SPINE: Bridging the Cyber-Physical Gap with Agentic AI

Minkyu Ham[1,*], Dongho Kim[1,*], Chan Lee[1,*], Min Jun Kim[1], Yixi Zhang[2], Jiayi Wang[2], and Han Liu[1,2,†]

Website: magics-lab.github.io/SPINE-web

***Abstract*— Foundation models give robots powerful high-level reasoning, yet turning that intelligence into reliable physical action remains difficult: roboticists must still align device drivers, network interfaces, sensors, controllers, and safety constraints for each platform. This often-overlooked integration layer acts as the robot's spinal cord, translating high-level intent into coordinated physical behavior, and remains a primary bottleneck for scalable Embodied AI. Hence, we propose SPINE (Scalable Physical Integration with ageNtic Expertise), an agentic framework for systematically debugging and deploying bimanual robots for teleoperation. SPINE centers on two subagent-driven workflows: a profile builder that compiles robot-specific context and a debugger that uses that context to iterate through diagnosis, repair, and validation until teleoperation succeeds. Across two bimanual robots and 12 debugging scenarios, novice-operated SPINE debugged more accurately and efficiently than expert operators using vanilla LLMs. On DOBOT X-Trainer, SPINE improved success from 76% to 100% and reduced mean time-to-teleoperation by 30%; on AgileX PiPER, SPINE also achieved 100% success and reduced mean time-to-teleoperation by 38%. Together, these results show that SPINE leverages agentic AI to solve a critical integration bottleneck, helping bridge the cyber-physical gap that limits scalable real-world embodied AI. (Website: https://magics-lab.github.io/SPINE-web/)**

I. Introduction

Over the past decade, robot learning has expanded robots' ability to acquire skills from data and reuse experience across tasks and platforms. Deep visuomotor policies demonstrated learned mappings from camera images to motor commands [1], while Diffusion Policy and RT-1 extended learned manipulation through diffusion-based action generation and transformer policies trained on diverse demonstrations [2], [3]. Open X-Embodiment further demonstrated the benefits of sharing training data across robotic platforms [4]. Foundation models have accelerated these advances by bringing visual and linguistic knowledge into planning and control. SayCan grounds language-based plans in the skills a robot can execute, while Code as Policies translates instructions into programs that combine perception and control functions [5], [6]. Vision-language-action models, including RT-2, OpenVLA, and π_0, combine pretrained representations with robot demonstrations to generate actions from observations and instructions [7]–[9]. Together, these advances broaden robot capabilities and support generalization to new tasks, objects, and environments.

*These authors contributed equally to this work and are ordered alphabetically by last name. [1]Department of Statistics and Data Science, Northwestern University. [2]Department of Computer Science, Northwestern University. †Correspondence: hanliu@northwestern.edu.

Deploying these models still requires a functioning software, communication, and hardware stack, yet dependency conflicts, configuration errors, and component incompatibilities can prevent basic operation [10], [11]. Debugging this integration layer demands broad expertise because software and hardware faults can produce similar symptoms and persist in combination. A missing camera stream, for example, can reflect an incorrect device identifier, an incompatible driver, or a disconnected cable. If the configuration and cable are both faulty, correcting either one alone leaves the stream unavailable. Operators must therefore connect evidence from source code, configuration files, runtime logs, and physical inspection, then verify whether their interventions restore the complete system. Studies of robotics development identify substantial integration effort and platform-specific variation across electronics, mechanics, communication protocols, and software [12]. These demands create an expertise barrier for newcomers and can divert research effort from experimentation toward establishing basic operability.

Existing recovery and diagnostic methods address parts of this problem, but their evaluations do not establish complete recovery from combined software, communication, and physical faults. DROC and RACER address task-execution failures, while ASPIRE debugs control programs through existing robot interfaces [13]–[15]. Operators can also use LLM-based coding agents to inspect and repair software [18], [21], but code edits alone cannot reconnect hardware or establish that middleware communicates with the intended devices. The remaining gap is coordinating software repair, live middleware and device checks, and guided physical interventions until validation confirms that the complete robot operates.

Closing this deployment gap requires a coordinated process that gathers evidence across software and hardware, selects digital or physical interventions, and verifies recovery through live execution. In this work, *operational* specifically means *teleoperation-ready*: an operator can control the follower arms and grippers through the master arms while viewing the required camera streams. This definition does not imply readiness for autonomous policy execution. Making this process systematic and accessible would reduce the expertise required to establish the working physical systems that robot learning depends on.

To address this gap, we introduce SPINE (Scalable Physical Integration with ageNtic Expertise), an agentic harness for restoring teleoperation with minimal robotics expertise. SPINE runs within Claude Code [21] and centers on two subagent-driven workflows: a profile builder and a debugger

(Fig. 1). The profile builder compiles documentation, known-good source code, and operator-confirmed facts into a persistent reference for diagnosis, repair, and validation. The debugger combines this reference with live evidence and incident memory to coordinate parallel diagnosis, targeted software repairs, and step-by-step guidance for physical interventions. It iterates through diagnosis, repair, and validation, closing a case only when the complete teleoperation pipeline and profile-defined readiness checks pass in the same validation cycle. SPINE retains incident outcomes and validation evidence to inform subsequent debugging sessions.

We evaluate novice-operated SPINE across twelve software-only, hardware-only, and hybrid debugging scenarios on DOBOT X-Trainer and AgileX PiPER. We compare it with human operators using Claude Code and the same underlying language model, but without SPINE's structured workflow. SPINE resolved all implanted faults on both platforms (Table II). On DOBOT, it increased the mean fraction of faults resolved from 76% to 100% and reduced mean time-to-teleoperation by 30%; on PiPER, it increased fault resolution from 99% to 100% and reduced mean time-to-teleoperation by 38%. These results demonstrate more complete and efficient teleoperation recovery across the two platforms, including scenarios that require both software repair and physical intervention.

DOBOT ablations suggest that both the robot profile and hardware playbook contribute to SPINE's performance (Table III). Removing them individually reduced mean fault resolution from 100% to 79% and 57%, respectively, and increased mean debugging duration. With only one trial per scenario for each ablated variant, these findings provide preliminary support for robot-specific context and structured physical guidance.

Together, these results support a role for agentic AI beyond controlling already functional robots: helping operators make them teleoperation-ready. SPINE advances this direction through structured, validated debugging that bridges software and hardware, making teleoperation recovery more accessible and efficient.

II. Related Work

SPINE is closest to work on robot failure recovery, language-guided robot systems, and tool-using software agents. The distinction is the starting state: SPINE assumes that the robot may not yet expose a reliable action interface, so recovery must span source, middleware, devices, and operator-visible hardware before task execution can resume.

A. Robot deployment, fault diagnosis, and recovery

ROS and ROS 2 standardize communication among robot drivers, sensors, controllers, and applications [16], [22], but deployment state remains distributed across launch files, environment variables, device identifiers, network settings, and vendor libraries. Industrial robotics studies likewise identify integration, maintenance, heterogeneous interfaces, and fault recovery as persistent barriers to dependable operation [23]. These systems provide the substrate on which SPINE operates, but they do not assemble platform-specific evidence into an end-to-end recovery decision.

Systems work also reduces particular deployment costs. SARA-RT targets the computational efficiency of large robot transformers, and Robo-DM targets storage and loading for large multimodal trajectory collections [24], [25]. RoboCrowd shows that carefully designed interfaces can broaden participation in bimanual teleoperation data collection [26]. These contributions make operational systems more scalable, but retain the assumption that the robot stack can already expose its intended sensing and control interfaces.

Robot failure-recovery research typically begins later in the stack. Self-Recovery Prompting revises prompts when a service robot encounters missing information, incorrect plans, or plan-execution failures [27]; CoPAL replans when semantic or physical constraints make an action sequence infeasible [28]; VLM-based recovery methods improve motion- and task-level corrections through optimized visual and textual prompts [29]; and RACER trains a language-guided visuomotor policy on augmented failure-recovery trajectories [14]. These methods recover behavior after an executable robot interface exists. SPINE addresses pre-operational faults that can prevent perception, middleware bringup, or teleoperation from becoming available in the first place.

B. Language-guided robot agents

Language and multimodal models provide increasingly general interfaces to robot behavior. SayCan grounds language-model proposals in learned affordances, Code as Policies converts instructions into executable programs, and PaLM-E and OpenVLA connect language and vision to embodied state or action [5], [6], [8], [30]. VLFM combines language semantics with geometric maps for zero-shot navigation on a physical robot [31], while CoPAL and SELP add feedback or formal constraints to generated plans [28], [32]. DROC instead distills online language corrections into a retrievable knowledge base, allowing later tasks to reuse prior human feedback [13].

Closer to the systems boundary, ROS-LLM maps natural-language task requirements into ROS behaviors with environment and human feedback, while ROSA exposes ROS capabilities through constrained tools and natural-language interaction [19], [33]. Such interfaces lower the expertise needed to invoke an operational robot. SPINE complements them by recovering the device and middleware path beneath that interface. Its profile stores not only callable capabilities but also source contracts, live probes, hidden readiness checks, and operator-confirmed physical playbooks needed when invocation fails.

C. Tool-using agents, memory, and validation

Software-engineering agents demonstrate that language models can inspect repositories, invoke tools, revise code, and use execution feedback over multiple steps [18], [34]–[36]. Their closure signals remain digital: a unit test passes, a

build succeeds, or a repository issue is resolved. Robot operationalization introduces observations that source inspection cannot settle, including cable state, device enumeration, bus connectivity, controller readiness, and the outcome of a live robot-level probe.

Persistent context and validation therefore play different roles in SPINE. The category profile preserves platform contracts and executable diagnostics, while incident memory retains only fixes supported by a passed validation record. The runtime does not treat an LLM's diagnosis as resolution. It adjudicates parallel evidence, routes the next fix according to whether it is digitally or physically actionable, and reruns the complete goal-conditioned readiness bundle. This combines the iterative tool use of software agents with a cyber-physical closure condition.

III. SPINE System

SPINE is an agentic harness for restoring teleoperation through two Claude Code skills: `/spine-profile-build` and `/spine-debug`. Language-model agents interpret evidence and propose repairs, while a Python backend manages execution, persistent state, and validation gates.

A. Debugging Workflow

SPINE establishes robot-specific context during setup and reuses it during subsequent debugging sessions (Fig. 1). For a new robot installation, the operator supplies documentation, a known-good software repository, and setup facts to the profile builder. The builder compiles and checks a reference profile that defines operating requirements, troubleshooting procedures, and validation criteria. The retained source evidence, profile, and incident memory accumulated across sessions form the *persistent robot context*.

When teleoperation fails, the debugger combines this context with live evidence in an observe–triage–repair–verify loop. It identifies active failures, coordinates parallel diagnosis, and selects software repairs or guided physical interventions. Verification returns unresolved failures to diagnosis and permits closure only after the teleoperation pipeline and required readiness checks pass. Incident records preserve the resulting evidence for future sessions.

We next detail how the profile builder constructs and validates robot-specific context (Section III-B), followed by how the debugging skill implements each stage of the recovery loop and maintains incident memory (Section III-C).

B. Robot Profile Building

The `/spine-profile-build` skill converts setup evidence into a structured, traceable reference. A Python compiler assembles source text, file references, configuration data, and command relationships into an evidence bundle. Nine extraction subagents work in parallel: eight cover robot identity, components, interfaces, software, configuration, procedures, safety, and diagnostics; the ninth reconstructs the teleoperation pipeline. Each receives task-specific instructions and an evidence packet, then returns a JSON draft. The compiler merges these drafts with deterministic extraction and retains references to supporting sources.

Eight category JSON files and a manifest form the canonical profile. Runtime adapters derive *contracts* that specify operating requirements, *probes and checks* that evaluate those requirements, and *hardware playbooks* that pair physical troubleshooting steps with verification procedures. A directed acyclic command graph represents the teleoperation pipeline, recording commands, working directories, execution environments, dependencies, and success criteria. Its nodes distinguish one-shot setup, persistent bringup, and the final teleoperation command. Mutable facts such as camera serials and device paths remain assumptions that live evidence can challenge.

Operator review grounds the profile in the physical installation. The builder requests confirmation of the command graph, required interfaces, hardware playbooks, and each *hidden readiness check*. These checks target configuration or device problems that a short teleoperation run may miss; they contain no hidden fault annotations. They use static inspection or non-motion probes as appropriate. The workflow also requires a recorded successful execution of the confirmed command graph.

A final review stage combines nine parallel curator subagents with one adjudicator. Curators propose evidence-backed revisions, and the adjudicator resolves cross-category conflicts before the compiler applies accepted changes. A deterministic *profile doctor* checks schema consistency, evidence coverage, runtime compatibility, command-graph structure, operator confirmations, and build-time validation. The builder seals the profile only after these checks pass and required questions have answers. The supporting `/spine-init` skill wraps setup and initializes per-robot memory.

C. Evidence-Guided Debugging

The `/spine-debug` skill implements the recovery loop through Python CLI commands: `debug start`, `debug after-fix`, `debug continue`, and `debug finish`. The backend persists validation results, active blockers, software changes, operator observations, and hardware-step history as JSON. Claude Code's `Workflow` tool dispatches diagnostic subagents. An optional FastMCP server exposes supporting tools for profile access, checks, validation lookup, and incident memory; the core loop does not require MCP for orchestration.

1) Observe: Evidence Collection: The debugger loads the profile, accepts supplied terminal output or executes the declared teleoperation pipeline, and runs teleoperation-scoped readiness checks. It consolidates failures into a *blocker ledger* that links each active failure to its evidence and originating check. Each validation cycle refreshes this ledger rather than carrying historical symptoms forward as unresolved failures.

The runner executes setup commands in dependency order, starts persistent bringup processes, and observes the final teleoperation command for the profile-defined window,

which defaults to 10 seconds. It captures process output, status, and declared success or failure signals, blocks commands with failed prerequisites, and cleans up its processes afterward. An invalid command graph triggers profile repair. The `/spine-check` skill can gather focused evidence but cannot independently establish readiness.

2) Triage: Parallel Diagnosis: The main diagnostic path dispatches four read-only subagents through one `parallel()` call. They examine terminal evidence, software and configuration differences, hardware visibility, and readiness-check relevance. Each uses shared robot context and role-specific evidence to produce a structured report containing findings, evidence references, confidence, and blocker assessments. Their instructions prohibit workspace edits, operator prompts, and success declarations.

The `debug diagnostics-adjudicate` command checks report completeness and association with the current diagnostic batch, then aggregates assessments using deterministic triage rules. The main agent selects software repair, physical intervention, or further evidence gathering. This separation prevents a device-related symptom, such as a missing camera stream, from automatically implying a hardware fault.

3) Repair: Digital and Physical Interventions: Software repair targets evidence-supported deviations without weakening operating requirements. The main agent edits relevant code, configuration, or environment settings and records the changes through `debug after-fix`. The skill prohibits apparent fixes such as disabling required sensors, lowering expected device counts, or replacing live teleoperation with a startup test.

Physical repair follows a component-specific playbook, presenting one action at a time. The operator performs the action and reports the outcome; `debug continue` records the response and runs the designated check or captures a required manual observation. Persistent step history prevents unnecessary repetition. Targeted checks can guide subsequent playbook steps, but they do not replace full-system verification.

4) Verify: Closure and Incident Memory: The `debug finish` command permits successful closure only when live teleoperation and hidden readiness checks pass in the same latest validation cycle after the most recent intervention, with no unresolved blockers. Software edits and confirmed hardware actions invalidate earlier success evidence. Failed validation refreshes the blocker ledger and returns new evidence to triage, following the feedback path in Fig. 1. The resulting readiness claim covers only the configured pipeline, checks, and observation window.

The `stores-memory` skill treats previous incidents as diagnostic context, not proof of the current cause. At closeout, `/spine-closeout` records symptoms, attempted repairs, suspected or confirmed causes, corrective actions, and validation evidence through `spine.bug.log`. It checks validation through `spine.validation.status`, and the logging backend requires a passing validation reference for a resolved record. Unvalidated cases retain a diagnosed status and their remaining blocker, preserving useful history without presenting hypotheses as verified repairs.

TABLE I: Injected debugging scenarios. D/P denote DOBOT/PiPER; SW, HW, and HY denote software-only, hardware-only, and hybrid faults.

ID	Description
DOBOT X-Trainer	
D-SW1	**Software:** We assign mismatched communication endpoints to the robot server and teleoperation client and alter launcher behavior to occupy a required local port.
D-SW2	**Software:** We alter the startup compatibility check to reject the installed controller or library version and assign a nonexistent serial number to one camera role. Packages and firmware remain unchanged.
D-HW1	**Hardware:** We latch the emergency-stop button, preventing arm actuation without changing software or connections.
D-HW2	**Hardware:** We unplug the right wrist camera cable and replace the left wrist camera cable with a known-broken cable.
D-HW3	**Hardware:** We disconnect one leader-arm cable and one follower-arm Ethernet cable, interrupting operator input and follower control.
D-HY1	**Hardware:** We replace the right wrist camera with a known-broken unit. **Software:** We assign that camera role a nonexistent serial number.
D-HY2	**Hardware:** We unplug the robot-control Ethernet cable. **Software:** We change both controller IP addresses in the launcher to the wrong subnet, leaving host network settings unchanged.
AgileX PiPER	
P-SW1	**Software:** We configure an incorrect USB bus address for the left CAN adapter and prioritize user-site Python packages over the intended environment, causing an incompatible Pinocchio import.
P-SW2	**Software:** We assign an incorrect serial number to the left wrist camera and change the teleoperation launch file's inverse-kinematics parameter path to a nonexistent file.
P-HW1	**Hardware:** We disconnect the left follower arm's CAN signal wire or connector and unplug the left wrist camera's USB cable.
P-HY1	**Hardware:** We unplug the left wrist camera's USB cable. **Software:** We assign that camera an incorrect serial number in its launch file.
P-HY2	**Hardware:** We unplug the left USB-to-CAN adapter's USB connection. **Software:** We increase the setup script's expected adapter count from two to three.

IV. Experimental Design

We evaluate SPINE by introducing a carefully selected set of software, hardware, and combined faults into otherwise functional teleoperation systems. An administrator prepares each faulty system without revealing the implanted faults to the participant. A robotics novice then attempts to restore teleoperation using SPINE, while human operators independently attempt the same scenarios using Claude Code and their own expertise.

A. Robot Platforms

The DOBOT X-Trainer benchmark uses a bimanual teleoperation platform with two follower arms, two leader arms, RealSense cameras, a control workstation, and a vendor SDK over static Ethernet. The AgileX PiPER evaluation uses a ROS Noetic and CAN-based bimanual platform with four PiPER 6-DOF arms, USB-to-CAN adapters, and RealSense cameras. The two platforms differ in actuation bus, middleware, camera integration, and emergency-stop behavior,

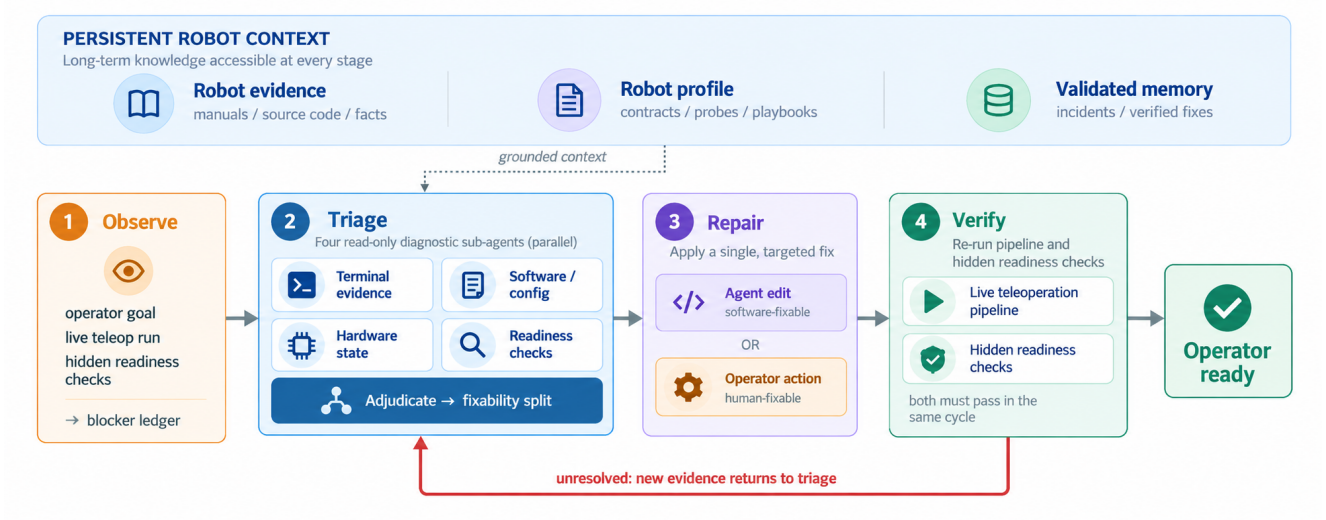
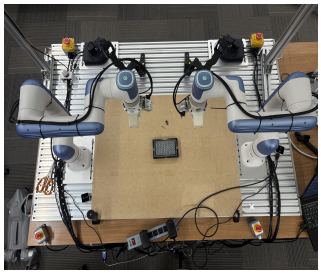


Fig. 1: SPINE separates persistent robot knowledge from a goal-conditioned recovery loop. Triage dispatches four read-only diagnostic sub-agents in one parallel call and adjudicates their reports into a fixability split; repair is a separate stage that applies either a software edit or one operator action. Verification reruns the live teleoperation pipeline and hidden readiness checks together, and unresolved cases return to triage with new evidence rather than restarting observation.

which allows us to test whether the same debugging structure transfers beyond one robot.



(a) DOBOT X-Trainer

(b) AgileX PiPER

Fig. 2: Bimanual robot platforms used in the evaluation.

B. Bug Cases

We construct seven DOBOT scenarios and five PiPER scenarios across three categories: software-only faults alter runtime code or configuration; hardware-only faults change physical connections, components, or safety-control states; and hybrid faults combine software and physical changes. Each scenario contains two faults except D-HW1, which contains a single latched emergency stop. See Table I for detailed descriptions of the faults introduced in each scenario. Table I in the appendix specifies the software mutations and physical interventions for every scenario.

We select these faults to cover device identification, communication, runtime compatibility, configuration, and physical connectivity. The compound scenarios test whether participants identify all required repairs rather than stop after correcting the first visible error. Hybrid cases further test whether participants distinguish software misconfiguration from physical failure when both affect the same subsystem. These scenarios provide targeted coverage of integration problems rather than an exhaustive or frequency-weighted sample of robot failures.

C. Experimental Protocol

We compare novice-operated SPINE with human operators on the same bug cases for both DOBOT X-Trainer and AgileX PiPER. An administrator prepares each trial from the reference software stack, implants the specified faults, and withholds the scenario identity and implantation records from participants.

In the SPINE condition, the agent uses a robot profile built from the working system before fault implantation. The novice initiates debugging, supplies requested observations, and follows SPINE's physical instructions, while SPINE directs diagnosis, software repair, and revalidation. In the human-operator condition, participants use standard Claude Code with Claude Sonnet 5.0 as the only permitted language model and independently choose diagnostic commands, software edits, and physical interventions. They receive the same underlying reference materials, but not SPINE's compiled profile, debugging skills, orchestration, or incident memory.

Both conditions aim to restore the standard teleoperation workflow. Successful recovery requires the operator to demonstrate that the follower arms and grippers respond to commands from the master arms while the required camera streams remain available; launching processes without errors does not suffice. SPINE additionally enforces its profile-declared validation pipeline and readiness checks before declaring completion.

V. Results

We evaluate fault resolution and debugging efficiency using the per-scenario outcomes in Table II. OSS measures the fraction of implanted faults resolved within each trial, rather than the fraction of trials that achieve complete recovery. Timing statistics include unsuccessful trials, so we interpret them alongside OSS.

TABLE II: Per-scenario outcomes. Scenario IDs encode platform and fault class, with D/P denoting DOBOT/PiPER and SW/HW/HY denoting software-only, hardware-only, and hybrid faults. OSS is reported as mean±SD over human-operator trials and as a decimal success score for SPINE. TTO is reported as mean±SD when repeated measurements are available. DOBOT human trials pool seven operators, and PiPER human trials pool nine operators.

		SPINE			Human operators	
ID	Description	n	OSS	TTO	n	OSS	TTO
DOBOT X-Trainer							
D-SW1	Mismatched local endpoint and busy port	3	1.00 ± 0.00	$6{:}20 \pm 0{:}01$	7	0.36 ± 0.38	$7{:}28 \pm 3{:}05$
D-SW2	Stale environment, version gate, camera ID	3	1.00 ± 0.00	$6{:}41 \pm 1{:}19$	7	0.57 ± 0.45	$20{:}11 \pm 13{:}25$
D-HW1	Latched emergency stop	3	1.00 ± 0.00	$13{:}35 \pm 0{:}13$	7	1.00 ± 0.00	$20{:}46 \pm 6{:}12$
D-HW2	Broken wrist-camera cabling	3	1.00 ± 0.00	$9{:}59 \pm 0{:}34$	7	1.00 ± 0.00	$17{:}54 \pm 17{:}52$
D-HW3	Leader and follower links unplugged	3	1.00 ± 0.00	$11{:}18 \pm 1{:}44$	7	1.00 ± 0.00	$13{:}01 \pm 7{:}47$
D-HY1	Broken wrist camera plus stale ID	3	1.00 ± 0.00	$14{:}55 \pm 0{:}55$	7	0.64 ± 0.38	$11{:}55 \pm 7{:}44$
D-HY2	LAN unplug plus wrong subnet	3	1.00 ± 0.00	$9{:}34 \pm 0{:}32$	7	0.71 ± 0.27	$12{:}43 \pm 9{:}14$
Overall			$\mathbf{1.00 \pm 0.00}$	$\mathbf{10{:}20 \pm 3{:}11}$		$\mathbf{0.76 \pm 0.36}$	$\mathbf{14{:}51 \pm 10{:}44}$
AgileX PiPER							
P-SW1	CAN binding and Python path drift	3	1.00 ± 0.00	$11{:}32 \pm 0{:}11$	9	0.94 ± 0.17	$16{:}00 \pm 10{:}16$
P-SW2	Camera ID and IK config drift	3	1.00 ± 0.00	$8{:}28 \pm 0{:}06$	9	1.00 ± 0.00	$10{:}56 \pm 13{:}01$
P-HW1	Camera and CAN links unplugged	3	1.00 ± 0.00	$2{:}45 \pm 0{:}22$	9	1.00 ± 0.00	$13{:}36 \pm 13{:}23$
P-HY1	Camera unplug plus stale ID	3	1.00 ± 0.00	$10{:}44 \pm 0{:}19$	9	1.00 ± 0.00	$13{:}43 \pm 8{:}07$
P-HY2	CAN adapter unplug plus count drift	3	1.00 ± 0.00	$5{:}33 \pm 0{:}04$	9	1.00 ± 0.00	$8{:}14 \pm 7{:}27$
Overall			$\mathbf{1.00 \pm 0.00}$	$\mathbf{7{:}48 \pm 3{:}27}$		$\mathbf{0.99 \pm 0.07}$	$\mathbf{12{:}30 \pm 10{:}35}$

A. DOBOT X-Trainer

SPINE improved fault resolution and reduced mean time-to-teleoperation across the seven DOBOT scenarios. SPINE achieved an OSS of 1.00 in every scenario, compared with an overall human-operator score of 0.76. Mean time-to-teleoperation decreased from 14 min 51 s to 10 min 20 s, a 30% reduction. The success advantage arose entirely from software-only and hybrid scenarios: human operators achieved mean OSS values of 0.36 and 0.57 on D-SW1 and D-SW2, and 0.64 and 0.71 on D-HY1 and D-HY2. Both conditions resolved all faults in the three hardware-only scenarios.

SPINE reduced mean time-to-teleoperation in six of seven scenarios, but its advantage did not always combine faster execution with more complete recovery. On D-HY1, which combined a broken camera with an incorrect camera identifier, SPINE took 14 min 55 s compared with 11 min 55 s for human operators, while achieving complete fault resolution rather than a mean OSS of 0.64. This exception highlights why shorter debugging duration alone does not establish better recovery performance.

B. AgileX PiPER

SPINE's principal advantage on PiPER was faster recovery while maintaining complete fault resolution. Across five scenarios, SPINE achieved an overall OSS of 1.00 compared with 0.99 for human operators, and reduced mean time-to-teleoperation from 12 min 30 s to 7 min 48 s, a 38% reduction. SPINE recorded a lower mean time in every scenario. The largest reduction occurred on P-HW1, which combined disconnected camera and CAN links: mean time fell from 13 min 36 s to 2 min 45 s, approximately 80%. Human operators already achieved complete fault resolution in four scenarios; their remaining unresolved faults occurred in P-SW1. These results extend the observed efficiency advantage to a second platform with a different middleware and communication stack, while showing only a small absolute improvement in success.

C. Ablation Study

Removing the robot profile reduced overall fault resolution and increased average debugging duration on DOBOT (Table III). Mean OSS fell from 1.00 to 0.79, while mean duration increased from 10 min 20 s to approximately 17 min. The profile-free variant resolved only half the faults in D-HW2 and none in D-HY1. Although it completed several other scenarios faster than full SPINE, these local improvements did not offset its incomplete recovery and longer durations elsewhere.

TABLE III: DOBOT ablation study outcomes comparing full SPINE against two ablated variants. SPINE results report mean OSS and mean TTO over $n = 3$ trials per scenario, while each ablation result reports a single trial ($n = 1$).

	SPINE		No Profile		No Hardware Playbook	
ID	OSS	TTO	OSS	TTO	OSS	TTO
D-SW1	1.00	6:20	1.00	4:40	1.00	20:47
D-SW2	1.00	6:41	1.00	6:37	1.00	5:23
D-HW1	1.00	13:35	1.00	11:01	0.00	45:00
D-HW2	1.00	9:59	0.50	45:00	0.00	45:00
D-HW3	1.00	11:18	1.00	23:52	1.00	32:55
D-HY1	1.00	14:55	0.00	20:37	0.50	25:50
D-HY2	1.00	9:34	1.00	7:25	0.50	45:00
Overall	**1.00**	**10:20**	**0.79**	**17:00**	**0.57**	**31:25**

Removing the hardware playbook produced a larger aggregate performance loss. Mean OSS fell to 0.57, and mean duration increased to 31 min 25 s, approximately three times that of full SPINE. This variant resolved no faults in D-HW1 or D-HW2 and only half the faults in each hybrid scenario. Both ablated variants retained complete fault resolution on the software-only scenarios, concentrating their observed failures in cases that required physical intervention.

The ablations provide preliminary evidence that robot-specific context and structured physical troubleshooting contribute to reliable recovery. However, each ablated condition includes only one trial per scenario, so these observations do not establish statistically reliable effect sizes or isolate the mechanisms behind individual failures.

VI. Discussion

Our results suggest that structured agentic assistance can improve robot debugging even when human operators already have access to a capable coding agent. The benefits differ across platforms: on DOBOT, SPINE improved fault resolution primarily in software-only and hybrid scenarios; on PiPER, where human operators already resolved nearly all faults, its principal advantage was faster recovery. These findings highlight two distinct requirements for effective debugging: identifying all faults that prevent operation and resolving them efficiently. The comparison evaluates the complete human–agent workflow, not the language model's reasoning ability in isolation.

The ablations suggest that robot-specific context and physical troubleshooting guidance help connect software diagnosis to hardware recovery. Both ablated variants retained complete fault resolution on software-only scenarios but left faults unresolved in hardware-only and hybrid cases. This pattern is consistent with the intended roles of the profile and hardware playbooks: the former establishes platform-specific expectations, while the latter structures interventions that require physical access. However, one trial per scenario for each ablation provides only preliminary evidence and cannot establish reliable effect sizes or explain individual failures conclusively.

Our experimental design limits the breadth of these conclusions. We evaluate controlled faults on two bimanual platforms, with few SPINE repetitions, rather than naturally occurring failures across diverse deployments. Furthermore, debugging durations include unsuccessful trials, so shorter times do not necessarily indicate faster successful recovery. The timing comparisons also begin after profile construction and therefore do not measure total setup effort. Larger studies should include additional embodiments, more operators, naturally occurring faults, and the cost of building and maintaining robot profiles.

SPINE's readiness claim also depends on the quality and coverage of its reference information and validation procedures. A passing teleoperation run and readiness checks establish operation only within the configured checks and observation window; they do not establish long-term reliability, autonomous policy readiness, or comprehensive safety. SPINE still requires operator review and human access for physical interventions. Future work should examine incomplete or outdated profiles and failures that emerge during sustained operation, while preserving the distinction between observed recovery and broader operational guarantees.

VII. Conclusion

SPINE treats teleoperation readiness as a cyber-physical debugging problem that requires coordinated software repair, physical intervention, and live verification. Its profile builder and debugger turn robot-specific evidence into a reusable recovery workflow. Across two bimanual platforms, novice-operated SPINE resolved all implanted faults and reduced mean time-to-teleoperation relative to human operators using Claude Code. Within the evaluated scope, these results show how agentic AI can help bridge the gap between capable robot intelligence and the functioning physical systems it requires, making teleoperation recovery more accessible and efficient.

VIII. Acknowledgements

We thank all the people who volunteered to be human operators for our experiments and/or run experiments for our SPINE agent. Their names are: Guo Ye, Jihai Zhao, Tony Dong, Jiale Chen, Vincent Cheng, Soyeon Park, Dennis Wu, Hongyu Chen, Donggyun Lee, and Mingxian Wang. We also want to thank Guo Ye for the numerous conversations we had with him about the design of SPINE, which helped inspire certain elements of it. AI tools (Claude and ChatGPT) helped us generate and refine the figures in this paper.

References

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016. [Online]. Available: https://jmlr.org/papers/v17/15-522.html

[2] C. Chi *et al.*, "Diffusion policy: Visuomotor policy learning via action diffusion," in *Proceedings of Robotics: Science and Systems XIX*, 2023, doi: 10.15607/RSS.2023.XIX.026.

[3] A. Brohan *et al.*, "RT-1: Robotics transformer for real-world control at scale," in *Proceedings of Robotics: Science and Systems XIX*, 2023, doi: 10.15607/RSS.2023.XIX.025.

[4] Open X-Embodiment Collaboration, "Open X-embodiment: Robotic learning datasets and RT-X models," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2024, pp. 6892–6903, doi: 10.1109/ICRA57147.2024.10611477.

[5] B. Ichter *et al.*, "Do as I can, not as I say: Grounding language in robotic affordances," in *Proceedings of the 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 205, 2023, pp. 287–318. [Online]. Available: https://proceedings.mlr.press/v205/ichter23a.html

[6] J. Liang *et al.*, "Code as policies: Language model programs for embodied control," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2023, pp. 9493–9500, doi: 10.1109/ICRA48891.2023.10160591.

[7] B. Zitkovich *et al.*, "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Proceedings of the 7th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 229, 2023, pp. 2165–2183. [Online]. Available: https://proceedings.mlr.press/v229/zitkovich23a.html

[8] M. J. Kim *et al.*, "OpenVLA: An open-source vision-language-action model," in *Proceedings of the 8th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 270, 2025, pp. 2679–2713. [Online]. Available: https://proceedings.mlr.press/v270/kim25c.html

[9] K. Black *et al.*, "π_0: A vision-language-action flow model for general robot control," in *Proceedings of Robotics: Science and Systems XXI*, 2025, doi: 10.15607/RSS.2025.XXI.010.

[10] C. S. Timperley *et al.*, "ROBUST: 221 bugs in the Robot Operating System," *Empirical Software Engineering*, vol. 29, no. 3, Art. no. 57, 2024, doi: 10.1007/s10664-024-10440-0.

[11] P. Canelas, B. Schmerl, A. Fonseca, and C. S. Timperley, "Understanding misconfigurations in ROS: An empirical study and current approaches," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 1161–1173, doi: 10.1145/3650212.3680350.

[12] S. García *et al.*, "Software variability in service robotics," *Empirical Software Engineering*, vol. 28, Art. no. 24, 2023, doi: 10.1007/s10664-022-10231-5.

[13] L. Zha *et al.*, "Distilling and retrieving generalizable knowledge for robot manipulation via language corrections," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2024, pp. 15172–15179, doi: 10.1109/ICRA57147.2024.10610455.

[14] Y. Dai, J. Lee, N. Fazeli, and J. Chai, "RACER: Rich language-guided failure recovery policies for imitation learning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2025. [Online]. Available: https://arxiv.org/abs/2409.14674

[15] R. Lu *et al.*, "ASPIRE: Agentic /Skills discovery for robotics," *arXiv preprint arXiv:2607.00272*, 2026. [Online]. Available: https://arxiv.org/abs/2607.00272

[16] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, Art. no. eabm6074, 2022, doi: 10.1126/scirobotics.abm6074.

[17] C. S. Timperley, T. Dürschmid, B. Schmerl, D. Garlan, and C. Le Goues, "ROSDiscover: Statically detecting run-time architecture misconfigurations in robotics systems," in *Proceedings of the IEEE International Conference on Software Architecture*, 2022, pp. 112–123, doi: 10.1109/ICSA53651.2022.00019.

[18] J. Yang *et al.*, "SWE-agent: Agent-computer interfaces enable automated software engineering," in *Advances in Neural Information Processing Systems*, vol. 37, 2024. [Online]. Available: https://arxiv.org/abs/2405.15793

[19] R. Royce *et al.*, "Enabling novel mission operations and interactions with ROSA: The Robot Operating System Agent," in *Proceedings of the IEEE Aerospace Conference*, 2025. [Online]. Available: https://arxiv.org/abs/2410.06472

[20] Y. Ren, F. Deichsel, J. Seiler, A. Kaup, and P. Beckerle, "Enhancing LLM inference with human expert knowledge: A case study on multi-agent robotics fault diagnosis and prediction," *Artificial Life and Robotics*, 2026, doi: 10.1007/s10015-026-01136-3.

[21] Anthropic, "Claude Code overview," *Claude Code Documentation*. Accessed: Sep. 15, 2026. [Online]. Available: https://code.claude.com/docs/en/overview

[22] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[23] C. Urrea and J. Kern, "Recent advances and challenges in industrial robotics: A systematic review of technological trends and emerging applications," *Processes*, vol. 13, no. 3, Art. no. 832, 2025, doi: 10.3390/pr13030832.

[24] I. Leal *et al.*, "SARA-RT: Scaling up robotics transformers with self-adaptive robust attention," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2024, doi: 10.1109/ICRA57147.2024.10611597.

[25] K. Chen *et al.*, "Robo-DM: Data management for large robot datasets," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2025. [Online]. Available: https://arxiv.org/abs/2505.15558

[26] S. Mirchandani *et al.*, "RoboCrowd: Scaling robot data collection through crowdsourcing," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2025. [Online]. Available: https://arxiv.org/abs/2411.01915

[27] M. Shirasaka *et al.*, "Self-recovery prompting: Promptable general purpose service robot system with foundation models and self-recovery," *arXiv preprint arXiv:2309.14425*, 2023. [Online]. Available: https://arxiv.org/abs/2309.14425

[28] F. Joublin *et al.*, "CoPAL: Corrective planning of robot actions with large language models," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2024, doi: 10.1109/ICRA57147.2024.10610434.

[29] H. Chen, Y. Yao, R. Liu, C. Liu, and J. Ichnowski, "Automating robot failure recovery using vision-language models with optimized prompts," *arXiv preprint arXiv:2409.03966*, 2024. [Online]. Available: https://arxiv.org/abs/2409.03966

[30] D. Driess *et al.*, "PaLM-E: An embodied multimodal language model," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 202, 2023, pp. 8469–8488.

[31] N. Yokoyama, S. Ha, D. Batra, J. Wang, and B. Bucher, "VLFM: Vision-language frontier maps for zero-shot semantic navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2024. [Online]. Available: https://arxiv.org/abs/2312.03275

[32] Y. Wu *et al.*, "SELP: Generating safe and efficient task plans for robot agents with large language models," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2025. [Online]. Available: https://arxiv.org/abs/2409.19471

[33] C. E. Mower *et al.*, "ROS-LLM: A ROS framework for embodied AI with task feedback and structured reasoning," *arXiv preprint arXiv:2406.19741*, 2024. [Online]. Available: https://arxiv.org/abs/2406.19741

[34] C. E. Jimenez *et al.*, "SWE-bench: Can language models resolve real-world GitHub issues?" in *Proceedings of the 12th International Conference on Learning Representations*, 2024. [Online]. Available: https://arxiv.org/abs/2310.06770

[35] Q. Wu *et al.*, "AutoGen: Enabling next-gen LLM applications via multi-agent conversations," in *Proceedings of the First Conference on Language Modeling*, 2024.

[36] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.