Representation Handoffs for OpenArm-Based Laboratory Mobile Manipulation

Yang Shen*, Chonghao Cheng*, Ziyi Zhao*, Jialuo Zhu†, Zhenyi Yi†,
Qi Zhao†, Jian Yang†, Yuhui Shi†, Chin-Teng Lin*
*University of Technology Sydney
†Southern University of Science and Technology

***Abstract*—Open-source robotics and foundation models have lowered the barrier to embodied AI, yet language-guided laboratory automation still requires reliable alignment from instructions and observations to safe actions. This field report presents an OpenArm-based mobile manipulation prototype for laboratory-style tasks, built by integrating dual OpenArm manipulators with a mobile base, vertical slide, RGB-D sensing, lidar-based mapping, ROS2/MoveIt execution, and profile-defined skill interfaces. The system is organized around representation handoffs: natural language requests are constrained into registered skill calls, sensor observations are grounded into maps and object poses, object priors provide role and skill constraints, and runtime bindings compile validated skills into executable motion goals. We use dry-run traces and startup checks to evaluate this integration path, showing how the prototype exposes missing calibration, incomplete object assets, and unfinished real scene visual grounding as explicit deployment blockers. These intermediate representations serve as practical debugging interfaces for integrating language, perception, planning, and robot safety in embodied systems. Code is available at https://github.com/yshenfab/xEI.**

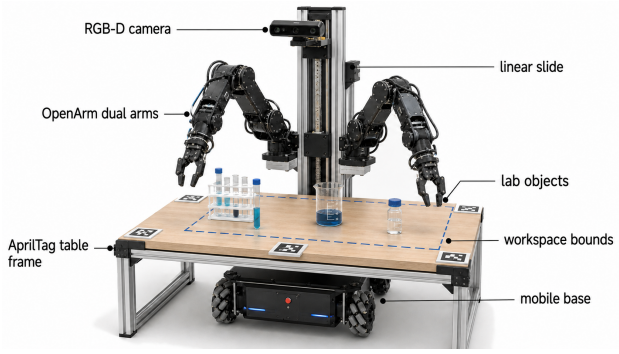


Fig. 1. Our OpenArm-based prototype for language-based laboratory manipulation. Our system targets tasks in which natural-language instructions must be grounded to laboratory objects and executed as manipulation skills, including moving containers, picking and placing items, handling tubes or racks, and preparing actions such as pouring or sample transfer.

I. Introduction

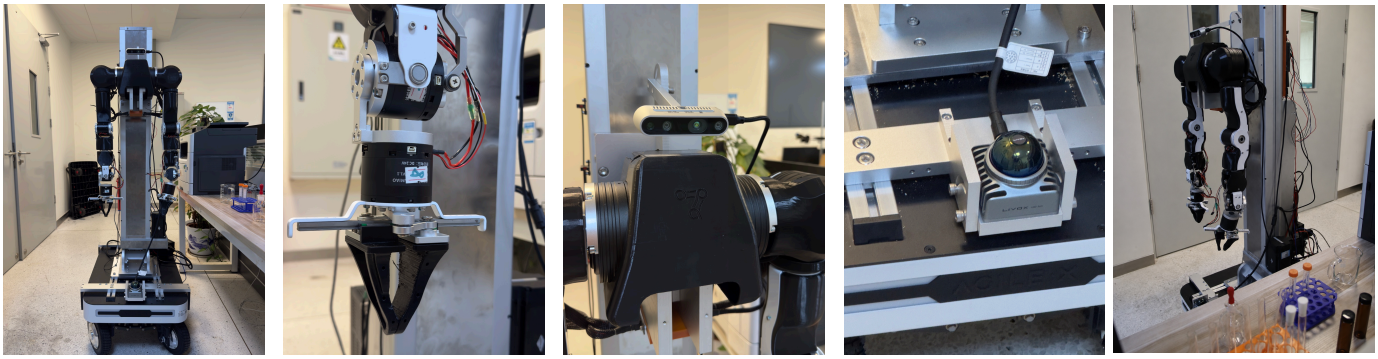
Open-source robot hardware, ROS-based middleware, and foundation model perception have made it easier to assemble embodied intelligence prototypes, but turning such prototypes into reliable task-execution systems remains difficult. In this field report, we introduce an OpenArm-based mobile manipulation prototype for laboratory-style embodied tasks. Our mobile manipulation prototype is built on OpenArm[1], a low-cost, open-source humanoid arm platform. The prototype integrates dual OpenArm manipulators, grippers, a vertical slide, a mobile base, RGB-D sensing, lidar-based mapping, ROS2/MoveIt execution, and profile-defined skill interfaces for laboratory-style manipulation tasks. The prototype is shown in Fig. 1. For mobile manipulation under language-based lab instructions, the main integration question is therefore not only how to represent the scene, but which representation is actionable by a real robot.

The workflow can be described as follows. A user gives a natural-language laboratory task, the robot moves to an appropriate scene origin, grounds relevant objects, asks a large language model (LLM) planner to emit for registered skill calls, validates the result against a profile-defined skill bank, expands each call into robot frame motion goals, and records the resulting trace. This workflow cannot be implemented safely by passing an LLM directly to robot control, and a 6D object pose is not enough either: it must be paired with object roles, skill contracts, frames, geometric operation parameters, safety limits, and execution feedback. We implement the OpenArm-based mobile manipulation prototype through representation handoffs: laboratory instructions are represented by validated skill-call sequences; lidar and RGB-D observations are represented by maps and object poses; object priors are represented by skill arguments; and runtime bindings are represented by executable motion goals.

The central observation from our deployment is that these interfaces need to be treated as explicit representations. We therefore frame the prototype around representation handoffs: instructions are constrained into registered skill calls, sensing outputs are grounded into frames and object states, object priors specify roles and admissible actions, and validated skills are bound to executable motion goals. This field report frames the prototype as a representation-centric system integration effort. The main contributions are summarized as follows:

1) A system view of the representation handoffs among lab instructions, maps, object poses, object priors, registered skill calls, runtime bindings, and motion goals;
2) An OpenArm-based integration that connects ROS2/MoveIt execution, mobile base navigation, vertical motion, RGB-D grounding, a skill bank, and deterministic skill templates;

[1]https://openarm.dev/



(a) Prototype body (b) Arm gripper (c) RGB-D camera (d) Lidar (e) Scene

Fig. 2. Hardware components of the OpenArm-based mobile manipulation prototype. (a) The prototype body integrates the OpenArm dual arms, vertical slide, and mobile base. (b) The arm gripper provides the end-effector interface for manipulation skills. (c) The RGB-D camera provides tabletop observations for object grounding and pose estimation. (d) The lidar supports mobile base mapping and navigation. (e) The application scenario.

3) Dry-run evidence and field lessons showing where missing calibration, object models, perception outputs, and capability contracts block real-robot deployment.

This system field report includes dry runs and mock perception traces, startup checks, and integration lessons.

II. System Overview

We built an OpenArm-based mobile manipulation prototype for laboratory-style embodied tasks. The main hardware components are shown in Fig. 2. The current software stack uses ROS2 middleware [9, 11], MoveIt/OpenArm arm execution [4, 12], linear slide, mobile base navigation, FoundationPose model [15], and AprilTag [10]. In our embodied system, the top-level agent is a deterministic orchestrator; the LLM is used only by a unified planner that must emit registered `SkillCall` records against the development profile skill bank. If the planner fails validation or cannot cover a requested capability, the system reports `unmapped_requests`. Physical execution is routed through runtime bindings, safety checks, and ROS2/OpenArm backends.

Figure 3 summarizes the pipeline and data flow. Before grounding the world, the robot may move to a configured scene origin. World grounding then combines usage-profile object priors, system objects, perception, or a FoundationPose response. The resulting `WorldObject` list, current robot state, planner-facing skill bank, and memory context form the input to a unified planner. The planner returns a `SkillPlan` containing ordered `SkillCall` records, explicit `unmapped_requests`, and safety notes. The output is validated against the skill bank, object role requirements, argument types, and step-id format; one validation-repair attempt is allowed before the request is reported as unmapped. Finally, the embodied runtime pre-checks the plan, refreshes the robot and world states before each skill call, resolves object- and target-specific operation parameters, validates safety limits, and executes a sequence of `MotionGoal` records. For more details, please refer to section III and our code repository.

This structure sits between classic robot middleware and recent language-based robotics. LLM and Vision-Language-Action (VLA) systems demonstrate powerful task-level priors and generalization [1, 5, 7, 8, 16].

III. Task-aware Representation Pipeline

A. Environment Representation

In our laboratory manipulation setting, the first representation problem is spatial: before the robot can ground objects or execute a skill, it must know how the navigation map, tabletop workspace, RGB-D camera, and OpenArm base relate to one another. Laser scans are accumulated into a navigation map for mobile base motion [3, 13], while RGB-D observations are expressed in calibrated camera and robot frames for manipulation. The calibration profile defines the coordinate frames used by the system, including the world, robot base, MoveIt base, tool, camera, lidar, slide, and table frames. It also records the static transforms and verification tolerances used to check frame alignment. Sensor topics, table bounds, and workspace bounds are treated as deployment constraints rather than implicit assumptions. AprilTag detections are used as table anchors: given detected tag poses and a known tag-to-table layout, the system can estimate or verify the tabletop frame consumed by downstream grounding and manipulation modules. Hand/eye and frame calibration are well-studied problems [14], but in a field system, they become operational representations. This spatial handoff makes deployment assumptions explicit, so that a dry-run or startup trace can explain which map frame, table frame, camera transform, and workspace bounds were used before object grounding or motion execution began.

B. Object Roles and Skill Interfaces

The second handoff is from laboratory objects to skill-facing symbols. The current object registry separates system objects from day-to-day laboratory objects: the registry keeps system boundaries and a usage-profile pointer, while `usage_profile.yaml` stores the current laboratory object priors. Each laboratory object is represented by a name, a point cloud or mesh path, aliases, physical dimensions, semantic roles, and exposed skills. For example, a tube can be

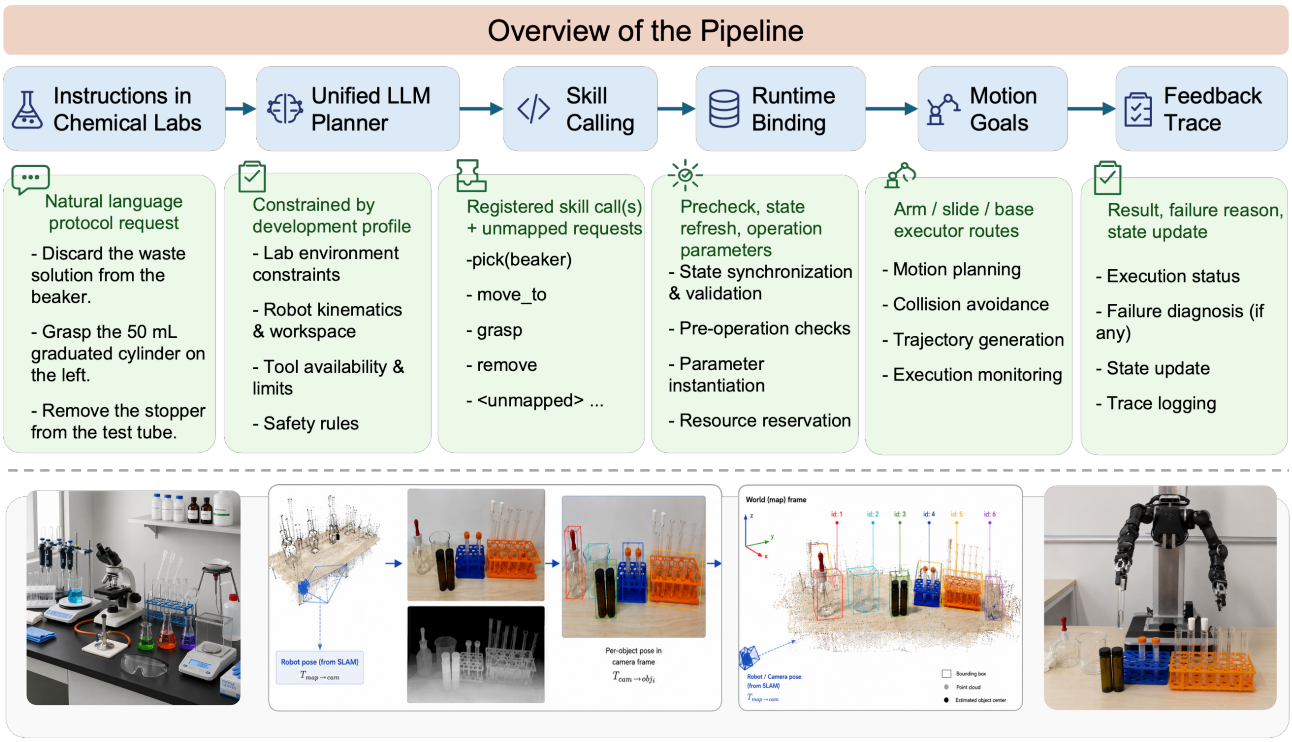


Fig. 3. Overview of the proposed language grounded laboratory manipulation pipeline. A unified LLM planner converts natural language instructions into executable robot skills through runtime binding and motion planning, while feedback traces monitor execution and update system states. Representative examples of world grounding and robotic operation are shown at the bottom.

represented as movable, a liquid container, a reaction vessel, an insertion target, and a sample container; a rack can be a fixture, a surface, and a place target. This is related to the affordance view of perception, where the meaning of an object is grounded in possible actions [2, 6]. For our prototype, affordances are deliberately engineered as explicit roles and skill memberships before they are learned. The development profile then exposes a skill bank: each skill declares an owner, target role requirements, typed arguments, preconditions, unsupported requests, and internal process notes.

C. Object Poses and Operation Parameters

The third handoff converts perception into runtime-facing objects. A `WorldObject` contains a 6D pose, frame id, confidence, roles, skills, and metadata. The FoundationPose service boundary is now a concrete HTTP/RGB-D protocol: the agent sends target queries, an output frame, camera intrinsics, RGB, depth, and optional masks; the GPU-side service returns object names, frames, confidence, skills, roles, poses, and mesh metadata [15]. The service also exposes health, model, and object endpoints, with a mock mode for software integration before the GPU perception stack and masks are available.

The prototype then enriches `WorldObject` records with deterministic operation parameters. Given pose, computed size, roles, available skills, and runtime bindings, the operation-parameter planner generates approach, grasp, lift, place, insert, pour, dispense, tap, swipe, and twist parameters. These defaults are stored as `perception_args`, `skill_args`, and `target_skill_args`, then merged by the runtime when compiling a skill call. This handoff serves as an interpretable bridge between perception outputs and skill templates, so that a dry-run trace can explain exactly why a particular pick pose, place pose, insertion depth, or pour clearance was chosen.

D. Goal and Task Representation

The final handoffs are from language to registered skills and from skill structure to motion. A `SkillPlan` stores registered `SkillCall` records, explicit `unmapped_requests`, and safety notes. Validation checks the selected skill, owner, target role requirements, typed arguments, object references, and step-id ordering. The runtime then applies skill-runtime bindings and expands each skill call into one or more `MotionGoal` records such as `move_to`, `open_gripper`, `set_do`, `slide_move`, or `nav_to_pose`. Every motion goal passes through a safety gate and is then routed to the appropriate executor. Before execution and each skill call, the runtime checks required objects, low-confidence objects, held-object preconditions, current robot state, and refreshed world grounding. If an object is missing, below confidence, violates a held-object precondition, or fails during execution, the runtime records `RuntimeFeedback` and stops.

This stop-on-failure policy is deliberately conservative. It prevents a language planner from papering over missing perception or unsupported skills. It also makes the representation

TABLE I
DRY-RUN TRACE STRUCTURE FOR THE INSTRUCTION "PUT SOURCE OBJECT ON DESTINATION OBJECT."

Stage	Representation	Observed dry-run
Scene setup	scene origin `MotionGoal`	Navigation target in map frame before grounding
World state	`WorldObject` list	Lab objects plus system objects with roles, skills, and frames
Planning	`SkillPlan` plus capability gaps	Registered skill-call sequence, or explicit gap when LLM access or validation fails
Runtime precheck	required objects, held-object state, safety gate	Target/object availability and goal safety checked before execution
Runtime execution	operation parameters and `MotionGoal` records	pick-above, pick-down, gripper close, lift, place, release, retreat
Feedback	`RuntimeFeedback`	Per-goal robot state, world snapshot, resolved args, executor result

handoffs easy to debug, i.e., a failure can be localized to scene origin preparation, world grounding, planner validation, skill coverage, operation-parameter resolution, runtime precheck, safety validation, executor routing, or hardware execution.

IV. PIPELINE EXECUTION

Table I summarizes the current evidence level. In a dry run, the system can execute all representation handoffs and generate a complete executable trace. For a simple transfer instruction with a valid LLM plan, the planner emits a registered `move_object` call against a movable source and a destination object satisfying target-role requirements, and the runtime expands it into a pick-place sequence. When the LLM is unavailable, or the request falls outside the skill bank, the same interface produces explicit `unmapped_requests`. The trace contains not only the final motion goals but also the world snapshot, grounded object metadata, resolved operation parameters, robot state, runtime precheck feedback, and executor feedback.

Startup checks also expose the current deployment boundary. The profile files are parseable, and the dry-run execution path is available, but the real-robot visual loop still depends on recording first-time calibration, replacing placeholder camera/lidar/slide transforms, populating object mesh and point-cloud assets, measuring table height, and enabling the FoundationPose service with real RGB-D input and masks or a detector. At this stage, the system validates the representation pipeline and software integration; currently, it does not provide real-world success rates for visual grasping, pouring, or insertion into evaluation.

V. FIELD LESSONS

Lesson 1: 6D pose is necessary but insufficient. The outputs of FoundationPose are valuable, but a mobile manipulation system needs pose plus frame, confidence, object identity, geometry, skills, roles, and target-specific operation parameters. The downstream planner should see an actionable object, not a raw perception result.

Lesson 2: Profiles are representations, not merely configurations. The usage, calibration, and development profiles encode three different forms of state: object priors, geometric deployment facts, and capability contracts. Treating these profiles as first-class representations made startup checks and dry-run traces more informative than ad hoc configuration files.

Lesson 3: Constrained intermediate forms make LLM planning debuggable. The LLM only proposes registered skill calls under a skill bank contract. It cannot invent arbitrary ROS commands, bypass role requirements, use undeclared arguments, or execute unregistered recovery. This design sacrifices flexibility, but it makes unsupported capabilities visible as `unmapped_requests` rather than hidden model assumptions.

Lesson 4: Deployment blockers are representation blockers. The remaining real-robot work is not just hardware bringup. It is the completion of missing representations: measured camera/lidar/slide transforms; table frame localization; object meshes; real RGB-D masks or detections; and calibrated workspace bounds. The field report value is that these blockers appear at the representation interfaces, where they can be tested.

VI. CONCLUSION

This report reframes an OpenArm-based embodied intelligence prototype as a set of representation handoffs across system integration. The central artifact is not a new perception model or controller, but an auditable path from laboratory instructions, maps, object poses, object priors, and skill bank contracts to validated skill calls, operation parameters, motion goals, and runtime feedback. The current implementation provides software-level and dry-run evidence that these handoffs are executable, while also making clear what remains before real-scene visual manipulation can be claimed. The next step is to replace placeholder calibration and model assets with measured field data, align the FoundationPose service with the current usage-profile object registry, enable strict real-scene grounding, and evaluate task-level failures across pick, place, insert, pour, clean, and screen-interaction tasks.

ACKNOWLEDGMENTS

This work is supported by the Shenzhen Fundamental Research Program under Grant No. JCYJ20200109141235597, National Natural Science Foundation of China under Grants No. 72401122 and 61761136008, Guangdong Basic and Applied Basic Research Foundation under Grants No. 2024A1515012241 and 2021A1515110024, Shenzhen Peacock Plan under Grant No. KQTD2016112514355531, Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant No. 2017ZT07X386, and the Australian Research Council (ARC) under Discovery Grant DP210101093 and DP220100803.

References

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

[2] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on robotics*, 30(2):289–309, 2013.

[3] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32(6):1309–1332, 2016.

[4] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *IEEE robotics & automation magazine*, 19(1):18–19, 2012.

[5] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[6] James J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979.

[7] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

[8] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[9] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science robotics*, 7(66):eabm6074, 2022.

[10] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*, pages 3400–3407. IEEE, 2011.

[11] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.

[12] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[13] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[14] Roger Y Tsai, Reimar K Lenz, et al. A new technique for fully autonomous and efficient 3 d robotics hand/eye calibration. *IEEE Transactions on robotics and automation*, 5(3):345–358, 1989.

[15] Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. Foundationpose: Unified 6d pose estimation and tracking of novel objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 17868–17879, 2024.

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