nine challenging instances involving toast preparation, drawing, and tool-mediated retrieval. Graph completion coordinates dependent manipulation choices, and agents using SPROUT demonstrate faster execution and improved recovery behavior.

2 Related Work

Interfaces for agentic manipulation. Language-conditioned manipulation has been formulated as symbolic goal inference [5, 1], executable policy synthesis [6, 2, 3], and trajectory optimization under generated objectives or constraints [7, 8, 9]. Motion specifications include point-based affordances [10] and end-effector trajectories [11]. Recent harnesses expose semantic skills *and* numerical motion commands [3, 12], with geometric computation delegated through agent-composed calls. GPIs also permit the composition to remain underspecified: goals, skill selections, and motion constraints define a problem whose remaining discrete structure and continuous parameters are resolved by the compiler and solver.

Hybrid task and motion specification. Program sketching [13] and plan-outline synthesis [14] complete partial specifications through constraint solving. CDL/Crow [4] combines declarative goals with commitments to behavior selection, ordering, and variable bindings. Agentic approaches also integrate generated specifications with geometric feasibility [15]. PRoC3S [16] searches continuous parameters of generated programs, while OWL-TAMP [17] accepts partial action sequences and generated continuous constraints. Our formulation extends across task requirements and prescribed motion, including complete trajectories. Following CROW's distinction between external commitments and unresolved decisions, SPROUT preserves agent-supplied requirements while synthesizing the additional graph structure from variables and physical constraints needed for execution.

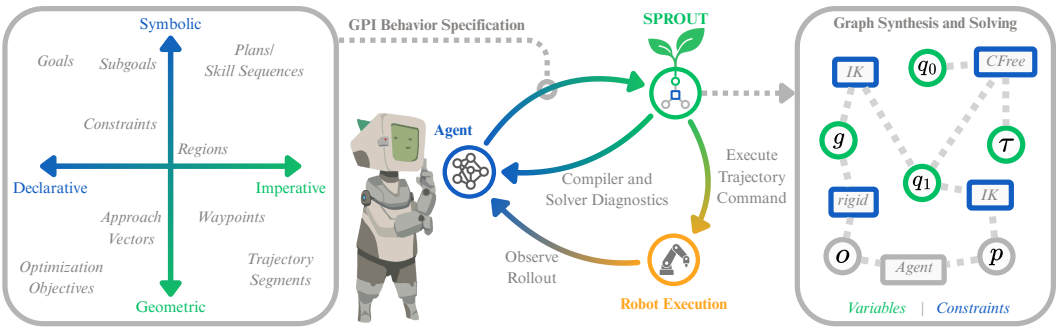


Figure 1: Agent control through a GPI instantiated with SPROUT. The agent combines declarative requirements and prescribed actions across task and motion levels. SPROUT synthesizes and solves constraint graphs to produce trajectories or diagnostic feedback. Execution observations inform subsequent requests.

GPU-accelerated task and motion planning. SPaSM [18] optimizes placements and trajectories for a prescribed action sequence, while STAMP [19] searches action sequences and motion parameters through continuous relaxation. GPU parallelism accelerates inverse kinematics and trajectory optimization in cuRobo [20], and continuous refinement in cuTAMP [21]. Whereas cuTAMP batches assignments for a selected plan skeleton, SPROUT pools numerical queries across candidate graphs with *heterogeneous* constraint structures. Alternative action decompositions can therefore share GPU computation, accelerating the solver.

3 Generalized Planning Interfaces

We consider an agent controlling a robot to complete a natural-language manipulation instruction ℓ. The system has access to a robot model M, a controller, model-based perception, and available skills and generative models. At interaction round k, perception provides a scene estimate $\hat{s}_k$ containing the robot configuration and observed objects' semantic attributes, geometry, and spatial relations. Additional objects may be discovered during execution.

Conditioned on ℓ, observations, and previous feedback, the agent issues a request r_k through a generalized planning interface (GPI). A GPI supports declarative requirements and imperative specifications at task, skill, and motion levels. Declarative requirements constrain outcomes and intermediate states whilst imperative specifications prescribe actions, their order, or motion segments. Both forms may occur within the same request scoped to a task, action, or motion interval. This allows the agent to construct requests which prescribe a particular skill sequence, while leaving its geometric parameters free, or constrain motion while leaving the overall sequence unresolved.

The interface receives $(\hat{s}_k, r_k)$ and must resolve unspecified discrete and continuous decisions to produce an executable trajectory τ_k of robot and object states with associated robot controls. A valid completion satisfies

$$\tau_k \in \mathcal{F}(M, \hat{s}_k) \cap \mathcal{C}(r_k), \tag{1}$$

where $\mathcal{F}(M, \hat{s}_k)$ contains trajectories feasible under the robot model and scene estimate, and $\mathcal{C}(r_k)$ contains trajectories satisfying the request's hard requirements. Requested costs are used to rank and/or optimize admissible trajectories. If no valid completion is found within the available computation budget, the interface returns diagnostic feedback. Execution supplies observations for subsequent requests.

In Sec. 5 we measure task success against ℓ and latency, reporting time spent on agent inference and request completion separately. Experiments test how varying abstraction and degree of specification affects task coverage and the allocation of computation between agent and solver.

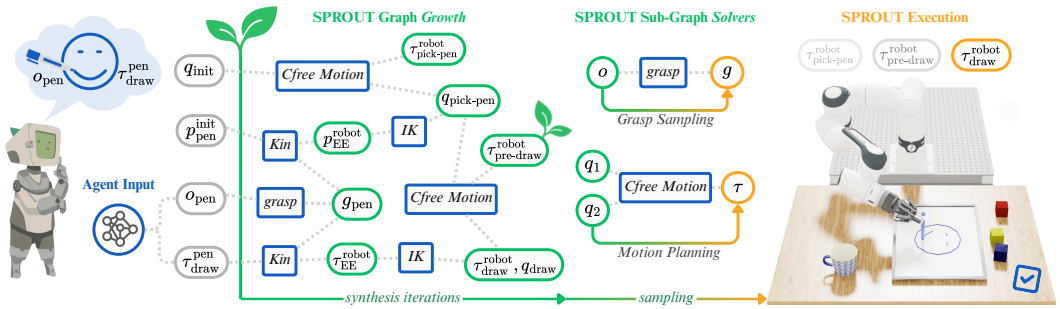


Figure 3: An agent-SPROUT example for the particular task: *Draw a big smiley-face on the whiteboard :)*. The agent reasons, identifying the pen object, and defining a trajectory for that pen to move through in order to the draw the smiley face. This request, along with variables defined by the initial world state s are used to seed SPROUT graph growth. Then, once the graph is complete, the parallelized solvers produce valid, executable trajectories that are run on the robot to complete the agents request.

4 SPROUT: Completing GPI Requests by Growing Constraint Graphs

SPROUT completes GPI requests through interleaved graph synthesis and parameter search. Given an agent request r, a perceived scene $\hat{s}$, and a robot model M, it constructs a partial constraint graph. Its compiler expands unresolved requirements into actions and physical constraints, while sampling and optimization assign values to the remaining variables. Appendix A provides additional algorithmic details.

4.1 Representing agent requests

Our Pythonic API provides operations for declaring goals, prescribing skills, and expressing geometric constraints or motion commands. A snippet of this is shown in Fig. 2. Concrete agent-supplied values become implicit equality constraints. Unspecified quantities remain free variables that the SPROUT solver must find. This API also includes action nesting and ordering and the state or interval over which each constraint applies. For example, placing a mug upright constrains its *final* orientation, whereas carrying it upright constrains its orientation *throughout* transport. A handle-grasp requirement, prescribed action sequence, or waypoint added to the GPI request further constrains the solution trajectory. In the following sections, we refer to Agent-supplied variables and constraints as *extrinsic*. Those introduced by SPROUT's compiler and solver are referred to as *intrinsic*.

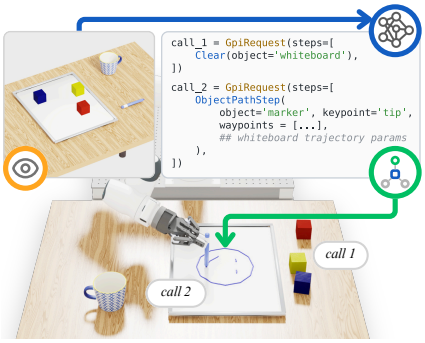


Figure 2: An example of the GPI requests that agents produce for the task: *Draw a big smiley-face on the whiteboard.*

4.2 Synthesizing constraint graphs

In SPROUT, a GPI request is represented as a partially complete constraint graph, as shown in Fig. 3. Edges identify the variables used by each constraint. Typed variables include things like object identities, poses, and joint configurations, with coordinate frames and temporal references attached where applicable. References to the same quantity can therefore share a variable in the graph. However, quantities over different temporal states, such as a the robot's initial and final configurations (q), use distinct variables.

A registry $\mathcal{R}$ defines variable types, skill subgraphs, as well semantic and geometric relations, and sampling procedures (including generative models). For example, SPROUT has access to a grasp

sampling networks like GraspGen [22], and motion planning and optimization through cuRobo [20]. Entries in $\mathcal{R}$ specify how constraints and variables are expanded or evaluated, defining the base graph growth procedure. Expanding a placement goal can introduce grasping, transport, and release. A handle-grasp constraint is expanded into geometric relations between the gripper and handle. Graph expansion adds intermediate structure, bridging from the initial state of the world s while preserving agent-supplied constraints, fixed values, and costs, to valid robot trajectories. *Completing* a graph involves reaching a set of robot trajectory nodes that define a valid executable motion. Model-based synthesis and search is essential here, allowing us to validate the graphs with the internal robot and scene models, ensuring continuous and physically realizable world-state transitions. This also allows us to guide the graph expansion towards minimal completions, efficiently producing sets of valid complete graphs for a given GPI request.

Importantly, both graph synthesis and solving is non-unique. That is, each GPI request admits many candidate graph completions. Then, within a single graph, alternative values for a graph's variables (grasps, placements, etc) produce candidate assignments within that graph.

4.3 Sampling and numerical refinement

A graph is *solved* when its constraints have all been satisfied by the variable choices. Numerical search can begin earlier, even before graph completion, allowing tentative variable values to guide expansion. For example, even a prescribed object trajectory leaves choices of grasp and robot motion unresolved. A fully specified executable trajectory requires validation and conversion to controller commands.

For a completed graph G, let V_G contain its variables and C_G its hard constraints. An assignment z gives each variable v a value in its domain $\mathcal{D}_v$. Over graph alternatives $\mathfrak{G}(r, \hat{s})$ synthesized under M and $\mathcal{R}$, SPROUT searches for

$$\begin{aligned} \min_{G \in \mathfrak{G}(r,\hat{s}),\, z \in \prod_{v \in V_G} \mathcal{D}_v} \quad & J_G(z) \\ \text{subject to} \quad & c(z_{N(c)}) = 1, \qquad c \in C_G, \end{aligned} \tag{2}$$

where $N(c)$ identifies a constraint's variables and J_G evaluates trajectory cost, including requested preferences (for example, the agent may define certain optimization objectives explicitly). The trajectory $\tau = d_G(z)$ is assembled from the assigned motion segments and controls.

The solver maintains multiple variable assignments per graph, similar to the particle-style solve used by cuTAMP [21]. Registered generators in $\mathcal{R}$ propose unknown values conditioned on known/solved ones, such as robot configuration waypoints conditioning a motion planner (as shown in Fig. 3). Interleaved optimization adjusts supported variable groups (within differentiable sub-graphs) to satisfy coupled constraints and reduce cost, as well as to improve the performance relative to any extrinsic costs or preferences. Each completed trajectory is checked against all hard requirements before acceptance, and unsuccessful assignments lead to further numerical search or exploration of another graph.

4.4 Pooling computation across graphs

Different graphs often require the same variable samplers. For example, motion planning or inverse/forward kinematics are common operations across all graphs. SPROUT pools these queries across graphs and assignments for batched GPU execution. Each query records its operation, inputs, and destination graph and assignment. For ready queries $\mathcal{Q}$ with GPU implementations, $\mathcal{B}_o = \{q \in \mathcal{Q} : \text{type}(q) = o\}$, where o identifies an operation and compatible input dimensions. Inputs are gathered from GPU-resident environment representations (spheres, prims and SDF), evaluated together, and returned to their originating assignments. Batching therefore does not require identical graph structure, accelerating search through parallelization. Both graph structure and variable-constraint dependencies determine the order and parallel throughput capacity of each graph. As a result, certain GPI requests admit stronger parallelization than others. This pooled query-based approach is an approach to handle this heterogeneous structure.

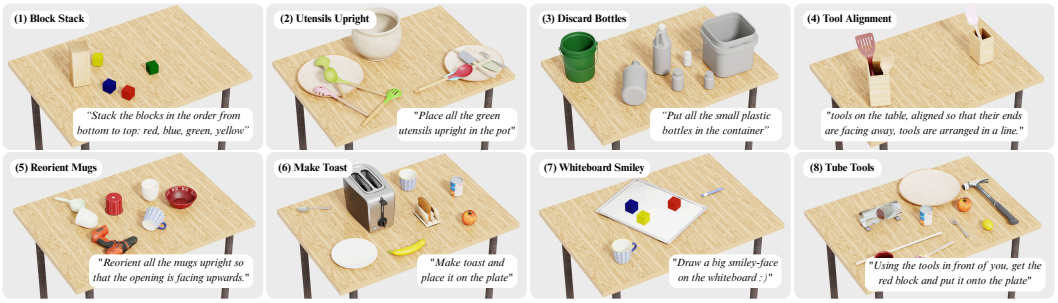


Figure 4: The eight simulation tasks, shown with representative scenes and instructions. Three scenes per task vary object arrangements, geometry, and clutter. Tasks combine relational objectives, orientation requirements, and interactions beyond pick-and-place.

Graph expansion and numerical refinement continue until the computation budget expires or the active search is exhausted. SPROUT returns the lowest-cost validated trajectory found. If no solution is found, solver diagnostics are fed back to the agent, guiding it in-context to produce different GPI requests in the next interaction round $k + 1$.

5 Experiments

5.1 Experimental setup

We evaluate eight tabletop manipulation tasks in Isaac Sim using a FR3+Robotiq 2F-85 gripper. Each task has 3 scenes that vary in layout, geometry, and clutter. Fig. 4 shows representative layouts. The suite includes ordered stacking, utensil placement, bottle disposal, tool alignment, mug reorientation, toast preparation, whiteboard drawing, and tool-mediated retrieval. The final three require interaction beyond pick-and-place, forcing the agent to use its full interface capabilities. All layouts are shown in Appendix B.

Compared systems. We compare against Agentic end-effector control (`EE`), as well as more standard mixed skill api's (`Skill+EE`), such as the Guava [12] or CaP [2, 3] formulations. We also compare against Agent-TAMP calling formulations. `TAMP` requires each request to be expressed as a symbolic goal in its fixed planning domain, motivated by goal-directed systems such as TiPToP [1]. `SPROUT` accepts mixed GPI requests and completes unspecified actions and motion. All systems use claude-sonnet 5 as the agent, and are restricted to 5, 10 and 15 minute timeouts for problem solving. This represents a 10-15 $\times$ increase in solution time compared to human teleoperation speed for these tasks. This timeout is the reason why we use sonnet, rather than more capable agents like Opus 5.5 or Astra, which exceed these time limits due to extensive reasoning. We provide the same scene-state and perception information to all systems under a unifed `get-scene()` operation.

Measurements. We report successful instances out of three for each task, aggregate success over 24 instances, and mean wall time per task. TAMP fails to specify a command on the final three tasks and returns N/A. These outcomes count as failures in the 24-instance denominator and have no numerical runtime entry. The aggregate runtime comparison uses the first five tasks, for which all conditions have reported timings. Runtime is interpreted jointly with success because unsuccessful episodes can terminate without completing the required behavior.

5.2 Task success and runtime

SPROUT completes 16/24 instances (66.7%), compared with 9/24 (37.5%) for both `Skill+EE` and `TAMP`, and 4/24 (16.7%) for `EE` (Fig. 5). This is an improvement of 29.2 percentage points over either of the two stronger comparison approaches. SPROUT obtains at least one success on every

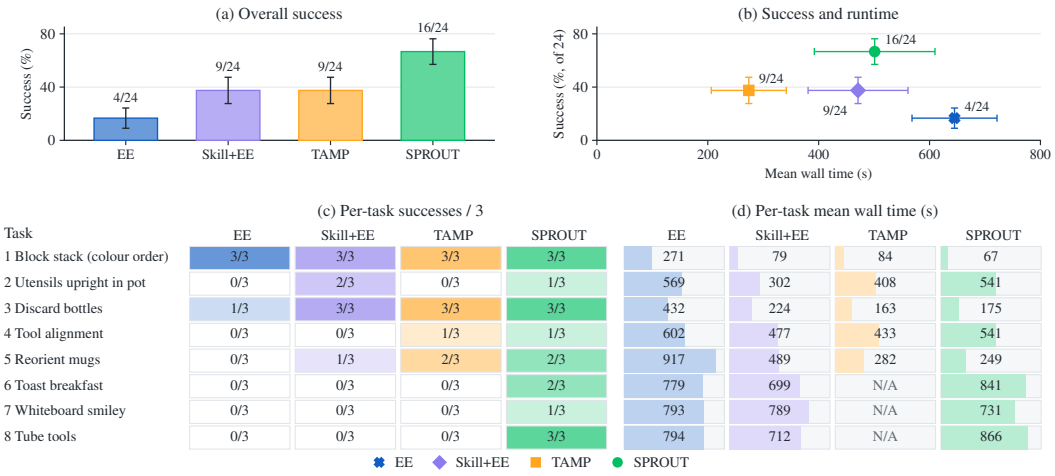


Figure 5: Comparison of four control interfaces. (a) Overall success on 24 task–scene instances. (b) Success and mean wall time on the first five tasks, where every condition has numerical timings. (c) Successful instances out of three per task. (d) Mean wall time in seconds. N/A denotes TAMP's failure to specify a command, which returns NaN and counts as a task failure. Runtime coordinates in (b) are calculated from the rounded task means in (d).

task. Its largest gains occur on toast preparation, whiteboard drawing, and tube-tool retrieval. The other conditions record no successes on these tasks.

Performance varies with tasks, indicating the differences in solve difficulty. All the interfaces complete the block stacking task, with the interfaces able to specify skills or declarative objectives (`SPROUT`, `TAMP`, `Skill+EE`) completing much faster than specifying end-effector controls (`EE`).

Abstraction improves both success and speed relative to direct agentic end-effector control. Across the first five tasks, mean wall time is approximately 558 s for `EE`, 314 s for `Skill+EE`, 274 s for `TAMP`, and 315 s for SPROUT. The corresponding success counts are 4/15, 9/15, 9/15, and 10/15. TAMP is fastest over this shared task set, where its constrained interface delegates geometric decisions to planning. SPROUT's mean is approximately 15% higher than TAMP's, reflecting the added complexity of reasoning about more flexible GPI specifications.

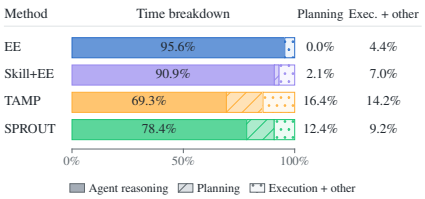


Figure 6: Timing breakdown of each approach, with `TAMP` and `SPROUT` able to more rationally allocate computation than `SKill+EE` and `EE` agent-robot interfaces.

SPROUT also benefits from fast declarative completion when detailed agent control is unnecessary. It has the lowest reported mean for stacking (67 s) and mug reorientation (249 s), while TAMP is faster on bottle disposal (163 s versus 175 s). Across all eight tasks with recorded numerical times, `EE` has a mean of approximately 645 s, compared with 471 s for `Skill+EE` and 501 s for SPROUT. Thus, the agent-heavy EE condition has both the lowest success and the highest mean runtime. SPROUT uses additional computation relative to the more restricted interfaces to complete tool and contact tasks that those conditions fail to solve.

5.3 Computationally Rational Reasoning Allocation

From the results, we see that GPI interfaces such as SPROUT allow agents to flexibly define task goals and constraints, at any level of abstraction. Through this ability, the SPROUT robot-agent interface results in an improved balance between agent reasoning, planning/solving and execution, as shown in Fig. 6. At the same time, this flexibility allows the agent equipped with SPROUT to outperform the other interfaces with equivalent perception and reasoning capacity. An example of

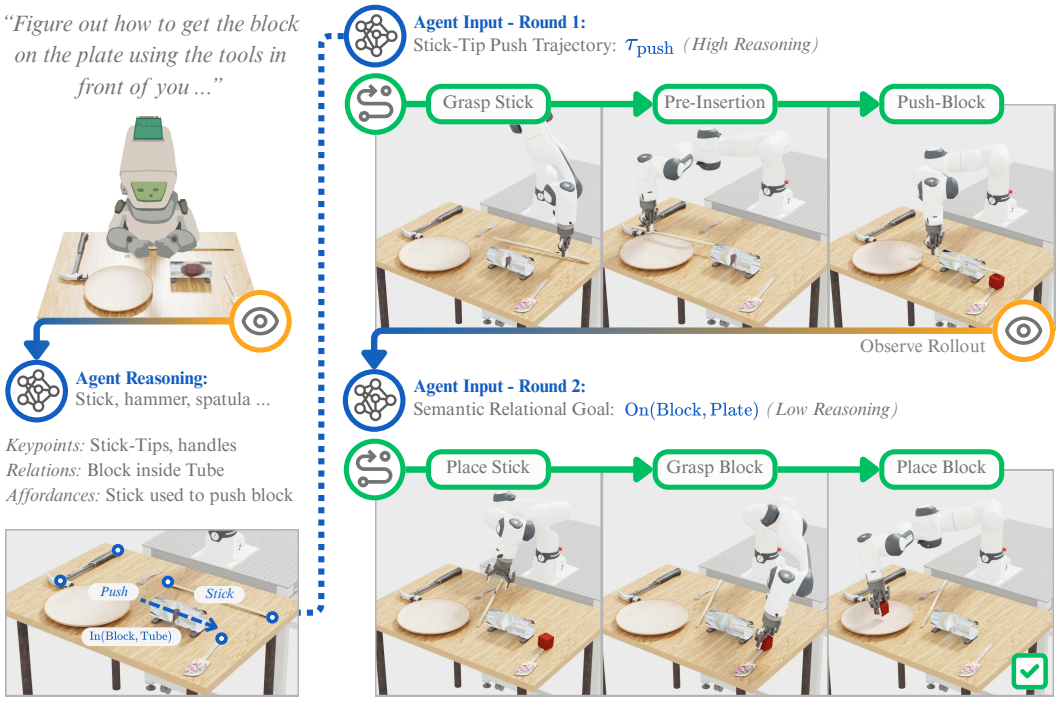


Figure 7: Two-round execution of the tube-tool task. In the first round, the agent specifies a stick-tip push trajectory. SPROUT completes the grasp and approach before executing the tool motion. After observing the extracted block, the agent supplies the relational goal $\mathrm{On}(\mathrm{block}, \mathrm{plate})$, and SPROUT completes tool placement followed by block transfer. The requests differ in abstraction and degree of specification within the same episode.

this is shown in Fig. 7, where we see that for the first call, the agent spends a long time reasoning about the tool trajectory to push the block out of the red tube. Then, it's able to provide a high-level semantic relational GPI request with little additional reasoning. In these cases, SPROUT succeeds where the other interfaces fail because of the solver flexibility and SPROUTs model-based reasoning capacity. For example, being able to produce valid grasps on tools so that they can reach the block in the tube, and delegating this to the SPROUT solver saves on reasoning time.

6 Conclusion

Generalized planning interfaces let agents combine task objectives with prescribed actions and motion within a single request. SPROUT completes these requests through constraint-graph synthesis and pooled GPU-accelerated search. It combines the computational benefits of flexible requests, with improved generality from agent reasoning, balancing the two to achieve stronger performance and faster solutions.

Concurrent work is improving execution reliability and the allocation of computation between agent specification and solver search, as well as deploying the algorithm for tasks in the real-world. **Future work** involves fine-tuning fast, local agents in simulation to solve these embodied reasoning tasks, taking in perceptual input and generating and monitoring GPI requests.

References

[1] W. Shen, N. Kumar, S. Chintalapudi, R. Lindeborg, J. Wang, C. Watson, E. Hu, J. Cao, D. Jayaraman, L. P. Kaelbling, and T. Lozano-Pérez. TiPToP: A modular open-vocabulary robot manipulation system that plans. *arXiv preprint arXiv:2603.09971*, 2026. doi:10.48550/arXiv.2603.09971. URL https://arxiv.org/abs/2603.09971.

[2] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500, 2023. doi:10.1109/ICRA48891.2023.10160591. URL https://arxiv.org/abs/2209.07753.

[3] L. Fu, J. Yu, K. El-Refai, E. Kou, H. Xue, H. Huang, W. Xiao, G. Wang, D. Niu, F.-F. Li, G. Shi, J. Wu, S. Sastry, Y. Zhu, K. Goldberg, and L. Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026. doi:10.48550/arXiv.2603.22435. URL https://arxiv.org/abs/2603.22435.

[4] J. Mao, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Hybrid declarative-imperative representations for hybrid discrete-continuous decision-making. In *Algorithmic Foundations of Robotics XVI, Volume 1*, volume 37 of *Springer Proceedings in Advanced Robotics*, pages 3–22. Springer, 2026. doi:10.1007/978-3-032-09967-9_1. URL https://link.springer.com/chapter/10.1007/978-3-032-09967-9_1.

[5] Y. Xie, C. Yu, T. Zhu, J. Bai, Z. Gong, and H. Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023. doi:10.48550/arXiv.2302.05128. URL https://arxiv.org/abs/2302.05128.

[6] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. ProgPrompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530, 2023. doi:10.1109/ICRA48891.2023.10161317. URL https://progprompt.github.io/.

[7] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia. Language to rewards for robotic skill synthesis. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 374–404. PMLR, 2023. URL https://proceedings.mlr.press/v229/yu23a.html.

[8] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and F.-F. Li. VoxPoser: Composable 3D value maps for robotic manipulation with language models. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 540–562. PMLR, 2023. URL https://proceedings.mlr.press/v229/huang23b.html.

[9] W. Huang, C. Wang, Y. Li, R. Zhang, and F.-F. Li. ReKep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 4573–4602. PMLR, 2025. URL https://proceedings.mlr.press/v270/huang25g.html.

[10] K. Fang, F. Liu, P. Abbeel, and S. Levine. MOKA: Open-world robotic manipulation through mark-based visual prompting. In *Proceedings of Robotics: Science and Systems*, 2024. doi:10.15607/RSS.2024.XX.062. URL https://www.roboticsproceedings.org/rss20/p062.html.

[11] T. Kwon, N. Di Palo, and E. Johns. Language models as zero-shot trajectory generators. *IEEE Robotics and Automation Letters*, 9(7):6728–6735, 2024. doi:10.1109/LRA.2024.3410155. URL https://arxiv.org/abs/2310.11604.

[12] H. Liu, X. Li, S. Yao, P. Shi, T. Zhou, J.-B. Huang, F. Huang, and J. Mao. Guava: An effective and universal harness for embodied manipulation. *arXiv preprint arXiv:2606.18363*, 2026. doi:10.48550/arXiv.2606.18363. URL https://arxiv.org/abs/2606.18363.

[13] A. Solar-Lezama. Program sketching. *International Journal on Software Tools for Technology Transfer*, 15(5–6):475–495, 2013. doi:10.1007/s10009-012-0249-7. URL https://link.springer.com/article/10.1007/s10009-012-0249-7.

[14] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki. SMT-based synthesis of integrated task and motion plans from plan outlines. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 655–662, 2014. doi:10.1109/ICRA.2014.6906924. URL https://moll.ai/publications/nedunuri2014smt-based-synthesis-of-integrated-task.pdf.

[15] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan. AutoTAMP: Autoregressive task and motion planning with LLMs as translators and checkers. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024. URL https://arxiv.org/abs/2306.06531.

[16] A. Curtis, N. Kumar, J. Cao, T. Lozano-Pérez, and L. P. Kaelbling. Trust the PRoC3S: Solving long-horizon robotics problems with LLMs and constraint satisfaction. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 1362–1383. PMLR, 2025. URL https://proceedings.mlr.press/v270/curtis25a.html.

[17] N. Kumar, W. Shen, F. Ramos, D. Fox, T. Lozano-Pérez, L. P. Kaelbling, and C. R. Garrett. Open-world task and motion planning via vision-language model generated constraints. *IEEE Robotics and Automation Letters*, 11(3):3366–3373, 2026. URL https://ieeexplore.ieee.org/document/11361071.

[18] L. Chen, S. R. Iyer, and Z. Kingston. Differentiable particle optimization for fast sequential manipulation. *arXiv preprint arXiv:2510.07674*, 2025. doi:10.48550/arXiv.2510.07674. URL https://arxiv.org/abs/2510.07674.

[19] Y. Lee, A. Z. Li, P. Huang, E. Heiden, K. M. Jatavallabhula, F. Damken, K. Smith, D. Nowrouzezahrai, F. Ramos, and F. Shkurti. STAMP: Differentiable task and motion planning via stein variational gradient descent. *IEEE Robotics and Automation Letters*, 10(6):6007–6014, 2025. doi:10.1109/LRA.2025.3561575. URL https://www.mit.edu/~k2smith/pdf/Lee_et_al_STAMP_2025.pdf.

[20] B. Sundaralingam, S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, N. Ratliff, and D. Fox. CuRobo: Parallelized collision-free robot motion generation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. URL https://research.nvidia.com/publication/2023-05_curobo-parallelized-collision-free-robot-motion-generation.

[21] W. Shen, C. R. Garrett, N. Kumar, A. Goyal, T. Hermans, L. P. Kaelbling, T. Lozano-Pérez, and F. Ramos. Differentiable GPU-parallelized task and motion planning. In *Proceedings of Robotics: Science and Systems*, 2025. doi:10.15607/RSS.2025.XXI.050. URL https://www.roboticsproceedings.org/rss21/p050.html.

[22] B. Han, Y.-W. Chao, E. Coumans, C. Eppner, J. Deng, S. Birchfield, and A. Murali. Graspgen-x: Cross-embodiment 6-dof diffusion-based grasping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20878–20889, 2026.

A Additional method details

A.1 Temporal constraints and shared variables

A request associates each constraint with a state or motion interval. For robot and object state $x(t)$ and a condition c returning one when satisfied, terminal and interval requirements are, respectively,

$$\begin{aligned} c(x(t_b)) &= 1, \\ c(x(t)) &= 1 \quad \forall t \in [t_a, t_b]. \end{aligned} \tag{3}$$

The interval boundaries can remain unknown during graph synthesis. A prescribed action sequence fixes physical ordering independently of the order in which its numerical parameters are evaluated. When expansion introduces additional segments within an interval, its constraints apply to every inserted segment. Waypoints constrain selected states. A supplied trajectory fixes the represented motion quantities, which can still leave other quantities unresolved. For example, an end-effector trajectory does not by itself specify the robot's joint trajectory.

Shared variables couple requirements introduced at different abstraction levels. During rigid transport of a mug,

$$\begin{aligned} \mathrm{FK}(q(t)) &= T_m(t) G_m, \\ e_z^\top R_m(t) e_z &\geq \cos\epsilon, \qquad t \in [t_a, t_b], \end{aligned} \tag{4}$$

where $q(t)$ is the robot configuration and FK gives its end-effector pose. The mug's world pose $T_m(t)$ has rotation $R_m(t)$, and G_m is the end-effector pose relative to the mug. The vector $e_z = (0, 0, 1)^\top$ denotes the upright axis in each frame, and ϵ is the permitted tilt. A handle-grasp requirement restricts G_m. An upright-transport requirement restricts $R_m(t)$. Their coupling through the kinematic relation constrains the same robot motion.

The compiler also connects consecutive segments through shared boundary states. Action models specify which quantities change and which persist. Agent-supplied requirements are retained directly or translated into equivalent relations. When a numerical value selects a particular expansion, the condition selecting that expansion remains part of the resulting graph and must hold after numerical refinement.

A.2 Conditional generation and optimization

Registered numerical procedures declare the variables they require and those they assign. A generator has the form

$$z_O \sim S(\cdot \mid z_I), \qquad I \cap O = \emptyset, \tag{5}$$

where I contains input variables and O contains output variables. Available values determine which procedures can run. Given a robot configuration, forward kinematics computes the end-effector pose. Given a desired end-effector pose, inverse kinematics proposes compatible configurations. Different registered procedures can therefore support different directions of computation for the same physical relation.

A generator can satisfy a subset of graph constraints while leaving others for subsequent refinement. Optimization updates the variable groups supported by its implementation, conditioned on the remaining values. Numerical functions supplied by the agent provide constraint evaluations once their arguments have values. They do not require every graph variable or evaluator to participate in one differentiable optimization problem. Agent-fixed quantities remain fixed throughout sampling and refinement. Acceptance requires validation against all graph constraints, including the translated request requirements.

The constraint graph describes the physical and task relations among variables. The dependencies of numerical procedures determine their evaluation order. A graph can therefore admit several numerical schedules without changing the trajectory requirements it represents. In particular, pooling inverse-kinematics or collision queries across graphs changes how those computations are executed while retaining each query's association with its source graph and assignment.

Algorithm 1: SPROUT: Graph synthesis and pooled numerical search

Input: Request r, scene $\hat{s}$, robot model M, registry $\mathcal{R}$, deadline T

$G_0 \leftarrow \mathrm{Build}(r, \hat{s})$
$\mathcal{A} \leftarrow \{G_0 \text{ with its known variable values}\}$
$\mathcal{S} \leftarrow \emptyset$
while $\mathcal{A} \neq \emptyset$ *and* $\mathrm{clock}() < T$ **do**
  Expand selected unresolved requirements using $\mathcal{R}$ and M, retaining candidate alternatives in $\mathcal{A}$
  $\mathcal{Q} \leftarrow$ ready queries from $\mathcal{A}$ with registered GPU implementations
  foreach *operation type o present in* $\mathcal{Q}$ **do**
    $\mathcal{B}_o \leftarrow \{q \in \mathcal{Q} : \mathrm{type}(q) = o\}$
    $Y_o \leftarrow \mathrm{GPU}_o(\mathrm{Gather}(\mathcal{B}_o))$
    Return Y_o to their source graphs and assignments in $\mathcal{A}$
  end
  Evaluate other ready procedures with their registered implementations
  Validate completed trajectories and store accepted pairs (G, z) in $\mathcal{S}$
  Refresh numerical candidates and retire exhausted alternatives in $\mathcal{A}$
end
if $\mathcal{S} = \emptyset$ **then**
  return *diagnostic feedback*
end
$(G^*, z^*) \leftarrow \arg\min_{(G,z)\in\mathcal{S}} J_G(z)$
return $d_{G^*}(z^*)$

Table 1: Objectives of the eight evaluation tasks. Descriptions follow the supplied task figure.

Task	Objective
Block stack	Stack the blocks in red, blue, green, yellow order from bottom to top.
Utensils upright	Place all green utensils upright in the pot.
Discard bottles	Put all small plastic bottles in the container.
Tool alignment	Arrange the tools in a line with their ends facing away.
Reorient mugs	Reorient all mugs with their openings facing upward.
Toast breakfast	Make toast and place it on the plate.
Whiteboard smiley	Draw a large smiley face on the whiteboard.
Tube tools	Use the available tools to retrieve the red block and place it on the plate.

A.3 Interleaved graph synthesis and pooled search

Algorithm 1 summarizes the search. The active set $\mathcal{A}$ contains partial or completed graphs and their current assignments. Expansion introduces graph alternatives while numerical procedures propose and refine variable values. A query becomes ready when its required inputs are available. Queries sharing an operation and compatible input dimensions form a batch, and each result returns to its source assignment. The set $\mathcal{S}$ stores validated graph–assignment pairs independently of further changes to active candidates. Its lowest-cost member determines the returned trajectory. If no candidate passes validation within the budget, diagnostic feedback reports unsuccessful completion rather than establishing infeasibility.

B Tasks and evaluation criteria

Tab. 1 lists the task objectives shown in Fig. 4. Each is instantiated in three scenes with different layouts, geometry, or clutter. The instruction remains the reference for task success even when an agent requests only an intermediate action. A feasible completion of an intermediate request need not complete the full task.

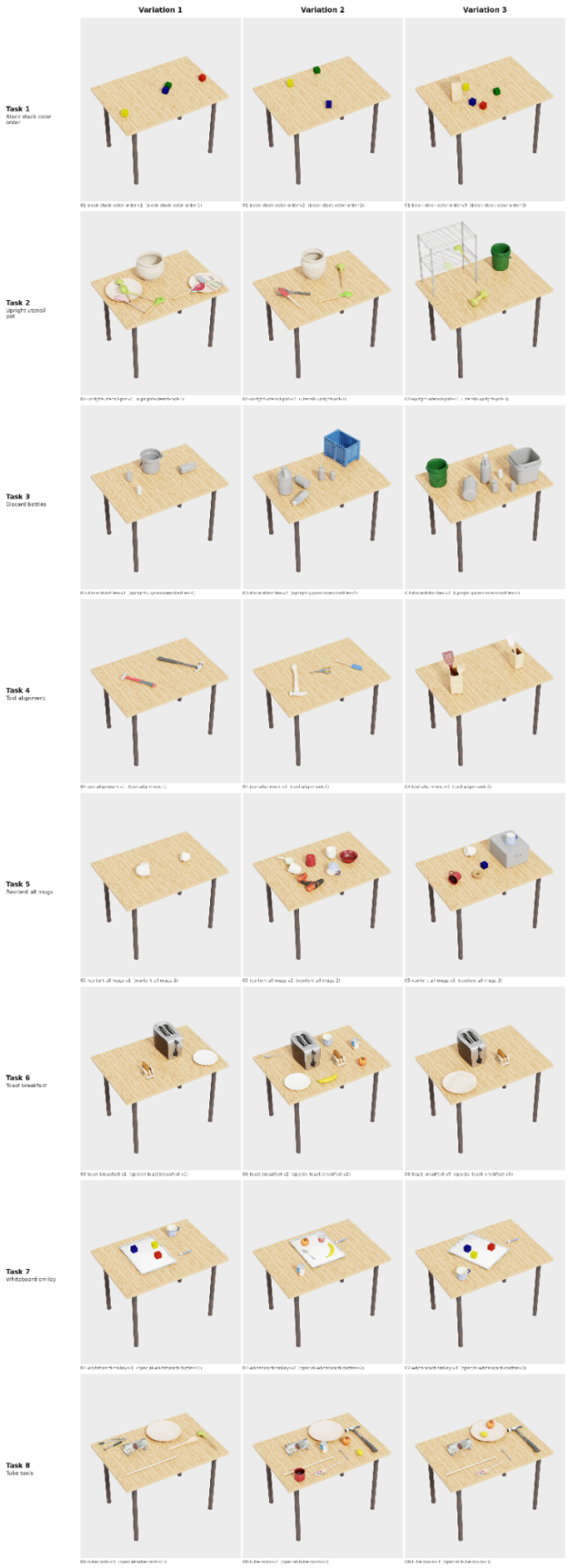


Figure 8: All task and scene variations that we evaluate over.