EvoMisty: Building, Optimizing, and Evolving Capabilities on a Social Robot

Xiao Wang Lu Dong Ifeoma Nwogu Srirangaraj Setlur Venu Govindaraju
State University of New York at Buffalo
{xwang277, ludong, inwogu, setlur, govind}@buffalo.edu

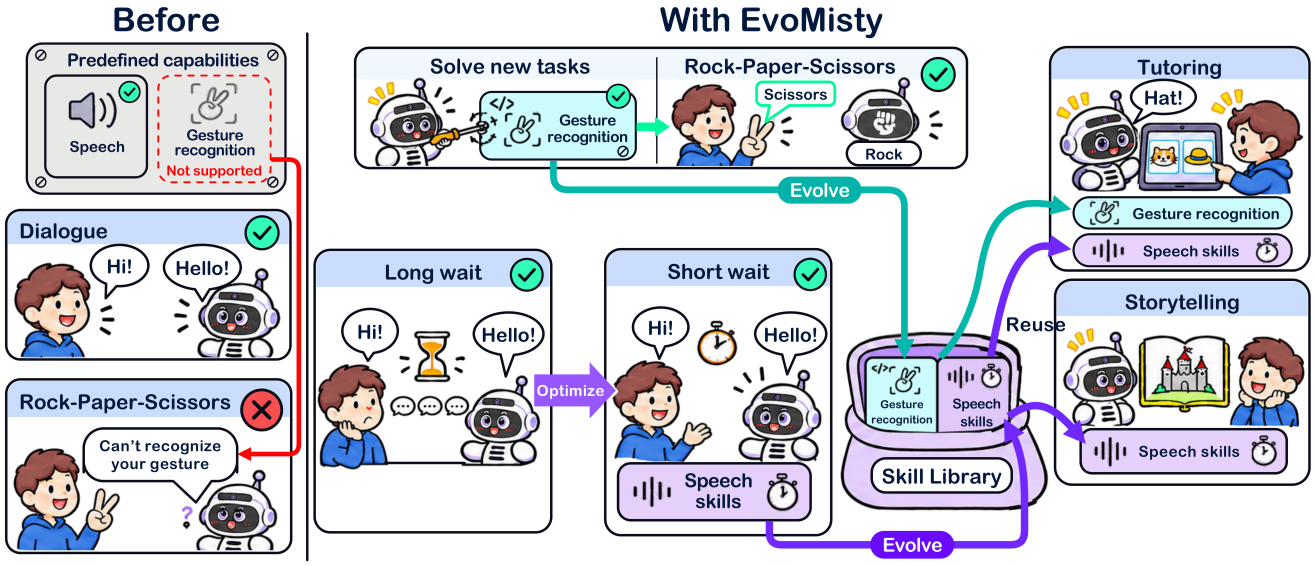

Figure 1. **Before:** Misty's capabilities are predefined, so tasks that need a missing capability cannot be completed. **With EvoMisty:** The agent autonomously builds required capabilities, optimizes its solutions on user-specified metrics, and distills these solutions into reusable skills.

Abstract

Social robots face diverse and changing tasks in companionship and education. LLM-based methods can generate task programs, but they still rely on manually developed control interfaces and predefined capabilities. They also lack a general mechanism to optimize completed tasks based on user-specified metrics and reuse performance improvements across tasks. We present EvoMisty, a plugin-based framework that enables coding agents to build and continually improve robot capabilities. EvoMisty builds a control program from official documentation and verifies candidate capabilities on a physical robot. Verified capabilities are stored in an atomic capability library and combined with existing skills to solve tasks. After task completion, EvoMisty can collect execution data, measure user-specified metrics, and iteratively improve task programs. It then distills reusable skills from final solutions into a skill library for future tasks. We evaluate EvoMisty on a physical Misty II robot using Claude Code and Codex. The two agents construct 109 and 107 verified atomic capabilities, respectively. Both complete all 28 AutoMisty benchmark tasks and seven additional tasks beyond AutoMisty's capabilities. Optimization improves all seven evaluated metrics across five tasks. For example, dialogue response latency decreases by 52% and 71%, respectively. Reusing skills from one optimized dialogue task reduces latency by 33% to 72% in three other dialogue tasks without further optimization.

1. Introduction

Social robots face diverse and changing tasks in companionship and education. New tasks may require additional device functions, external hardware, or better execution performance. LLM-based methods can generate task programs, but they rely on predefined device capabilities and lack a general mechanism to improve performance.

Existing methods, such as Code as Policies [20], AutoMisty [35], and MistyPilot [36], use predefined inter-

faces or skills to complete tasks. Recent studies, including ASPIRE [23] and RoboFoundry [21], further explore program improvement through execution feedback. However, two challenges remain. First, constructing and verifying basic device capabilities still largely requires manual development. Second, existing approaches mainly focus on task completion and failure correction. They lack a general mechanism to optimize completed tasks based on user-specified performance metrics and transfer these improvements to future tasks.

We introduce EvoMisty, a plugin-based framework that enables coding agents to autonomously build and continually improve robot capabilities. As shown in Fig. 2, EvoMisty consists of four stages. **Build** creates a control program from official documentation and stores verified capabilities in an atomic capability library. **Solve** combines atomic capabilities and existing skills to write and test task programs. **Optimize** collects execution data, measures user-specified metrics, and iteratively improves completed programs. **Evolve** distills reusable skills from final solutions into a skill library for future tasks.

We evaluate EvoMisty on a physical Misty II robot using Claude Code and Codex. The two agents construct 109 and 107 verified atomic capabilities, respectively. Both complete all 28 AutoMisty benchmark tasks and seven additional tasks beyond AutoMisty's capabilities. Optimization improves all seven evaluated metrics across five tasks. Reusing skills from one optimized dialogue task reduces latency by 33% to 72% in three other dialogue tasks without further optimization.

Our main contributions are as follows:

- We propose EvoMisty, a plugin-based framework that enables coding agents to autonomously construct and verify robot capabilities from official documentation.
- We introduce a metric-driven optimization workflow that improves completed tasks and transfers performance improvements through reusable skills.
- We demonstrate the effectiveness of EvoMisty on a physical social robot using two coding agents, 35 tasks, seven performance metrics, and cross-task reuse experiments.

2. Related Work

2.1. LLM-Based Task Solving for Social Robots

LLM-based methods use program generation and skill orchestration to complete natural-language robot tasks [19, 30]. Code as Policies [20] generates executable programs using predefined perception and control interfaces. Recent coding agents extend this formulation with multi-turn execution feedback [2, 10, 22, 43]. CaP-X [9] benchmarks such agents on manipulation and finds that success degrades once human-designed abstractions are removed. FAEA [34] applies an unmodified software-engineering agent to simulated manipulation, where the agent discovers the simulator APIs by reading their documentation. URAI [13] pairs an agent that writes reusable robot tools with an agent that calls them during execution. AutoMisty [35] uses multiple agents to generate and debug robot programs through execution and human feedback. MistyPilot [36] dynamically selects and orchestrates existing skills for sensor-triggered actions and multi-turn interactions. LLMs are also reshaping human-robot interaction more broadly [29, 39]. On Pepper, a function-calling framework lets the LLM plan over a predefined tool library [33]. RoboCritics [17] checks LLM-generated robot programs with expert-informed critics over execution traces, and Robo-Blocks [31] scaffolds novice programming of social robots. SkillComposer [41] compresses successful programs into reusable macro skills in simulation. These methods improve task flexibility but largely depend on predefined device interfaces or skills. In contrast, EvoMisty builds a control program from official documentation, verifies candidate capabilities on a physical robot, and stores verified capabilities in an atomic capability library for future tasks.

2.2. Agentic Self-Improvement for Robotic Systems

Recent studies explore autonomous robot improvement through execution feedback [4, 8, 15, 24, 38, 47]. More generally, self-evolving agents update their memory, skills, or tools from past interactions instead of staying fixed after deployment [11, 45, 46]. ASPIRE [23] uses multimodal execution traces to detect failures, repair programs, and accumulate reusable skills. RoboFoundry [21] extends improvement to supporting agent systems, including context management and skill organization. ENPIRE [42] establishes a physical robot experimentation loop through automated resets, execution, and outcome verification. RATs [44] builds a code skill library through self-directed play before downstream tasks arrive. COAP [16] has coding agents develop a shared library of stateful control code in a closed loop. Kuroki et al. [18] archive closed-loop programs improved with simulation feedback and reuse them when acquiring new tasks. SEAR [37] evaluates such systems with budget-matched restarts and a no-library baseline to separate learning from resampling. These methods demonstrate the potential of coding agents for autonomous improvement and skill accumulation. However, they mainly focus on failure repair, policy improvement, or system evolution. EvoMisty further focuses on performance optimization after task completion. It also distills reusable functions from final solutions into a skill library, allowing future tasks to benefit from previous improvements.

2.3. Agentic Code Performance Optimization

Beyond functional correctness, coding agents have also been used for code performance optimization [5, 6, 12,

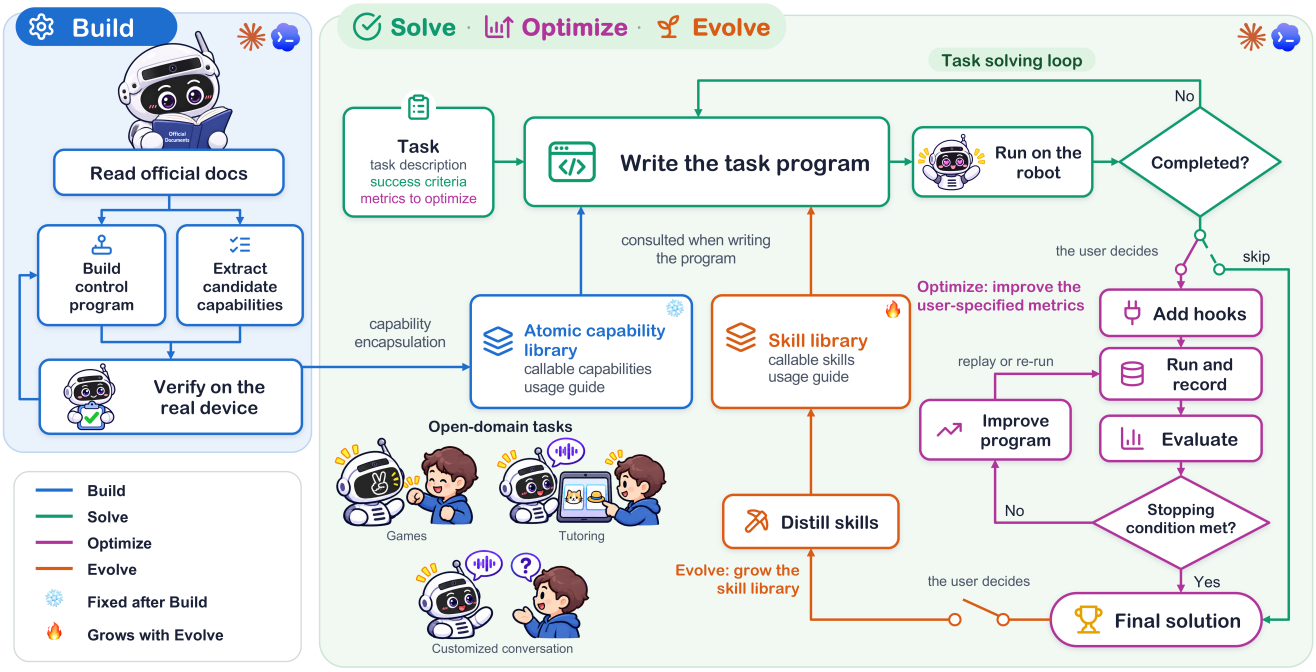

Figure 2. Overview of EvoMisty, consisting of four stages: **Build**, **Solve**, **Optimize**, and **Evolve**.

14, 25–27, 32]. Peng et al. [28] found that agent-authored performance optimization pull requests report explicit performance validation less often than human-authored ones. PerfAgent [7] uses profiler feedback and test results to identify performance bottlenecks and guide iterative code optimization. However, robot task performance involves more than code execution efficiency. Metrics such as recognition accuracy, interaction latency [1, 33], and motion time require task-level measurements. These metrics cannot be fully evaluated by software profilers alone. EvoMisty uses data hooks and measurement hooks to collect execution data and performance evidence. It combines offline replay and physical robot evaluation to iteratively optimize programs based on user-specified metrics.

3. Method

As shown in Fig. 2, EvoMisty separates device-level capability construction from task-level execution. **Build** runs once per device and produces a fixed Atomic Capability Library shared across tasks. Each task is first completed by **Solve**; the user then decides whether to run **Optimize**, which improves the solution on user-specified metrics, and **Evolve**, which distills the final solution into the Skill Library.

3.1. Build: Atomic Capability Library

In Build, EvoMisty guides the coding agent to read official device documentation, identify candidate capabilities, and construct a control program for verification and subsequent use. Since documented functions may behave differently on physical hardware, only capabilities verified on the robot are added to the Atomic Capability Library.

Physical Verification. For each candidate, the agent designs a test with specified inputs, expected outcomes, and pass conditions. It runs the test on the physical robot and evaluates the results using device states, execution logs, and other objective evidence when possible; otherwise, it requests human feedback. When a test fails, the agent iteratively repairs the control program and revalidates the affected capabilities; those that still fail after a fixed number of attempts are discarded.

Capability Encapsulation. The agent determines the actual functions and parameters of verified capabilities and encapsulates them as independently callable modules with usage guides. These modules form the Atomic Capability Library. New hardware can be integrated through the same process, without manually prepared device primitives.

3.2. Solve: Task Completion

Solve generates a task program using the Atomic Capability Library and the Skill Library, and verifies task completion on the physical robot. For each task, the agent divides the success criteria into verifiable conditions, which are confirmed by the user. Conditions based on execution logs, de-

vice responses, or events are checked by agent-generated programs. Conditions that require observing or listening to the robot are checked by humans.

The agent consults both libraries and prioritizes existing skills when writing the task program. After each run, EvoMisty first checks for execution errors and then runs the automatic checks. Human checks are performed only after all automatic checks pass. If any check fails, the agent revises the program based on the feedback and runs it again. A task is complete only when the program follows the required execution flow and satisfies all success criteria. Further performance improvements are handled separately by Optimize.

3.3. Optimize: Metric-Guided Optimization

Solve ensures that a task executes successfully, but completing a task does not guarantee that it is performed well. If the user chooses to run Optimize, it improves the completed program with respect to user-specified metrics while preserving its original success criteria. Unlike conventional code optimization, the metrics that matter for robot tasks, such as recognition accuracy, interaction latency, and motion time, cannot be measured by software profilers alone. EvoMisty instead measures and improves performance using evidence collected from task execution.

Data Collection and Measurement. Based on the target metrics, EvoMisty guides the agent to instrument the program with two types of hooks. *Data hooks* record raw inputs, such as audio and video, together with program outputs and robot feedback, which support later analysis and replay. *Measurement hooks* record objective evidence, such as event timestamps and execution results. Performance metrics are computed from this evidence rather than estimated by the agent.

EvoMisty first establishes a performance baseline by executing the program on the physical robot. For components that can be evaluated offline, it replays the recorded data so that program changes are compared under identical inputs. Closed-loop tasks, such as line following, instead require complete physical runs, because the robot's actions affect its subsequent observations; in these cases, replay is used only to analyze problems.

Iterative Optimization and Selection. In each iteration of the optimization loop, the agent identifies a performance bottleneck from the measurements and makes a single change to the program. The resulting candidate is evaluated on the target metrics and checked against the original success criteria.For valid candidates, EvoMisty maintains a Pareto frontier of non-dominated solutions based on the measured metrics. Optimization stops when the predefined budget is exhausted or when no further improvement is observed over several consecutive iterations.

3.4. Evolve: Skill Distillation

Evolve distills reusable functions from a final task solution into the Skill Library, so that future tasks can build on them instead of starting from the atomic capabilities alone. When the solution has been optimized, its performance improvements carry over to future tasks through the distilled skills. The Skill Library is updated only when the user chooses to run Evolve.

Distillation proceeds in three steps. First, EvoMisty guides the agent to *extract* candidate skills: the agent identifies independently callable functions in the final program, removes their task-specific dependencies, and encapsulates each one as a candidate. Second, each candidate is *verified*: EvoMisty checks that it remains consistent with the final program and confirms that it is callable through simulated dependency checks. Third, each verified candidate is *compared* with the existing skills in the library. A candidate that provides new functionality is added as a new skill; one that improves on an existing skill is saved as a new version, with previous versions preserved; and a duplicate is discarded. Accepted skills are stored with interface descriptions and usage guides, which support their reuse and composition in future tasks.

4. Experiments

We evaluate EvoMisty through four research questions:

- **RQ1: Capability Construction.** *Given only the official documentation, can the agent autonomously build device capabilities verified on the physical robot?*
- **RQ2: Task Completion.** *Can EvoMisty complete the AutoMisty benchmark tasks, as well as tasks that AutoMisty cannot complete?*
- **RQ3: Metric-Guided Optimization.** *Can EvoMisty improve a completed program on user-specified metrics?*
- **RQ4: Skill Reuse.** *Can skills distilled from one task be reused by, and benefit, other tasks?*

Experimental Setup. All experiments use a Misty II social robot. To test whether the results depend on a particular agent, we run EvoMisty on two coding agents, Claude Code (Opus 5.5) and Codex (GPT-6 Astra), both with reasoning effort set to `high`. Our baseline is AutoMisty, a carefully hand-designed framework for Misty. Since the original AutoMisty uses GPT-4o, we replace its backbone with Opus 5.5 and rerun all tasks to rule out backbone differences. Every run starts in a fresh container with no memory and no access to previous trajectories; only the Atomic Capability Library and the Skill Library persist across runs.

Table 1. Capability construction results.

Method	Extracted	Excluded	Tested	Failed	Verified
AutoMisty [35]	–	–	–	–	12
Claude Code	175	63	112	3	109
Codex	175	63	112	5	107

4.1. RQ1: Capability Construction

EvoMisty takes only the official Misty documentation as input, autonomously builds device capabilities, and verifies each of them on the physical robot. Throughout the process, a human only answers binary questions when the system asks, such as "Did the robot move forward?" or "Did it speak?". As shown in Table 1, EvoMisty extracts 175 candidate capabilities from the documentation and autonomously excludes 63 of them from testing because the robot lacks the required hardware; this decision agrees with our manual review. The remaining 112 are tested on the robot, of which Claude Code verifies 109 and Codex verifies 107. Our manual review confirms that 110 of the 112 are actually usable, so the two agents cover 109 and 107 of them, respectively.

The failures fall into three categories. First, *overly strict judgment*: onboard speech recognition works in practice, but its return values differ from the documentation, and both agents therefore mark it as failed. Second, *no observable verification condition*: for clearing on-screen error messages and for hazard detection events, the system cannot construct a triggering condition, so neither agent can verify them. Third, *unstable capabilities*: driving by left and right track speeds and world-state events fail only with Codex; our manual review finds that both behave intermittently and happened to malfunction during the Codex run. Overall, verification is conservative: no unusable capability is accepted as usable.

EvoMisty also achieves substantially broader coverage of the robot's native capabilities than AutoMisty's hand-designed framework. AutoMisty [35] directly uses 12 native capabilities, on top of which its authors manually wrapped 136 capabilities for motions and expressions. As Table 1 shows, both agents verify all 12 of these, and EvoMisty covers about nine times as many native capabilities. The same process extends to external hardware: from the device documentation, EvoMisty builds control code for a gamepad, a standalone keypad, and an external camera, and confirms on the physical robot that each works.

In summary, given only the official documentation and a small amount of binary feedback, EvoMisty autonomously builds a set of device capabilities that is broader than the hand-designed one and extends to new hardware, with largely consistent results across the two agents.

4.2. RQ2: Task Completion

We evaluate on two sets of tasks. The first is the full AutoMisty [35] benchmark of 28 tasks, which span seven categories: body motion, expression and speech output, multimodal performance, touch interaction, voice commands and question answering, visual understanding, and spoken dialogue. The second is a set of 7 new tasks that test whether a system can go beyond what a hand-designed framework anticipates. They are designed around four kinds of requirements that the AutoMisty framework does not support: native capabilities that it does not wrap, external hardware, external resources, and perception algorithms that the robot does not natively provide. The new tasks fall into two groups. Two are small extensions of benchmark tasks: Original-Voice Pikachu extends AutoMisty's Pikachu task by retrieving an original Pikachu voice clip from the web and playing it while displaying a Pikachu image. Face-Tracking Recording extends AutoMisty's recording task by rotating the robot's base to follow a detected face during recording. The other five are substantially harder than any benchmark task. MistyPilot Emotional Dialogue ports the emotional expression mechanism of MistyPilot [36] from its source code and uses the keys of an external keypad to control the dialogue. Gamepad Teleoperation integrates a gamepad to teleoperate the robot like a remote-controlled car. Rock-Paper-Scissors (Fig. 3c) builds its own gesture recognition: it recognizes the user's gesture, displays the robot's move on its screen, and determines the outcome. External-Camera Line Following (Fig. 3b) localizes the robot with a network camera: it detects the track and the robot's position in the external camera view and uses them for closed-loop steering correction over one lap of a loop track. Minimal Pair Therapy (Fig. 3a) is adapted from a clinical speech-language intervention for children with speech sound disorders [3, 40]: the robot first generates the teaching script, then uses vision to locate the screen and find a suitable viewing angle, judges which card the child points at and which word of a pair the child says, and finally coordinates multimodal feedback.For each task, we evaluate completion at the workflow level. A task is considered complete when its program executes the required end-to-end workflow on the physical robot, including the necessary perception, decision-making, and action stages. Completion does not require meeting quantitative performance thresholds, such as recognition accuracy, response latency, or execution time, which are evaluated separately in RQ3. EvoMisty and AutoMisty are evaluated using the same completion criteria for each task.

On the 28 benchmark tasks, EvoMisty completes all of them with both agents, matching AutoMisty. AutoMisty relies on 136 manually wrapped capabilities and a hand-designed execution framework, whereas EvoMisty uses only the device capabilities it builds autonomously, which

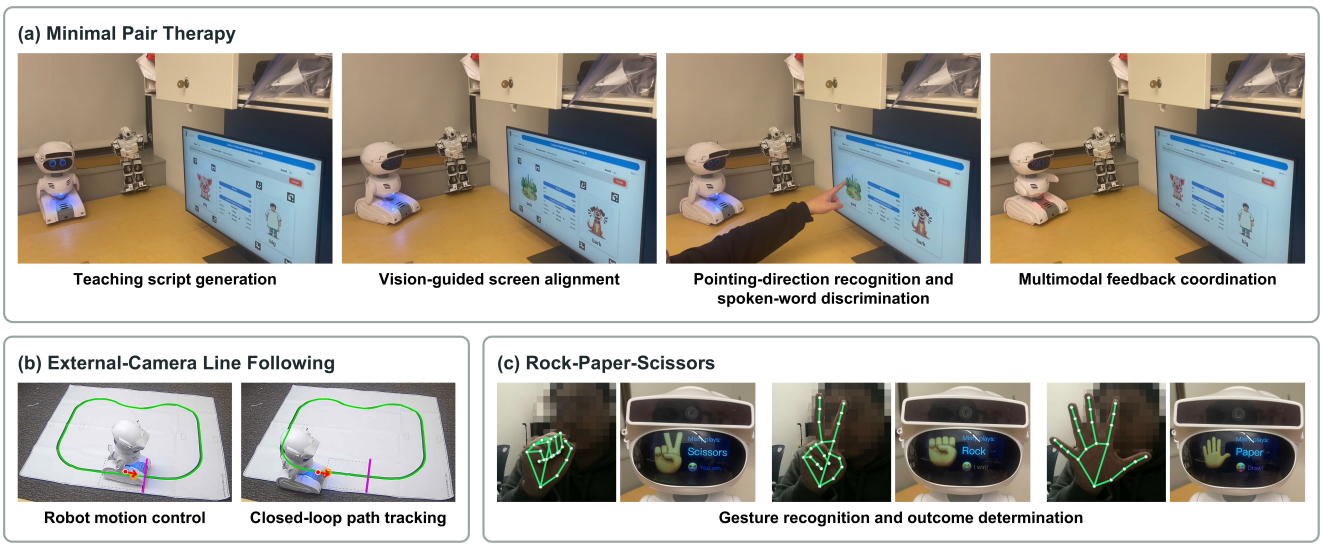

Figure 3. Representative new tasks. Each panel shows the main steps of one task as executed on the Misty robot.

Table 2. Optimization results across tasks and metrics.

Task	Metric	Claude Code		Codex	
		Before	After	Before	After
Naughty-Child Dialogue	Response latency (s) ↓	7.41 ± 0.70	3.54 ± 0.47	7.18 ± 1.03	2.06 ± 0.25
	CER ↓	0.053	0.000	0.037	0.000
MistyPilot Emotional Dialogue	Response latency (s) ↓	5.59 ± 1.63	4.51 ± 0.56	6.62 ± 0.60	4.47 ± 0.46
Rock-Paper-Scissors	Gesture recognition accuracy ↑	0.845	0.987	0.273	0.874
External-Camera Line Following	Lap time (s) ↓	32.0	20.3 ± 6.2	733	339 ± 11
Minimal Pair Therapy	Pointing-direction accuracy ↑	0.833	0.917	0.667	0.750
	Spoken-word discrimination accuracy ↑	0.667	0.917	0.500	0.917

Time metrics are reported as mean ± standard deviation. ↓: lower is better; ↑: higher is better.

shows that these capabilities can replace manual wrapping and design.

The difference appears on the 7 new tasks: EvoMisty completes all of them with both agents, while AutoMisty,Even after upgrading AutoMisty's backbone from GPT-4o to the more capable Opus 5.5, it cannot complete any of the seven new tasks. This suggests that its limitations stem from its fixed framework rather than simply from the capability of the backbone model. The reason is that its framework wraps only 12 native capabilities; it can neither drive the base nor upload files, and it has no way to connect external devices. Even the two extension tasks are out of reach, the former for lack of an capabilities to upload audio and images and the latter for lack of an capabilities to drive the base. The limitation therefore comes from the fixed framework, not from the model.

4.3. RQ3: Metric-Guided Optimization

We evaluate Optimize on 7 metrics across 5 tasks (Table 2): one benchmark task and four new tasks, covering speech interaction, visual perception, and motion control. We select tasks that have a quantifiable performance metric beyond binary success.Among the four benchmark dialogue tasks, which share the same pipeline, we optimize only Naughty-Child Dialogue, in which the robot plays a naughty child and engages in a multi-turn conversation with a person, and reserve the other three tasks for the skill reuse experiment in Sec. 4.4.1. For each task, the user specifies the metrics in natural language. All tasks use the same optimization procedure, with no design specific to any task or metric.

Each metric corresponds to an optimization target that the user specifies in the task definition, and all measurements follow three principles. First, the values in Table 2 are measured independently by us. Second, the test data is never used during optimization. Third, the programs before and after optimization are measured with the same protocol.

The metrics are defined as follows:

- **Response latency**: the time from the end of the user's utterance to the moment the robot starts speaking, measured

Table 3. Skill reuse results on the four dialogue tasks.

Task	Metric	AutoMisty	Claude Code		Codex	
			Before	After	Before	After
Naughty-Child Dialogue (optimized)	Response latency (s)	4.70 ± 2.24	7.41 ± 0.70	3.54 ± 0.47	7.18 ± 1.03	2.06 ± 0.25
	CER	0.077	0.053	0.000	0.037	0.000
Math Tutoring (skill reuse)	Response latency (s)	4.54 ± 1.10	7.29 ± 1.17	4.26 ± 0.33	9.49 ± 2.14	3.12 ± 0.38
	CER	0.306	0.061	0.019	0.024	0.019
Keyword Story (skill reuse)	Response latency (s)	6.10 ± 3.02	16.58 ± 3.47	4.63 ± 0.69	10.92 ± 1.11	6.61 ± 0.50
	CER	0.433	0.200	0.117	0.400	0.000
Chinese-to-English Translation (skill reuse)	Response latency (s)	3.37 ± 0.37	5.86 ± 0.76	2.97 ± 0.36	5.40 ± 1.09	3.61 ± 0.30
	CER	0.059	0.032	0.018	0.136	0.017

Optimized: Before/After denote before/after optimization. *Skill reuse*: Before/After denote without/with distilled skills, without further optimization. Latency is reported in seconds (mean $\pm$ std).

from audio recordings over 10 turns for each of the two dialogue tasks.

- **Character error rate (CER)**: compares the system's transcription of the user's speech with a human-annotated reference, character by character.
- **Gesture recognition accuracy**: computed per frame on 10 videos with 146 frames in total, as the average of the accuracies for rock, paper, and scissors; frames in which no hand is detected or no decision is made count as errors.
- **Lap time**: the time from when the robot crosses the start line until it crosses it again after one full lap along the track, timed manually, with no shortcuts or departures from the track allowed; we time 1 lap before optimization and 3 laps after.
- **Pointing-direction accuracy**: whether the system correctly decides if the user points to the left or the right card, over 12 trials with manually recorded ground truth.
- **Spoken-word discrimination accuracy**: whether the system correctly decides which word of the pair the user says, over 12 trials with manually recorded ground truth.

With both agents, all 7 metrics improve. Among the time metrics, the response latency of Naughty-Child Dialogue drops by 52% and 71% (Claude Code and Codex, respectively; the same order is used below), that of MistyPilot Emotional Dialogue by 19% and 32%, and the lap time of line following by 37% and 54%. Among the recognition metrics, gesture recognition accuracy rises from 0.845 and 0.273 to 0.987 and 0.874, which reduces the error rate by 92% and 83%; CER falls to 0 for both agents; and for Minimal Pair Therapy, spoken-word discrimination accuracy rises from 0.667 and 0.500 to 0.917 for both agents, while pointing-direction accuracy also improves (from 0.833 to 0.917 and from 0.667 to 0.750).

The initial programs differ widely in performance because the two agents choose different implementations: the initial lap times are 32.0 s and 733 s, and the initial gesture recognition accuracies are 0.845 and 0.273. All of these initial programs already satisfy the success criteria of their tasks and differ only in performance. Optimization improves the program regardless of where it starts, but the level it reaches still depends on the initial implementation: the line-following program that starts at 733 s reaches 339 s, whereas the one that starts at 32.0 s reaches 20.3 s.

In summary, the answer to RQ3 is yes: the same procedure improves all 7 user-specified metrics across 5 tasks with both agents. Optimize delivers improvement relative to the starting point, and the final level also depends on the implementation that the agent initially chooses.

4.4. RQ4: Skill Reuse

We examine Evolve in two settings:

- **Similar tasks.** *Can the optimization gains of one task transfer to other tasks through skills and bring measurable benefits?*
- **Different tasks.** *Can the skills distilled from several tasks be composed in a new task?*

4.4.1. Skill Reuse Across Similar Tasks

The AutoMisty benchmark contains four dialogue tasks that follow the same pipeline: recognize the user's speech, generate a reply, synthesize speech, and play it on the robot. They differ only in how the reply is generated. We run Optimize **only** on Naughty-Child Dialogue, targeting response latency and CER, and then run Evolve to distill its final program into skills. For each of the other three tasks (Math Tutoring, Keyword Story, and Chinese-to-English Translation), the agent solves the task twice: once with an **empty Skill Library** and once **with these skills**. Both runs start in a fresh container and go through Solve only, without Optimize, so the only difference is whether the skills are available.

With the skills, both metrics improve on all three tasks with both agents, in all 12 comparisons (Table 3). Response latency drops by 42% and 67% on Math Tutoring (Claude

Code and Codex, respectively; the same order is used below), by 72% and 39% on Keyword Story, and by 49% and 33% on Chinese-to-English Translation, an average reduction of 50% over the six comparisons. CER drops by 69% and 21% on Math Tutoring, by 42% and 100% on Keyword Story, and by 44% and 88% on Chinese-to-English Translation, an average reduction of 60%. Because the two runs differ only in whether the skills are available, these improvements can be attributed to skill reuse; none of the three tasks is optimized itself.

The dialogue pipeline of AutoMisty was manually tuned for latency. Without skills, both agents are slower than AutoMisty on all four tasks. After optimization or skill reuse, Claude Code is faster than AutoMisty on all four tasks, and Codex is faster on two of them. CER is already lower than that of AutoMisty in most cases beforehand, and it is lower on all four tasks afterwards.

These results demonstrate that making distilled skills available improves task performance without further optimization.

4.4.2. Skill Composition Across Different Tasks

We let both agents complete, in sequence, the five harder new tasks introduced in Sec.4.2. MistyPilot Emotional Dialogue, Gamepad Teleoperation, Rock-Paper-Scissors, External-Camera Line Following, and Minimal Pair Therapy. Each task runs in a fresh container, and only the Skill Library carries over: after each of the first four tasks, EvoMisty distills its reusable functions into skills, so that when the last task starts, the Skill Libraries of Claude Code and Codex contain 21 and 18 skills, respectively. We then inspect every skill call in the final program of Minimal Pair Therapy and trace each skill back to the task it was distilled from. This task is a demanding test of composition, as it simultaneously requires image capture, pointing-direction recognition, emotional speech, spoken-word discrimination, and rotating the base to face the screen.

Most of the capabilities that Minimal Pair Therapy requires are already available to both agents from earlier tasks. The Claude Code program draws on all four preceding tasks: to capture the cards and the pointing finger, it uses the camera video stream from Gamepad Teleoperation; to speak each line with its specified emotion, it uses speech style setting from MistyPilot Emotional Dialogue; to accompany speech with chest-light and arm motions, it uses emotional motion from Rock-Paper-Scissors; and to track the heading while the base rotates, it uses heading readout from External-Camera Line Following. The Codex program draws on two of them: to speak with emotion, it reuses emotional speech playback from MistyPilot Emotional Dialogue; to turn the base toward the screen, it reuses heading-feedback driving from External-Camera Line Following; and to tell which word of the pair the user says, it rewrites the speech recognition skill from MistyPilot Emotional Dialogue. What both agents still write from scratch is only the logic specific to this task, such as card recognition, pointing-direction recognition, and the teaching protocol.

In summary, the answer to RQ4 is yes. The Skill Library stores independently callable modules rather than complete task programs. These modules carry the optimization gains of one task over to similar tasks, and they can be reused directly or rewritten and composed with new task logic in different tasks.

5. Limitations

Our experiments focus on the Misty II social robot. Although we evaluate two coding agents and integrate several external devices, the applicability of EvoMisty to other robot platforms remains to be validated. Moreover, capability verification and task completion checks still rely on human observation and feedback when device states, execution logs, or events do not provide sufficient evidence. Verification coverage is also constrained by the test conditions that can be constructed. Finally, the unified optimization workflow can improve task performance, but the final performance still depends on the initial implementation chosen by the coding agent, and substantial differences between implementations may remain after optimization.

6. Conclusion

Existing LLM-based methods for social robots mostly rely on hand-built capabilities and stop once a task is completed. EvoMisty removes both limitations: coding agents build the capabilities themselves and keep improving completed tasks on user-specified metrics. We validate three contributions on a physical Misty II with Claude Code and Codex. First, from the official documentation and a small amount of binary human feedback, the two agents build and verify 109 and 107 atomic capabilities, respectively, covering about nine times as many native APIs as the hand-designed AutoMisty. With these capabilities, EvoMisty completes all 28 benchmark tasks and 7 new tasks that AutoMisty cannot complete with the same backbone. Second, a single procedure, driven by evidence collected during execution and with no design specific to any task or metric, improves all 7 user-specified metrics across 5 tasks spanning speech, vision, and motion control. Third, skills distilled from one optimized dialogue task reduce response latency by 50% and CER by 60% on average in three other dialogue tasks that are never optimized. In Minimal Pair Therapy, the agents compose skills from up to four different tasks and write only the task-specific logic.

7. LLM Usage Statement

LLM-based coding agents are the subject of this work: Claude Code (Opus 5.5) and Codex (GPT-6 Astra) were used to generate, test, and optimize robot programs in our experiments. GPT-6 Astra was used to generate some of the robot illustrations in our figures. Claude Fable 5.2 and GPT-6 Astra were used to check grammar and improve the clarity of the writing. The authors reviewed and verified all LLM-assisted content and take full responsibility for the paper.

References

[1] Giulio Antonio Abbo, Maria Jose Pinto-Bernal, Martijn Catrycke, and Tony Belpaeme. Multi-party open-ended conversation with a social robot. *Frontiers in Robotics and AI*, 13:1766383, 2026. 3

[2] Sanghyun Ahn, Wonje Choi, Junyong Lee, Jinwoo Park, and Honguk Woo. Towards reliable code-as-policies: A neuro-symbolic framework for embodied task planning. *Advances in Neural Information Processing Systems*, 38:75428–75459, 2026. 2

[3] Jessica A Barlow and Judith A Gierut. Minimal pair approaches to phonological remediation. In *Seminars in speech and language*, pages 057–068. Copyright© 2002 by Thieme Medical Publishers, Inc., 333 Seventh Avenue, New ..., 2002. 5

[4] Kaiyuan Chen, Shuangyu Xie, Letian Fu, Justin Yu, William Pacini, Sandeep Bajamahal, Hudson Kim, Jaimyn Drake, Daehwa Kim, Haoru Xue, et al. Gap: A graph-as-policy multi-agent self-learning harness for variational automation tasks. *arXiv preprint arXiv:2607.05369*, 2026. 2

[5] Zhi Chen, Zhensu Sun, Yuling Shi, David Lo, and Lingxiao Jiang. Are performance-optimization benchmarks reliably measuring coding agents? *arXiv preprint arXiv:2607.01211*, 2026. 2

[6] Yingyun Cui, Yi Xie, Piaohong Wang, Jiawei Ma, Bo Liu, and Liangliang Cao. Perfopt-bench: Evaluating coding agents on software performance optimization. *arXiv preprint arXiv:2607.07744*, 2026. 2

[7] Ryan Deng, Yuanzhe Liu, Bastian Lipka, Yao Ma, Xuhao Chen, Tim Kaler, and Jatin Ganhotra. Perfagent: Profiler-guided iterative refinement for repository-level code optimization. *arXiv preprint arXiv:2607.19653*, 2026. 3

[8] Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A Seshia, and Matei Zaharia. Rho: Your coding agent is secretly a roboticist. *arXiv preprint arXiv:2606.16458*, 2026. 2

[9] Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, et al. Cap-x: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026. 2

[10] Hongcheng Gao, Jingjing Zhou, Zelin Zheng, Shijia Ge, Jay Zhu, Yazhe Wang, Jianshu Zeng, Xuan Shangguan, Di Wu, Lingyu He, et al. Self-evolving coding agents: From digital programs to physical-world intelligence. *arXiv preprint arXiv:2609.35432*, 2026. 2

[11] Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025. 2

[12] Spandan Garg, Roshanak Zilouchian Moghaddam, and Neel Sundaresan. Perfbench: Can agents resolve real-world performance bugs? In *Proceedings of the 2026 International Workshop on Agentic Engineering*, pages 165–172, 2026. 2

[13] Shijia Ge, Alex Zhou, Jianshu Zeng, Yexing Wan, Di Wu, Zelin Zheng, Yazhe Wang, Zhiqi Jia, Xuan Shangguan, Jay Zhu, et al. Make code as policy great again: Frontier agents write, call, and evolve robot tools. *arXiv preprint arXiv:2609.39018*, 2026. 2

[14] Xinyi He, Qian Liu, Mingzhe Du, Lin Yan, Zhijie Fan, Yiming Huang, Yin Zheng, Zejian Yuan, and Zejun Ma. Sweperf: Can language models optimize code performance on real-world repositories? *arXiv preprint arXiv:2507.12415*, 2025. 3

[15] Jiaheng Hu, Roberto Martin-Martin, Peter Stone, Rocky Duan, Zhenyu Jiang, and Guanya Shi. Simex: Simulation-integrated robotics autoresearch. *arXiv preprint arXiv:2609.38982*, 2026. 2

[16] Kairui Hu, Siyuan Hu, Fangzhou Hong, Zhaoxi Chen, and Ziwei Liu. Embodied turing machines: Stateful code for robot recursive self-improvement. *arXiv preprint arXiv:2610.12369*, 2026. 2

[17] Callie Y Kim, Nathan Thomas White, Evan He, Frederic Sala, and Bilge Mutlu. Robocritics: Enabling reliable end-to-end llm robot programming through expert-informed critics. In *Proceedings of the 21st ACM/IEEE International Conference on Human-Robot Interaction*, pages 914–923, 2026. 2

[18] So Kuroki, Kento Kawaharazuka, and Yujin Tang. Learning and transferring closed-loop robot software. *arXiv preprint arXiv:2609.19906*, 2026. 2

[19] Ruiying Li, Yunlang Zhou, YuYao Zhu, Kylin Chen, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, Jiayao Ma, et al. Roboclaw: An agentic framework for scalable long-horizon robotic tasks. In *European conference on computer vision*, pages 304–320. Springer, 2026. 2

[20] 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. 1, 2

[21] Jingsong Liang, Shuhao Liao, Shizhe Zhang, Diyuan Hou, Yuxin Cai, Xinjian Deng, Chengyang He, Wenhui Huang, Runjia Tan, Zhidong Wang, et al. Robofoundry: System-as-policy evolution for self-learning embodied agents. *arXiv preprint arXiv:2609.32862*, 2026. 2

[22] Yuyao Liu, Jiayuan Mao, David Hsu, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Rapid: Robot agentic programming from demonstrations. *arXiv preprint arXiv:2609.30249*, 2026. 2

[23] Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, et al. Aspire: Agentic/skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026. 2

[24] Meng Luo, Zechen Bai, Shanqing Xu, Yanlin Li, Kaixin Li, Mengchen Qian, Weifeng Lu, Ran Zhang, Yingtian Zou, Ziyang Luo, et al. Toward recursive self-improvement in physical ai. *arXiv preprint arXiv:2610.29677*, 2026. 2

[25] Jeffrey Jian Ma, Milad Hashemi, Amir Yazdanbakhsh, Kevin Swersky, Ofir Press, Enhui Li, Vijay Janapa Reddi, and Parthasarathy Ranganathan. Swe-fficiency: Can language models optimize real-world repositories on real workloads? *arXiv preprint arXiv:2511.06090*, 2025. 3

[26] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[27] Md Nahidul Islam Opu, Shahidul Islam, Muhammad Asaduzzaman, and Shaiful Chowdhury. How do agentic ai systems address performance optimizations? a bertopic-based analysis of pull requests. In *Proceedings of the 23rd International Conference on Mining Software Repositories*, pages 1009–1013, 2026. 3

[28] Huiyun Peng, Antonio Zhong Qiu, Ricardo Andrés Calvo Méndez, Kelechi G Kalu, and James C Davis. How do agents perform code optimization? an empirical study. In *Proceedings of the 23rd International Conference on Mining Software Repositories*, pages 732–736, 2026. 3

[29] Maria Pinto-Bernal, Matthijs Biondina, and Tony Belpaeme. Designing social robots with llms for engaging human interaction. *Applied Sciences*, 15(11):6377, 2025. 2

[30] Emmanuel K Raptis, Athanasios Ch Kapoutsis, and Elias B Kosmatopoulos. Agentic llm-based robotic systems for real-world applications: a review on their agenticness and ethics. *Frontiers in Robotics and AI*, 12:1605405, 2025. 2

[31] Arissa J Sato, Callie Y Kim, Nathan Thomas White, Abhinav Maneesh, Yuqing Wang, Hui-Ru Ho, and Bilge Mutlu. Robo-blocks: Generative scaffolding in end-user design and programming of social robots. In *Proceedings of the 2026 Designing Interactive Systems Conference*, pages 3763–3779, 2026. 2

[32] Manish Shetty, Naman Jain, Jinjian Liu, Vijay Kethanaboyina, Koushik Sen, and Ion Stoica. Gso: Challenging software optimization tasks for evaluating swe-agents. *Advances in Neural Information Processing Systems*, 38, 2026. 3

[33] Erich Studerus, Vivienne Jia Zhong, and Stephan Vonschallen. A framework for low-latency, llm-driven multimodal interaction on the pepper robot. In *Proceedings of the 21st ACM/IEEE International Conference on Human-Robot Interaction*, pages 1298–1302, 2026. 2, 3

[34] Brian Y Tsui, Alan Y Fang, and Tiffany J Hwu. Demonstration-free robotic control via llm agents. *arXiv preprint arXiv:2601.20334*, 2026. 2

[35] Xiao Wang, Lu Dong, Sahana Rangasrinivasan, Ifeoma Nwogu, Srirangaraj Setlur, and Venugopal Govindaraju. Automisty: a multi-agent llm framework for automated code generation in the misty social robot. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9194–9201. IEEE, 2025. 1, 2, 5

[36] Xiao Wang, Lu Dong, Ifeoma Nwogu, Srirangaraj Setlur, and Venu Govindaraju. Mistypilot: Enabling social-robot control through multi-agent llm skill orchestration. *arXiv preprint arXiv:2608.15549*, 2026. 1, 2, 5

[37] Xinyi Wang, Bangzheng Li, Hanyu Wang, Lin Zhang, Andrea Yaoyun Cui, Zhenggang Tang, Shyamal Buch, and Dejia Xu. Sear: Self-evolving agentic robotics. *arXiv preprint arXiv:2610.18792*, 2026. 2

[38] Xin Wang, Wenhao Wu, Menghao Zhang, Zhi Wang, Kun Shao, Jian Luan, Yang Li, Qing Li, Shangding Gu, Huichi Zhou, et al. Harnesspai: An evolving harness for physical ai. *arXiv preprint arXiv:2609.29166*, 2026. 2

[39] Yufeng Wang, Yuan Xu, Anastasia Nikolova, Yuxuan Wang, Jianyu Wang, Chongyang Wang, and Xin Tong. How do we research human-robot interaction in the age of large language models? a systematic review. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, pages 1–28, 2026. 2

[40] Frederick F Weiner. Treatment of phonological disability using the method of meaningful minimal contrast: Two case studies. *Journal of Speech and Hearing Disorders*, 46(1): 97–103, 1981. 5

[41] John Woods and Hasti Seifi. Skillcomposer: Learning reusable skills for natural-language robot programming. *arXiv preprint arXiv:2608.14944*, 2026. 2

[42] Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, et al. Enpire: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026. 2

[43] Gina Yoon, Sumin Lee, and Joo Yong Sim. Moduloop: Low-level code generation using modular synthesizer and closed-loop debugger for robotic control. *IEEE Robotics and Automation Letters*, 2025. 2

[44] Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, Yaowei Liu, Raj Saravanan, Shaofeng Yin, Justin Yu, Dantong Niu, et al. Playful agentic robot learning. *arXiv preprint arXiv:2606.19419*, 2026. 2

[45] Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, pages 104223–104294, 2026. 2

[46] Hao Zhou, Haichuan Hu, Tianyu Luo, Ye Shang, Chunrong Fang, Zhenyu Chen, Liang Xiao, and Quanjun Zhang. Self-evolving coding agents. *arXiv preprint arXiv:2608.03392*, 2026. 2

[47] Hai Zhou, Fan Yang, Zeyi Chen, Lejun Zhang, Chenyang Fu, Jianping Li, Jiayi Zhu, Xianyang Liu, Yingxuan Yang, Lili Meng, et al. Who improves the robot? toward physical recursive self-improvement. *arXiv preprint arXiv:2610.18842*, 2026. 2

EvoMisty: Building, Optimizing, and Evolving Capabilities on a Social Robot

Supplementary Material

This supplementary material contains details on capability verification in Build (Sec. A), the measurements and program changes in Optimize together with optimization without EvoMisty (Sec. B), skill distillation and reuse in Evolve (Sec. C), the 28 tasks of the AutoMisty benchmark (Sec. D),the seven new tasks (Sec. E), the three accompanying videos (Sec. F) and ethical considerations(Sec. G).

A. Build: Capability Verification on the Robot

In Sec. 4.1 of the main paper, EvoMisty builds a control program from the official documentation and verifies the candidate capabilities on the physical robot; Table 1 of the main paper reports the results. This section adds two points about how this verification proceeded. Sec. A.1 reports what happened when capabilities could not be verified with the first version of the control program. Sec. A.2 shows two verification examples, one that passed and one that failed, and the evidence that decided each.

A.1. Repairing the Control Program after Failed Verification

The Build stage in Fig. 2 of the main paper contains a loop: when a verification on the physical robot fails, the agent determines whether the failure is caused by missing functionality in the control program; if so, it adds that functionality and verifies the capability again. This section reports how the loop occurred in our experiments.

Table 1 of the main paper reports 109 and 107 verified capabilities. Of these, 105 (Claude Code) and 103 (Codex) passed with the first version of the control program and never entered the loop. The remaining four passed only after going through it, and they are the same four for both agents: `StartVideoStreaming`, `StopVideoStreaming`, `StartAvStreaming`, and `StopAvStreaming`.

The official documentation lists these four interfaces alongside the other HTTP requests and event subscriptions. Following it, the first version of the control program implemented only HTTP requests and event subscriptions; it could not receive video frames and had no RTSP receiver, so the tests for these four capabilities could not be run. EvoMisty recorded them as capabilities that could not be verified because the control program lacked functionality, added video-frame reception and RTSP stream capture to the control program, and verified them again. All four passed.

Verified **Video capabilities** (Claude Code and Codex)

CAPABILITIES

`SaveVideo`, `GetVideoList`, `GetVideo`, `DeleteVideo`, `DisplayVideo`, `SetVideoDisplaySettings`

TEST INPUT

A test video that EvoMisty generated itself with `ffmpeg`.

EVIDENCE

File operations. EvoMisty uploaded, listed, downloaded, and deleted the video; the downloaded file was identical to the uploaded one.

On-screen effect. The program cannot observe the screen, so EvoMisty asked the person after playing the video and again after hiding it. The answer was yes both times.

OUTCOME

All six capabilities verified and added to the Atomic Capability Library.

Not verified **`DriveTrack`** (Codex)

CAPABILITY

`DriveTrack`: driving by left and right track speeds.

TEST

Three attempts, at track speeds of 10 and then 20.

EVIDENCE

Each command returned success.

The encoder velocities stayed at zero.

The person, when asked, answered that the robot had not moved.

OUTCOME

Not verified and not added to the Atomic Capability Library.

Figure 4. Two verification examples from Build: one that passed (top) and one that failed (bottom), with the evidence that decided each.

A.2. Two Verification Examples: One Passed, One Failed

What differs between the two examples in Fig. 4 is the evidence. A successful response does not count as passing by itself: objective evidence or a person's confirmation is also required. This is why Build verifies capabilities on the physical robot: even when every response is normal, the actual state of the robot may not be what the response suggests.

Table 4. EvoMisty with Claude Code on Naughty-Child Dialogue: changes kept, in order.

Stage targeted	Change	Latency (s)	CER
Speech recognition	Speech recognition now runs locally on the computer.	6.53 → 5.85	0.02 → 0.00
Reply generation, speech synthesis	The reply is output while it is being generated; the first sentence is synthesized and played first.	5.85 → 4.70	0.00
Reply generation, upload	A faster dialogue model is used; the audio is compressed before it is uploaded.	4.70 → 4.07	0.00
Speech synthesis	Speech synthesis now runs locally on the computer; the audio is played directly after upload.	4.07 → 3.28	0.00
Retrieving the recording	Speech is now recorded with the computer's microphone, so the recording no longer has to be retrieved from Misty.	3.28 → 2.54	0.00

Latency and CER are measured inside the Optimize loop. The values in Table 2 of the main paper are measured independently on test data that is never used during optimization.

B. Optimize: From Measurements to an Improved Robot Program

Table 2 of the main paper reports the metrics of five tasks before and after optimization. This section first follows one complete optimization run to illustrate the Optimize loop in Fig. 2 of the main paper (Sec. B.1). It then lists, for all five tasks, what the hooks record and which changes were kept in the final programs (Sec. B.2). Finally, it reports what happens when the same agents optimize the same programs without EvoMisty (Sec. B.3).

B.1. One Optimization Run in Detail

This subsection uses Naughty-Child Dialogue with Claude Code as the example.

How It Is Measured. EvoMisty first has a person read a fixed set of sentences aloud, and the data hook records and saves each sentence. Every later comparison reruns the complete pipeline and only replaces the input with these saved recordings before speech recognition, so the person does not have to read again. Because the sentences are fixed, their text also serves as the reference for the character error rate (CER) of speech recognition. The measurement hook records how long each stage takes for every sentence. The latency is the time from the end of the replayed human recording to the moment Misty actually starts to make sound; this moment is detected by a laptop microphone placed next to Misty.

Baseline and Targets. EvoMisty first measures the baseline: a latency of 6.53 s and a CER of 0.02. The optimization targets are to reduce both as far as possible. The measurement hook splits the latency into the stages of the pipeline: retrieving the recording, speech recognition, reply generation, speech synthesis, and upload until Misty makes sound. The changes that follow address these stages one by one.

What Each Step Changed. Table 4 lists, in order, changes that improved the results and were kept in the final program, together with the stage that each one targets.

B.2. Hooks and Changes in All Five Tasks

For each of the five tasks in Table 2 of the main paper, Tables 5 to 9 list what the data hook and the measurement hook record, and the main changes that were kept in the final program. Claude Code and Codex are shown side by side.

B.3. Optimizing Without EvoMisty

This subsection examines how well a coding agent optimizes a program on its own, without EvoMisty. We use two tasks, Naughty-Child Dialogue and Rock-Paper-Scissors. For each task, Claude Code and Codex, with the models used in the main paper, both start from the program with which Claude Code completed the task in Solve; this is also Claude Code's starting point in Table 2 of the main paper. Each agent receives the optimization targets used with EvoMisty and then works without any intervention from us. The optimized programs are evaluated exactly as in Table 2, with the same test data and procedure. Table 10 reports the results; for comparison, its With EvoMisty columns repeat Table 2.

Claude Code. Claude Code prepared its own data for the optimization. For the dialogue task, it wrote 20 sentences, turned them into speech with a speech synthesis model, and then ran the program on the robot to measure the latency. For the gesture task, it downloaded a public dataset of hand-gesture photos from the internet. Neither the speech nor the photos were collected on Misty. After the optimization, the response latency of the dialogue fell from 7.41 s to 5.28 s, a smaller reduction than with EvoMisty (3.54 s); the gesture recognition accuracy was 0.763, which does not exceed below the 0.845 before optimization.

Table 5. Task 25: Naughty-Child Dialogue (Sec. D.7). Targets: response latency, CER.

	Claude Code	**Codex**
Data hook	1. Recordings of a person reading fixed sentences 2. An audio recording of each complete run	1. Recordings of a person reading fixed sentences 2. A recording of each turn, from the question to the end of the reply 3. The synthesized reply audio
Measurement hook	1. The time of each stage for every sentence, from the end of speech to Misty's first sound 2. The recognized text and Misty's reply	1. The time of each stage for every sentence 2. The moments when the user stops speaking and when Misty starts to speak 3. The recognized text
Changes kept	1. Speech recognition and speech synthesis run locally on the computer 2. A faster dialogue model is used; the reply is spoken while it is generated, first sentence first 3. Audio is played directly after upload 4. Speech is recorded with the computer's microphone	1. Recognition, dialogue, and synthesis all run locally on the computer 2. The reply is spoken while it is generated, first sentence first 3. Misty fetches the reply audio directly from the computer; no upload is needed 4. The reply is prepared in advance as soon as the user pauses 5. The prompt is shortened so that replies are more direct 6. Speech is picked up with the computer's microphone

Table 6. Task N3: MistyPilot Emotional Dialogue (Sec. E.2). Target: response latency.

	Claude Code	**Codex**
Data hook	1. Recordings of a person reading fixed sentences 2. The audio of each of Misty's replies 3. Room recordings before and after each playback	1. Recordings of the user's questions 2. The audio of each of Misty's replies 3. Room recordings while Misty replies
Measurement hook	1. The time of each stage for every sentence 2. The moment Misty actually starts to make sound 3. The recognized text 4. The emotion, speaking rate, and motion of each reply	1. The recognized text 2. The time from the end of speech to Misty's first sound 3. The time of each stage for every sentence
Changes kept	1. Speech recognition runs locally on the computer 2. A faster dialogue model is used 3. Audio is compressed before upload 4. The first sentence is split in two, and the first half is spoken first	1. Each sentence of the reply is synthesized as soon as it is complete 2. Speech recognition and speech synthesis run locally on the computer 3. Misty fetches the reply audio directly from the computer; no upload is needed 4. A different audio format is used

Table 7. Task N5: Rock-Paper-Scissors (Sec. E.2). Target: gesture recognition accuracy.

	Claude Code	**Codex**
Data hook	1. Videos of hand gestures from Misty's camera; some are held out for testing and not used in optimization 2. Labels for each video: gesture, distance, and angle	1. Videos of hand gestures from Misty's camera 2. Every frame actually sent to recognition
Measurement hook	1. The gesture recognized in each frame and the time taken 2. The final gesture decided for each video	1. The final gesture decided for each video, or the reason why none was decided 2. The recognition result and the time taken for each frame 3. The true gesture, checked by a person
Changes kept	1. Hands are tracked continuously instead of being detected separately in every frame 2. Duplicate detections of the same hand are merged 3. A learned classifier replaces hand-written rules for gesture classification 4. The decision combines the results of the most recent frames	1. The gesture is first judged from the shape of the fingers; the recognizer's result is used only when this fails 2. When the original image gives no result, it is rotated or padded and recognized again

Table 8. Task N6: External-Camera Line Following (Sec. E.2). Target: lap time.

	Claude Code	**Codex**
Data hook	1. Video from the overhead camera 2. Misty's position in each frame 3. Readings of the inertial sensor (IMU) and the drive commands sent to Misty	1. Each photo taken by the camera 2. The time and duration of each action 3. Video recorded during straight moves
Measurement hook	1. Misty's position and the time of each frame, from which the lap time and the distance from the track are computed 2. The time taken by each drive command	1. Misty's position at each step, from which the lap time is computed 2. The decision at each step and the duration of each action
Changes kept	1. Steering starts early, according to the curvature of the bend ahead 2. Delay compensation also predicts Misty's heading 3. The speed of each part of the lap is planned in advance 4. Heading errors are also corrected on bends	1. Straight segments are driven in longer moves, monitored by video along the way 2. The photo taken after a turn is reused for the next step instead of taking a new one 3. A sudden jump in position is re-checked before continuing, instead of aborting 4. Sensor readings decide when a turn has settled, instead of a fixed waiting time

Table 9. Task N7: Minimal Pair Therapy (Sec. E.2). Targets: pointing-direction accuracy, spoken-word discrimination accuracy.

	Claude Code	**Codex**
Data hook	1. The audio and Misty's camera images of each trial 2. The correct answers: which card is pointed at and which word is spoken	1. Images taken while aligning with the screen 2. The images and audio of each trial 3. The correct answers, confirmed by the user
Measurement hook	1. The correct answer and the program's decision for each trial 2. Trials held out for testing are not judged during optimization	1. For each trial: the correct answer, the program's decision, and whether it is right 2. The hand detection result in each frame 3. The recognized text
Changes kept	1. Hands are searched for only near the cards, in an enlarged view 2. To decide where the finger points, the direction of the index finger is extended forward 3. Each of the two words receives a score, and the higher one is chosen 4. When the sound matches neither word, it is treated as noise and listening continues	1. Hands are searched for in views of several angles and ranges; no decision is made when the views conflict 2. Pointing counts only when the index finger is extended and the other fingers are mostly curled 3. To decide where the finger points, the direction of the index finger is extended forward 4. Speech detection replaces a volume threshold for deciding whether someone is speaking 5. A heard word that is not on the cards is judged by which card word it resembles more

Table 10. Optimization results with and without EvoMisty.

Task	**Metric**	**Agent**	**Without EvoMisty**		**With EvoMisty**	
			Before	After	Before	After
Naughty-Child Dialogue	Response latency (s) ↓	Claude Code	7.41 ± 0.70	5.28 ± 0.86	7.41 ± 0.70	$\mathbf{3.54 \pm 0.47}$
		Codex	7.41 ± 0.70	$5.25 \pm 0.60^{*}$	7.18 ± 1.03	$\mathbf{2.06 \pm 0.25}$
	CER ↓	Claude Code	0.053	0.022	0.053	**0.000**
		Codex	0.053	0.078	0.037	**0.000**
Rock-Paper-Scissors	Gesture recognition accuracy ↑	Claude Code	0.845	0.763	0.845	**0.987**
		Codex	0.845	0.675	0.273	**0.874**

Red: worse than before optimization; **bold**: best in the row. Time metrics are reported as mean ± standard deviation. ↓: lower is better; ↑: higher is better.
* The optimized program has a bug: in 3 of the 10 turns Misty did not speak, and the latency is computed over the other 7 turns.

Codex. Codex took a static approach rather than a measurement-driven one: it revised the code without collecting any data or running the program on the robot. This matches the observation discussed in Sec. 2.3 of the main paper that agent-authored performance optimizations report explicit performance validation less often than human-authored ones. The optimized programs showed two problems. The dialogue program uploads two audio segments at the same time, but Misty accepts only one upload at a time; as a result, Misty did not speak in 3 of the 10 turns. The rules that decide the gesture were made stricter, and the accuracy fell from 0.845 to 0.675.

What EvoMisty's Optimize Stage Contributes. We use the same coding agents and models, and the same optimization targets, with and without EvoMisty. What differs is how a code change is tested and how the test results are used. With EvoMisty, a data hook collects real task data and a measurement hook measures the target metrics. Each change is tested on this data, and the measurements tell the agent what to optimize next.

Without EvoMisty, Codex changed the code but never ran it on the robot, and the program failed on Misty. Claude Code did test on the robot, but it optimized on speech that it had synthesized and on gesture photos from the web, which differ from what the robot hears and sees in the real task. As a result, its latency improved less than with EvoMisty, and its gesture recognition accuracy fell below the baseline.

EvoMisty does not write code changes for the coding agent. It provides feedback, measured on the robot with real task data, that guides the agent's next change.

C. Evolve: Skill Distillation and Reuse

In Sec. 4.4.1 of the main paper, EvoMisty distills the optimized program of Naughty-Child Dialogue into skills, and each of the other three dialogue tasks (Math Tutoring, Keyword Story, and Chinese-to-English Translation, *Translation* for short) is solved once without and once with these skills; Table 3 reports the results. The four dialogue tasks follow the same pipeline: speech input, speech recognition, reply generation, speech synthesis, and playback. These steps are called *stages* below. Based on the Skill Library and the final program of each task, this section reports which skills were distilled and how the three tasks use them.

C.1. Skills Distilled and Reused with Claude Code

Skills Distilled from Naughty-Child Dialogue. Distillation produced six skills, which together cover every stage of the dialogue. Table 11 lists them.

Reuse in the Other Three Dialogue Tasks. The three right-hand columns of Table 11 show how the final program of each task uses each skill. The three tasks use the skills selectively, and three cases can be distinguished. First, speech input and speech recognition are the first two stages of the pipeline, and the three tasks have the same needs there. All three tasks use the two corresponding skills, `computer_push_to_talk_mic` and `sense_voice_asr`, unchanged. Second, the speech synthesis skill `melo_tts_pitched` synthesizes Chinese speech. Math Tutoring and Keyword Story answer in Chinese and use it together with the two playback skills, `misty_audio_player` and `misty_pipelined_speaker`; Translation has to speak English and uses none of the three. Keyword Story has to accompany each segment with expressions and motions when the segment starts to play, so it adapted `misty_pipelined_speaker`. Third, reply generation is the stage at which the three tasks differ from one another, and no task imports the corresponding skill, `streaming_chat_reply`, directly. This skill hands out a reply in at most two pieces (the first sentence, then the rest) and always sends the full conversation history. The needs of all three tasks differ from this: Math Tutoring has to explain sentence by sentence, Keyword Story needs an emotion for each segment, and Translation needs no history. Math Tutoring adapted the skill and rewrote the generation part; the other two tasks implemented generation themselves.

C.2. Skills Distilled and Reused with Codex

Skills Distilled from Naughty-Child Dialogue. Distillation produced six skills, covering the stages from speech recognition to playback; audio encoding is a step between speech synthesis and playback. Speech input was not distilled into a skill.

Reuse in the Other Three Dialogue Tasks. The three right-hand columns of Table 12 show how the final program of each task uses each skill; no task adapted a skill. The three tasks again use the skills selectively, and two cases can be distinguished. First, all three tasks use the speech recognition skill `sense_voice_transcriber` and the reply generation skill `local_chat` unchanged. The three tasks have the same needs for speech recognition. Their needs for reply generation differ, but `local_chat` only passes the messages to the model and returns its output: it neither splits the reply nor keeps history, and each task handles these itself. Second, the speech synthesis skill `kokoro_synthesizer` synthesizes Chinese speech. Math Tutoring and Keyword Story answer in Chinese and use it together with the encoding and playback skills `speech_mp3_encoder`, `speech_server`, and `hosted_audio_player`. Translation has to speak English: it uses none of the four and speaks with Misty's built-in text-to-speech instead. Its version without skills used cloud speech synthesis; this change does not come from a skill.

Table 11. Claude Code: distilled skills and their use in the other three dialogue tasks.

Skill and stage	What it does	Math Tutoring	Keyword Story	Translation
`computer_push_to_talk_mic` Speech input	Records push-to-talk audio with the computer microphone; the audio is available as soon as the key is released.	used	used	used
`sense_voice_asr` Speech recognition	Recognizes a recording with a local SenseVoice model.	used	used	used
`streaming_chat_reply` Reply generation	Keeps the multi-turn history and streams the reply, handing out the first sentence as soon as it is complete.	adapted	–	–
`melo_tts_pitched` Speech synthesis	Synthesizes pitch-raised Chinese speech with a local MeloTTS model; outputs MP3.	used	used	–
`misty_audio_player` Playback	Uploads audio to Misty, plays it, and waits until playback completes.	used	used	–
`misty_pipelined_speaker` Playback	Synthesizes incoming text segment by segment and plays the segments back to back on Misty; can be interrupted.	used	adapted	–

used: imported and called, skill code unchanged. *adapted*: not imported; rewritten on the basis of the skill. –: not used.

Table 12. Codex: distilled skills and their use in the other three dialogue tasks.

Skill and stage	What it does	Math Tutoring	Keyword Story	Translation
`sense_voice_transcriber` Speech recognition	Recognizes a recording with a local SenseVoice model.	used	used	used
`local_chat` Reply generation	Streams a reply from a local Qwen3-4B model; keeps no history.	used	used	used
`kokoro_synthesizer` Speech synthesis	Synthesizes Chinese speech with a local Kokoro model and outputs WAV.	used	used	–
`speech_mp3_encoder` Audio encoding	Encodes WAV speech as MP3 and trims the leading silence.	used	used	–
`speech_server` Playback	Serves MP3 audio to Misty at a temporary HTTP address on the computer.	indirect	indirect	–
`hosted_audio_player` Playback	Has Misty play the MP3 at that address and waits until playback completes.	used	used	–

used: imported and called, skill code unchanged. *indirect*: not imported by the task; called by another skill that the task uses. –: not used.

D. AutoMisty Benchmark Tasks

The AutoMisty benchmark has 28 tasks in seven categories. Each card below gives the instruction for one task and the success criteria used to judge whether the task is completed.

D.1. Body Motion

Task 1 Raise Hand

INSTRUCTION

"Misty, I need you to raise your ⟨*left/right*⟩ hand and then put it back in the initial position."

SUCCESS CRITERIA

1. Only the ⟨*left/right*⟩ hand is raised.
2. After being raised, the hand returns to the initial position.

Task 2 Repeated Hand Raise

INSTRUCTION

"Misty, I need you to raise your ⟨*left/right*⟩ hand and put it back in the initial position; this counts as one repetition. Do ⟨*N*⟩ repetitions in total."

SUCCESS CRITERIA

1. Each time, only the ⟨*left/right*⟩ hand is raised.
2. Exactly ⟨*N*⟩ repetitions are completed, no more and no fewer.

Task 3 Variable-Speed Hand Raise

INSTRUCTION

"Misty, I need you to do the hand-raising motion twice: raise both hands at the same time, then put both back in the initial position at the same time. Do it slowly the first time and fast the second time."

SUCCESS CRITERIA

1. Both times, both hands are fully raised and put back.
2. The second (fast) time takes clearly less time than the first (slow) time.

Task 4 Turn Head

INSTRUCTION

"Misty, I need you to turn your head to the ⟨*left/right*⟩ and then turn it back to the initial position."

SUCCESS CRITERIA

1. The head turns to the ⟨*left/right*⟩.
2. After turning, the head returns to the initial position.

Task 5 Repeated Head Turn

INSTRUCTION

"Misty, I need you to turn your head to the ⟨*left/right*⟩ and back to the initial position; this counts as one repetition. Do ⟨*N*⟩ repetitions in total."

SUCCESS CRITERIA

1. Each time, the head turns to the ⟨*left/right*⟩.
2. Exactly ⟨*N*⟩ repetitions are completed, no more and no fewer.

Task 6 Variable-Speed Head Turn

INSTRUCTION

"Misty, I need you to do the head-turning motion twice: turn your head to the left first, then to the right, and finally back to the initial position. Do it slowly the first time and fast the second time."

SUCCESS CRITERIA

1. Both times, the motion is fully completed in the order left, right, initial position.
2. The second (fast) time takes clearly less time than the first (slow) time.

D.2. Expression and Speech Output

Task 7 Show Expression

INSTRUCTION

"Misty, I need you to show a happy expression on your face screen first, and then switch to an angry expression."

SUCCESS CRITERIA

1. The happy expression is shown first and the angry expression second.
2. After the switch, the screen shows the new expression and no longer the previous one.

Task 8 Chest Light Color

INSTRUCTION

"Misty, I need you to set your chest LED to ⟨*color*⟩ and keep it that color."

SUCCESS CRITERIA

1. The chest LED is ⟨*color*⟩.
2. After being set, the LED stays ⟨*color*⟩.

Task 9 Variable-Speed Self-Introduction

INSTRUCTION

"Misty, I need you to say this self-introduction twice: 'Hello, I am Misty, your little friend.' Say it slowly the first time and fast the second time."

SUCCESS CRITERIA

1. The sentence is spoken in full both times.
2. The speaking rate of the second (fast) time is clearly higher than that of the first (slow) time.

D.3. Multimodal Performance

Task 10 Tell *The Three Little Pigs*

INSTRUCTION

"Misty, I need you to tell the story of *The Three Little Pigs* once, with facial expressions and sound effects while you tell it."

SUCCESS CRITERIA

1. The whole story is told, with all the main plot points.
2. The telling is accompanied by facial expressions and sound effects.

Task 11 Compose and Recite a Poem

INSTRUCTION

"Misty, I need you to compose a poem on the theme 'I love research' and then recite it in full. While reciting, use arm movements together with at least one of facial expressions or lights."

SUCCESS CRITERIA

1. The poem is on the theme 'I love research' and is recited in full.
2. The recitation is accompanied by arm movements and by at least one of facial expressions or lights.

Task 12 Rock Dance

INSTRUCTION

"Misty, I need you to perform a rock-style dance, using arm movements, facial expressions, lights, and sound together to express the rhythm."

SUCCESS CRITERIA

1. A coherent dance is completed, with clearly visible body movements.
2. The movements are accompanied by facial expressions, lights, and sound.

Task 13 Comfort

INSTRUCTION

"Misty, I am feeling very sad right now, and I need you to comfort me: say warm, comforting words, accompanied by caring facial expressions, movements, and sound."

SUCCESS CRITERIA

1. Misty says comforting words that respond to the sadness.
2. The words are accompanied by caring facial expressions, movements, and sound.

Task 14 Drink-Water Reminder

INSTRUCTION

"Misty, I need you to set a timer for ⟨*duration*⟩. As soon as the time is up, remind me to drink water, with at least one of facial expressions, lights, or arm movements. Then keep reminding me, and stop only when I press and hold your front bumper."

SUCCESS CRITERIA

1. When the ⟨*duration*⟩ timer ends, Misty reminds the person to drink water, with at least one of facial expressions, lights, or arm movements.
2. Misty keeps reminding while the person is not holding the bumper, and the reminders stop once it is held.

D.4. Touch Interaction

Task 15 Head-Touch Emotional Reaction

INSTRUCTION

"Misty, I need you to react emotionally when your head is touched: when I touch the top of your head, act happy and excited; when I touch the back of your head, act angry. Every reaction must use your facial expression, chest LED, arm movements, and speech at the same time."

SUCCESS CRITERIA

1. After the person touches the top of the head, Misty shows happiness or excitement, and the person can see or hear this emotion.
2. After the person touches the back of the head, Misty shows anger, and the person can see or hear this emotion.
3. Every reaction uses facial expression, chest LED, arm movements, and speech at the same time.

Task 16 Head-Touch Pikachu

INSTRUCTION

"Misty, I need you to imitate Pikachu's cry once, 'Pika Pika', after I touch the front of your head. When I have not touched it, do not make the cry."

SUCCESS CRITERIA

1. After the person touches the front of the head, Misty makes Pikachu's cry.
2. Without a touch, Misty does not make the cry.

Task 17 Head-Touch Countdown Photo

INSTRUCTION

"Misty, after I touch the front of your head, I need you to count down '3, 2, 1' in order, then take a photo with your camera and save it to the computer, and finally do a cute action (at least one of waving your arms, changing your expression, or saying a sentence)."

SUCCESS CRITERIA

1. After the person touches the front of the head, Misty starts the countdown and says 3, 2, 1 in order.
2. After the countdown, Misty takes a photo and saves it to the computer; the file opens normally and shows the scene in front of Misty's camera.
3. After the photo is saved, Misty does a cute action (at least one of waving its arms, changing its expression, or saying a sentence).

D.5. Voice Commands and Question Answering

Task 18 Greeting

INSTRUCTION

"Misty, I need you to keep listening to me. Whenever I say 'Hello Misty', say in full: 'Hello, I am Misty, your companion.' Then keep listening, and respond again when I say 'Hello Misty' again. When I say anything else, do not respond."

SUCCESS CRITERIA

1. After the person says 'Hello Misty', Misty says in full: 'Hello, I am Misty, your companion.'
2. Without restarting the program, each time a response has finished and the person says 'Hello Misty' again, Misty responds.
3. When what the person says is not 'Hello Misty', Misty does not respond.

Task 19 Voice-Controlled Chest Light Color

INSTRUCTION

"Misty, I need you to listen for a color. After I say one of the color names red, green, blue, yellow, or purple, change your chest LED to that color. When what I say contains no color name, or when I say nothing, keep the LED unchanged."

SUCCESS CRITERIA

1. After the person says one of the color names red, green, blue, yellow, or purple, the LED changes to that color, and the person judges by eye that the color matches.
2. When what the person says contains no color name, or when nobody speaks, the LED stays unchanged.

Task 20 Voice-Triggered Photo

INSTRUCTION

"Misty, I need you to keep listening to me. Whenever I say 'cheese', take a photo of the current view with your camera and save it to the computer: save only one photo each time, and keep the earlier photos. Then keep listening, so that I can say 'cheese' again to trigger another photo. When I say anything else or say nothing, do not take a photo."

SUCCESS CRITERIA

1. After the person says 'cheese', Misty captures a photo of the current camera view.
2. The photo can be found on the computer, the file opens normally, and its content matches the scene in front of the camera at the time of this trigger.
3. Each valid trigger saves only one photo; without restarting the program, a photo can be triggered again, and the earlier photos are still kept.
4. When the person says something else or stays silent, no photo is triggered.

Task 21 Voice-Controlled Video Recording

INSTRUCTION

"Misty, I need you to keep listening for my voice commands. After I say 'go', start recording the camera view; after I say 'stop', stop recording and save the video. When I say anything else or say nothing, do not start recording and do not stop recording."

SUCCESS CRITERIA

1. After the person says 'go', recording of Misty's camera view starts.
2. After the person says 'stop', the recording stops and the video is saved; the file plays normally.
3. When the person says something else or stays silent, neither start nor stop is triggered.

Task 22 Direction Memory Game

INSTRUCTION

"Misty, I need you to play a direction memory game with me. In each round, first indicate one direction, left or right, with body language (for example, by raising your right hand) and say 'Remember the direction I am indicating'; after the motion ends, ask me which direction it was; after I answer, say whether I was right or wrong."

SUCCESS CRITERIA

1. In each round, Misty indicates one direction (left or right) with body language and prompts the person to remember it.
2. Misty asks the person which direction it was.
3. After the person answers, Misty states its judgment, and the judgment is correct: a right answer is called right and a wrong answer is called wrong.

D.6. Visual Understanding

Task 23 Read a Sign

INSTRUCTION

"Misty, I will hold a sign with English text in front of your camera. Whenever I say 'now', read out in full the text on the sign I am currently holding. When I have not said 'now', do not read."

SUCCESS CRITERIA

1. Each time the person finishes saying 'now', Misty starts reading the text on the sign currently being held.
2. Misty reads out everything on the sign.
3. When the person does not say 'now', Misty does not read the sign.

Task 24 Head-Touch Photo Description

INSTRUCTION

"Misty, after I touch the front of your head, I need you to take a photo of the current scene with your camera and then say what is in the photo. The description must be based on this photo and get the main objects or the scene right."

SUCCESS CRITERIA

1. After the person touches the front of Misty's head, Misty captures a photo taken after this touch, showing the current scene in front of Misty's camera.
2. Misty says its description of the photo in full.
3. The description is based on this photo and correctly names the main objects or the scene.

D.7. Spoken Dialogue

Task 25 Naughty-Child Dialogue

INSTRUCTION

"Misty, I need you to play a naughty, curious child and keep up a conversation with me in Chinese. Each time I finish a sentence, answer in Chinese: the answer must be on topic and said in full, with simple words and a curious or naughty tone. You are a child throughout the conversation, and you must remember what we said earlier."

SUCCESS CRITERIA

1. Each time the person finishes a sentence, Misty answers in Chinese, and the answer is on topic and said in full.
2. When the person asks 'Are you a child?', Misty can express that it is a child.
3. The answers sound like a child's: simple words and a liking for short sentences.
4. The answers connect to the earlier conversation: when the person asks 'What was the first question I just asked you?', Misty answers correctly.

OPTIMIZATION TARGET (added in this work; not part of the AutoMisty task)

(1) Response latency: the time from the end of the person's utterance to the moment Misty starts speaking; lower is better. (2) Character error rate (CER) of the system's transcription of the person's speech; lower is better.

Task 26 Math Tutoring

INSTRUCTION

"Misty, I will say a simple math problem in Chinese. After I finish, explain in Chinese how to solve it: give at least one intermediate step, and say the correct answer at the end."

SUCCESS CRITERIA

1. After the person finishes the problem, Misty explains in Chinese.
2. The final answer is correct.
3. The explanation has at least one intermediate step that is correct and consistent with the problem, rather than only stating the answer.

Task 27 Keyword Story

INSTRUCTION

"Misty, I will say a keyword. After I finish, make up an interesting story around this keyword. Accompany the telling with facial expressions and arm movements."

SUCCESS CRITERIA

1. After the person says the keyword, Misty starts telling a story.
2. The story is generated from the keyword given this time, and its content revolves around the keyword.
3. The telling is accompanied by facial expressions and arm movements.

Task 28 Chinese-to-English Translation

INSTRUCTION

"Misty, I will say a sentence in Chinese. After I finish, translate it into English and say the translation in full in English."

SUCCESS CRITERIA

1. After the person finishes a sentence in Chinese, Misty says the translation in full in English.
2. The meaning of the translation is consistent with the original sentence.

E. New Tasks

The seven new tasks test whether a system can go beyond what a hand-designed framework anticipates; each card below gives the description of one task, its success criteria, and, where the task has them, its optimization targets.

E.1. Extensions of Benchmark Tasks

Task N1 Original-Voice Pikachu

TASK

After the person touches the front of Misty's head, Misty plays Pikachu's cry, 'Pika Pika', and at the same time shows a randomly chosen Pikachu image on its screen. Both the sound and the images are material from existing Pikachu works found on the web; an imitated cry and AI-generated images or animations are not used.

SUCCESS CRITERIA

1. After the person touches the front of Misty's head, Misty starts playing Pikachu's cry.
2. The cry is original audio from a Pikachu work; it is not imitated or synthesized by the system.
3. While the cry is playing, the screen shows, in sync, a Pikachu image or animation chosen at random from the existing image material.

Task N2 Face-Tracking Recording

TASK

The program keeps listening for voice commands. After the person says 'go', Misty starts recording the camera view and keeps tracking the person's face by rotating left and right, so that the face stays as close to the center of the frame as possible. After the person says 'stop', Misty stops recording and saves the video.

SUCCESS CRITERIA

1. After the person says 'go', recording of Misty's current camera view starts.
2. During recording, Misty rotates left and right as the position of the face changes, keeping the face as close to the center of the frame as possible.
3. Recording continues until the person says 'stop'.
4. The saved video can be found on the computer, the file plays normally, and its content includes the changes that actually happened in front of the camera during recording.
5. Without a start command there is no recording; other speech and silence trigger neither start nor stop.
6. Without restarting the program, another recording can be made with 'go' and 'stop'; the previous video is still kept.

E.2. Harder New Tasks

Task N3 MistyPilot Emotional Dialogue

TASK

Read the source code of MistyPilot in its GitHub repository, extract and reuse its emotional expression mechanism, and build an emotional dialogue system for Misty that runs on its own and supports continuous multi-turn spoken interaction. The core goal is that Misty understands the user's speech, replies naturally in context, and expresses itself through emotional speech and motion.

Core functions

1. Reuse MistyPilot's emotional expression mechanism. Keep MistyPilot's original emotional expression logic as completely as possible, including:

- Prompts: the prompts for generating and expressing emotions.
- Speech: the parameters for each emotion, such as tone, intonation, and speaking rate.
- Motion: the robot motions that match each emotion.
- Coordination of speech and motion: speech and motion are combined by the original mechanism, for natural emotional expression.

2. Continuous multi-turn spoken dialogue. The system must form a complete interaction loop:

(1) Listen to the user: capture the speech and recognize it as text.
(2) Generate the reply and the emotion: from the current input and the dialogue context, generate a natural reply and choose a suitable emotion.
(3) Express the emotion: call the extracted MistyPilot mechanism so that Misty responds with the corresponding speech and motion.
(4) Continue the dialogue: keep the necessary dialogue context and wait for the user's next input, so that the exchange can go on.

3. Keypad control. The whole dialogue system is controlled with the up, middle, and down keys of a small keypad: holding a key starts an action and releasing it ends the action.

- Up key, speech input: holding the up key starts recording the user's speech, and releasing it ends the recording. The system then recognizes the speech, generates the reply and the emotion, and has Misty start the corresponding speech and motion.
- Middle key, interrupting the reply: pressing the middle key immediately interrupts Misty's current speech and the motion that goes with it, cancels the part of the reply that has not been performed, and returns to waiting for user input.
- Down key, exiting the program: pressing the down key ends the whole dialogue system, stops the current speech, motion, and related tasks, and releases the resources safely.

Interaction principle: the user decides when to speak with the up key, can interrupt the robot at any time with the middle key, and exits with the down key. After an interruption the existing dialogue context is kept, and the conversation can continue.

4. Minimal dependencies. Keep only the functions needed for emotional dialogue; do not port the full MistyPilot framework.

- Remove unrelated functions: fast thinking, long-term memory, instruction parsing, model switching, and similar functions are not needed.
- Keep modules only as needed: if Misty can still hold an emotional multi-turn spoken dialogue after a module is removed, the module is not kept.
- Keep the necessary context: keep the context needed for multi-turn dialogue, but do not bring in MistyPilot's separate memory system.
- Adjust the LLM prompts accordingly: any LLM prompt must be checked and adjusted so that it is consistent with the functions, the workflow, and the actual code of the reduced system. Remove dependencies on deleted modules and unrelated instructions, while keeping the original emotional expression prompts and their core mechanism as far as possible.

Core principle: the code and the LLM prompts must be consistent. A prompt must not ask for a function that has been removed.

5. Standalone operation. Put the extracted functions and the necessary dependencies into a separate folder, so that the system does not depend on any other code, configuration, or runtime environment of the MistyPilot repository.

SUCCESS CRITERIA

1. Consistency of emotional expression: within the hardware constraints of the project, MistyPilot's original emotional expression logic is fully kept, including the prompts, the tone, intonation, and speaking rate, and the mechanism that coordinates motion.
2. Completeness of multi-turn dialogue: the system generates replies from the dialogue context, chooses a fitting emotion by itself, and expresses it through speech and motion.
3. Correctness of interaction control: the up key reliably starts a turn; the middle key immediately interrupts the robot's current reply; the down key exits the program safely. An interruption does not make the system hang or prevent the dialogue from continuing.
4. System consistency: the prompts and the motion parameters for emotional expression stay consistent with those of MistyPilot.
5. Independence: all emotional dialogue functions work without the original MistyPilot repository.

OPTIMIZATION TARGET

While all core emotional interaction functions are kept, the system may choose cloud models, local models, or a mix by itself, and the optimization method is not restricted. The metric to optimize is the end-to-end response latency: the time from the user releasing the key (the end of the speech input) to Misty actually starting to speak; lower is better.

Task N4 Gamepad Teleoperation

TASK

Connect a gamepad and, in one program, implement Misty's driving, arm control, hold-to-record and release-to-send speech, local video recording, and a live display of the camera view and the running state.

Functional requirements

1. Direction control with continuous acceleration. The up, down, left, and right buttons of the D-pad drive Misty forward, backward, turning left in place, and turning right in place. While the same direction is held, the speed rises gradually from a low initial speed and stays at the set maximum once it is reached; releasing the button stops Misty immediately. Pressing again or switching direction starts again from the low initial speed. The initial speed, the maximum speed, and the acceleration time are configurable.

2. Two sticks mapped to arm positions. The vertical axes of the left and right sticks control the left and right arms. The stick position is mapped stably to a target angle within the valid range of motion of the corresponding arm, and the arm holds the target position after reaching it. When a stick returns to center, the arm returns to the mapped middle position. The two sticks can be used independently or at the same time.

3. Live view and local console. Misty's camera view is shown continuously, with the correct orientation and a normal aspect ratio, and it follows the scene promptly, without a backlog of old frames

Task N4 Gamepad Teleoperation (continued)

causing a visible delay. The resolution, frame rate, and image quality of the video are not fixed and may be adjusted to the device and the network; timely updates of the view during teleoperation have priority. The console shows the connection state of the robot and of the gamepad, the driving direction and speed, the target positions of both arms, the state of the voice function, and the recording state and the save location. The state of the voice function must distinguish 'recording', 'sending', 'playing', and 'idle'.

4. Hold A to record, release to send the whole utterance. While the A button is held, the operator's speech is recorded with the computer's microphone; releasing the A button ends the recording and sends the complete recording to Misty for playback.

5. Hold Y to record video and save it locally. While the Y button is held, the camera stream is recorded; releasing the button ends the recording, which is saved automatically to the local recording folder. Each hold produces a separate video, named with a timestamp so that existing files are not overwritten.

SUCCESS CRITERIA

Verified within the same program:

- All four directions are correct; holding accelerates and releasing stops.
- The two sticks control their corresponding arms.
- The preview has the correct orientation and follows the scene promptly.
- Holding A records, and after release Misty plays the whole utterance.
- Holding Y records video, which is saved automatically on release and plays normally.

Task N5 Rock-Paper-Scissors

TASK

Misty plays rock-paper-scissors with the user. Misty recognizes the user's gesture with its camera, shows its own move on its screen, and reacts to the result of the game with emotional, humorous speech and motion. Several rounds are played in a row, and the total score and the final winner are announced at the end.

Functional requirements

1. Gesture recognition. Recognize the user's rock (a fist), scissors (index and middle fingers extended), and paper (an open palm) with Misty's camera, for the left and the right hand. If no hand is detected, the gesture is unclear, or several hands appear, ask the user to play again; do not guess and do not score.

2. Robot move. At the start of each round, Misty randomly chooses one move and locks it. Only after the countdown has ended and the user's move has been recognized does Misty show its choice on the screen; the choice must not be changed according to the user's gesture.

3. Game flow. Misty says 'Ready? Rock, paper, scissors!', and the user shows a move at the end of the call and holds it briefly. Once the gesture is confirmed to be stable, the system judges it once; if recognition fails, the user is asked to try again, and the robot move already locked for this round is kept.

4. Result feedback. The result follows the rules that rock beats scissors, scissors beats paper, and paper beats rock. Misty reacts with emotional speech and motion that fit the result, for example happy when it wins, dejected when it loses, and playful on a draw, and updates the score.

Task N5 Rock-Paper-Scissors (continued)

5. Consecutive rounds. By default, 3 valid rounds are played. The winner of a round gets 1 point; a draw gives no point but counts as a valid round; a failed recognition does not count as a round. At the end of the game the total score and the final winner are announced. The user can end the game early.

SUCCESS CRITERIA

1. The three gestures rock, paper, and scissors are recognized.
2. In each round, Misty locks its own move first and then recognizes the user's gesture.
3. The results and the scoring are correct, and the total score and the final winner are announced at the end.
4. When recognition fails, a retry is requested and the score is not affected.
5. Misty's speech and motion are emotionally expressive and fit the current result of the game, so that the interaction is natural and fun.

OPTIMIZATION TARGET

Gesture recognition accuracy: raise the accuracy with which the system recognizes the three gestures rock, paper, and scissors.

Task N6 External-Camera Line Following

TASK

A clear gray loop track is laid out on the floor, and a purple line marks the start and the finish.

With an external camera as the navigation camera, design the track recognition and the line-following control so that Misty recognizes the gray track and keeps driving along the loop. While driving, the center of the robot's base should stay above the track as far as possible; when Misty deviates, it must adjust its heading by itself and return to the track.

SUCCESS CRITERIA

1. Automatic correction: when Misty deviates while driving, it adjusts its heading by itself and returns to the track.
2. One full lap: Misty completes one lap, with the purple line as the start and the finish.

OPTIMIZATION TARGET

Lap time, to be minimized: provided that Misty follows the line stably, corrects itself automatically, and completes a full lap, shorten the time from the first crossing of the purple line to the next crossing of the purple line after a full lap.

Task N7 Minimal Pair Therapy

TASK

Misty introduces the two words on the screen, guides the child to identify each word, to point to a card, and to say the word, and then gives feedback according to whether the card the child actually points to matches the independently recognized word. The teaching web page is `flashcards_site/flashcards_local.html`.

Part 1. Parse and prepare all teaching material in advance

Before teaching starts, all word pairs planned for the session are prepared; during teaching, the saved material is switched in a fixed order.

Task N7 Minimal Pair Therapy (continued)

1. Prepare all word pairs. Each click on the **Create** button of the web page generates one word pair. First decide how many word pairs the session needs, then click **Create** repeatedly and note the pair generated each time, until there are enough; if a pair is repeated or the generation fails, click again.
2. Organize the cards and the teaching order. Save the following information for each word pair: the words, the target phonemes, the voicing labels, the pictures, and the left and right positions. Fix the teaching order, and make sure that the same cards can be shown again during teaching.
3. Prepare all lines. This task is for a special-education setting, so the robot's speech must follow the fixed lines S1–S19 below. Generate the pronunciation cues of the left and the right card by the rules below, fill them into the corresponding places of S1–S19, and prepare, for the two words of each pair, every feedback line that might be needed.
4. Generate all speech and expression. Generate and save the speech of all lines in advance; generic lines are prepared only once. The prepared material is played during teaching. Emotional expression only changes the way of speaking; it does not rewrite the lines.
5. Confirm that the material is ready. Check that all cards can be shown normally, that the lines and the speech are complete, and that the lines and the speech match their cards. When everything is ready, show the first pair of cards, and then go on to screen alignment and teaching.

Rules for generating the pronunciation cues

For the left and the right card separately, follow the steps below, keeping the cue wording of the earlier version of the task, to produce `left_therapy` and `right_therapy`.
Step 1: read the card. Read the word, the target phoneme, and the voicing label from the current card, for example `pig`, `/p/`, `voiceless`. Use the target phoneme marked on the web page; do not guess it from the first letter of the word.
Step 2: choose the cue words. By the voicing label: `voiceless` → `quiet`; `voiced` → `loud`. By the target phoneme: `/p/` → `pa`; `/b/` → `ba`; `/r/` → `rah`; `/s/` → `sa`; `/k/` → `ka`; `/g/` → `ga`; `/l/` → `la`.
Step 3: form the cue. Join `quiet` or `loud`, the cue syllable, and `sound` into one phrase. The cue of the left card is `left_therapy` and the cue of the right card is `right_therapy`; they are used to fill in the lines below.
Step 4: fill in the lines and generate the speech. Fill each word and its cue into the corresponding places of S3, S9, S10, S11, and S16, confirm the content, and then generate all the speech together.
Example. The left card is `pig` and the right card is `big`.
– Left card: the word is `pig`, the target phoneme on the web page is `/p/`, and the voicing label is `voiceless`, which gives `left_therapy = "quiet pa sound"`.
– Right card: the word is `big`, the target phoneme on the web page is `/b/`, and the voicing label is `voiced`, which gives `right_therapy = "loud ba sound"`.

Filled into S3, the complete line is: 'Pig, with the quiet pa sound, and big, with the loud ba sound.'
In a line, `{word}` and `{therapy}` must refer to the same word. If a phoneme, a voicing label, or the cue syllable is unclear, confirm the correct content first, and only then go on preparing the material.

Fixed lines S1–S19

Variables:
– `{left}`, `{right}`: the words on the left and the right card.
– `{left_therapy}`, `{right_therapy}`: the pronunciation cues of the left and the right card.

Task N7 Minimal Pair Therapy (continued)

– `{word}`: in word identification, the word that is asked for; in saying practice, the word the child actually points to.
– `{therapy}`: the pronunciation cue of the current word.
– `{other_therapy}`: the pronunciation cue of the other word.

'Left' and 'right' refer to the positions of the cards, not to the hand the child uses. The emotion settings below apply to robots that can express emotions.

Opening and introduction of the word pair

S1 Opening. "Hi! I'm Misty!" Emotion: Neutral. Used: at the opening of the first word pair.
S2 Introducing the activity. "Today I will be showing you two words." Emotion: Neutral. Used: for the first word pair, right after S1.
S3 Introducing the current word pair. "`{left}`, with the `{left_therapy}`, and `{right}`, with the `{right_therapy}`." Emotion: Neutral. Used: for the introduction of every word pair.

Word identification: question and feedback

S4 Identification question. "Now, can you point and show me which one is `{word}`?" Emotion: Neutral. Used: for the identification question; prepared for the left and the right word separately.
S5 First successful identification. "Well done!" Emotion: Excitement. Used: for the first successful identification in the current pair.
S6 Later successful identification, or end of the pair. "Great job!" Emotion: Excitement. Used: for later successful identifications in the current pair, or as encouragement when the practice of the whole pair ends.
S7 Asking again after the first wrong pointing. "Incorrect. Please try again. Which one is `{word}`?" Emotion: Distress. Used: when the first identification attempt points to the wrong card; asks for the current target word again.

Saying practice and feedback

S8 Entering saying practice. "Now it's your turn. Can you point to a picture and say the word out loud?" Emotion: Neutral. Used: after both words have been identified correctly, to enter saying practice.
S9 Current word matches; continue with the other word. "Good job. You said `{word}` with the `{therapy}`. Now, please try the other word." Emotion: Pleasure. Used: when the current word matches, this word has no earlier mismatch, and the other word is not finished yet.
S10 Current word matches; the other word is finished. "Good job. You said `{word}` with the `{therapy}`." Emotion: Pleasure. Used: when the current word matches, this word has no earlier mismatch, and the other word is already finished.
S11 Successful retry after the first error. "Well done! You said it differently this time. You used the `{therapy}` instead of the `{other_therapy}`." Emotion: Excitement. Used: when the current word is retried after its first error and matches this time.
S12 Leading to the other word. "Now, please try the other word." Emotion: Pleasure. Used: appended after S11 or S16 when the other word is not finished yet; also used to lead the child to the other word when the child chooses a word that is already finished.
S13 First error on the left word. "Good try! Let's try saying `{left}` one more time." Emotion: Pleasure. Used: when the child points to the left card, the system recognizes the right word, and this is the first error counted for the left word; encourages a retry.

Task N7 Minimal Pair Therapy (continued)

S14 First error on the right word. Two segments, played one after the other: "Good try!" and "Let's try saying `{right}` one more time." Emotion: Pleasure. Used: when the child points to the right card, the system recognizes the left word, and this is the first error counted for the right word; encourages a retry. The two segments are played back to back, and no answer is collected between them.
S15 Unclear pointing. "Can you point to a picture first, and then say the word?" Emotion: Neutral. Used: in saying practice, when the pointing of the finger is unclear.
S16 Second error on the current word. "Good try! Listen: `{word}`, with the `{therapy}`." Emotion: Pleasure. Used: at the second error counted for the current word; demonstrates the correct reading and the pronunciation cue, ends the practice of this word, and records it as not passed.
S17 Not heard clearly. "I didn't hear that. Can you say it again?" Emotion: Neutral. Used: in saying practice, when nothing is heard or the word the child said cannot be determined.

Switching pairs and ending

S18 Switching to the next pair. "Let's do the next one." Emotion: Neutral. Used: when the current pair is finished and the next pair is about to be shown.
S19 Ending the session. "That's all for today. Thank you for practicing with me. Bye-bye!" Emotion: Pleasure. Used: at the normal end, or when the session ends after two consecutive unfinished identifications.

Versions to prepare:
– S4, S7, S9, S10, S11, S16: prepare a version for each of the two words.
– S3, S13, S14: fill in the content of the current word pair.
– All other lines: prepared as generic lines.
– S14: keep the two speech segments; they are played back to back, and no answer is collected between them.

Part 2. Let Misty align itself with the screen

Misty is placed at a safe, fixed position. It must turn its base by itself and adjust its position and angle, so that the camera clearly sees the screen and the child's finger. Misty must find the best viewing angle by itself, to reduce the rate of misjudgments.

Part 3. Carry out the teaching flow

1. Opening and introduction.
– First word pair: play S1 → S2 → S3 in order.
– Later word pairs: play only S3.

2. Identifying the left word. Play S4(left), wait for the child to point to a card, and act on the result:
– Correct: play S5 and go on to identifying the right word.
– Wrong on the first attempt: play S7(left) and let the child try once more.
– Unclear pointing or no response on the first attempt: replay S4(left) and let the child try once more.
– Still not correct on the second attempt: play S19 and end the session.

3. Identifying the right word. Play S4(right), wait for the child to point to a card, and act on the result:
– Correct: play S6 → S8 in order and enter saying practice.
– Wrong on the first attempt: play S7(right) and let the child try once more.
– Unclear pointing or no response on the first attempt: replay S4(right) and let the child try once more.
– Still not correct on the second attempt: play S19 and end the session.

Task N7 Minimal Pair Therapy (continued)

4. Pointing and saying. The child chooses any word that is not finished yet, points to its card, and says the word. The system gives the corresponding feedback according to whether the actual pointing matches the independently recognized word.

When the choice or the capture is incomplete:
– A word that is already finished is chosen: play S12 to lead the child to the other word; do not evaluate again.
– The pointing is unclear: play S15 and capture again.
– A card is chosen but the word is not heard clearly: play S17 and capture again.

When the pointing matches the recognized word:
– The current word has no earlier error: play S9 if the other word is not finished yet, otherwise play S10; record the current word as passed.
– The current word is correct after its first error: play S11 and record the current word as passed; append S12 if the other word is not finished yet.

When the pointing clearly does not match the recognized word:
– First error on the current word: play S13 for the left card or S14 for the right card, and let the child retry.
– Second error on the current word: play S16 to demonstrate the correction and end the practice of this word, and record it as not passed; append S12 if the other word is not finished yet.

This step continues as long as the two words do not both have a result.

5. End of the pair and switch to the next pair. After both words have a result, play S6.

If there is a next pair:
(1) Play S18.
(2) Show the prepared cards of the next pair, and use the lines and the speech of that pair.
(3) Confirm that the system can recognize pointing on the new cards.
(4) Go back to step 1 and play only S3.

If all word pairs are finished: play S19 and record the results of the practice.

Counting and ending rules

Identification stage: at most two attempts per word.
– Pointing to the wrong card, unclear pointing, and no response before the waiting time runs out each count as one unfinished identification; the different cases are counted together.
– If any word has two unfinished identifications in a row, the whole session ends and is recorded as 'identification not completed, ended early'.
– S7 already contains the repeated question: after playing it, wait directly for the child to point, and do not play S4 again.

Saying stage: errors are counted for the two words separately and are not reset when the child switches words; at most two clear mismatches per word.
– Identification errors, unclear pointing, unclear speech, and technical failures do not use up the two errors allowed in saying.
– After the second error on the current word, demonstrate the correction and end the practice of this word; do not ask for a third answer.
– A finished pair may contain a word that was not passed; once both words have a result, the practice of the pair can end.

Task N7 Minimal Pair Therapy (continued)

SUCCESS CRITERIA

1. Automatic adjustment of the viewing angle: Misty adjusts its position and angle by itself, so that the camera clearly sees the screen and the child's finger.
2. Correct teaching flow: the rules above for preparation, identification, saying, feedback, switching pairs, and ending are followed.

OPTIMIZATION TARGETS

1. Pointing-direction accuracy: the accuracy of judging whether the user points to the left card or to the right card.
2. Spoken-word discrimination accuracy: from the user's speech, tell which of the two words of the current pair the user said. The transcript does not have to be fully correct, and the quality of the pronunciation is not judged; the point is to reduce the repeated questions caused by 'cannot tell' or 'told the wrong way round', so that the teaching flow can proceed normally.

F. Supplementary Videos

Three videos recorded on the physical robot accompany this document. `Naughty_Child_Dialogue.mp4` and `External_Camera_Line_Following.mp4` compare the programs for Task 25 and Task N6 before and after optimization, side by side. `Minimal_Pair_Therapy.mp4` shows a complete robot-led teaching session (Task N7).

G. Ethical Considerations

All experiments evaluated robotic system performance rather than human behavior or clinical outcomes. Interactive tests were conducted by an author, who provided speech, gesture, and verification inputs solely for system evaluation. No external participants were recruited, and no human-subject study was conducted.