MistyPilot: Enabling Social-Robot Control through Multi-Agent LLM Skill Orchestration

Xiao Wang*, Lu Dong*, Ifeoma Nwogu, Srirangaraj Setlur, and Venu Govindaraju

State University of New York at Buffalo
https://wangxiaoshawn.github.io/MistyPilot.html

Abstract. Programming small social robots from natural-language instructions requires more than invoking isolated APIs. Interactive tasks combine reactive physical behaviors with stateful social behaviors, while existing interfaces often require developers to manually compose APIs into skills, configure their parameters, bind sensor events to skills, and manage task states at runtime. We present MistyPilot, a multi-agent LLM framework that interprets high-level natural-language instructions and orchestrates the corresponding skills on the Misty social robot. A Task Router dispatches each instruction to one of two specialized agents: a Physically Interactive Agent for sensor-triggered robot control and direct skill invocation, and a Social Interaction Agent for dialogue-oriented task-state management and context-dependent multimodal response generation. To improve efficiency, the Social Interaction Agent reuses previously generated results when applicable and invokes full generation otherwise. We evaluate MistyPilot on five component-level suites, with sensor bindings and skill invocations executed on the physical Misty robot, and a preliminary user study with 12 participants. MistyPilot attains high accuracy on routing, sensor-skill binding, task-state parsing, result reuse, and skill extension up to 100 skills, and lower variance than an otherwise identical single-agent baseline, while participants report positive perceptions of usability and interaction quality. The code will be made publicly available via the project page.

Keywords: Social robots · Multi-agent LLMs · Robot skill orchestration · Natural language robot programming

1 Introduction

Social robots are increasingly used for everyday assistance, education, and social interaction [4,9,10,12,26,29,31,35]. A key challenge for social-robot platforms is enabling everyday users, rather than only programmers, to control robot behavior. This accessibility is particularly important in assistive settings, where users may have little or no technical background. Platforms such as Misty partially address this challenge by exposing open APIs for speech, vision, head and arm

* Equal contribution.

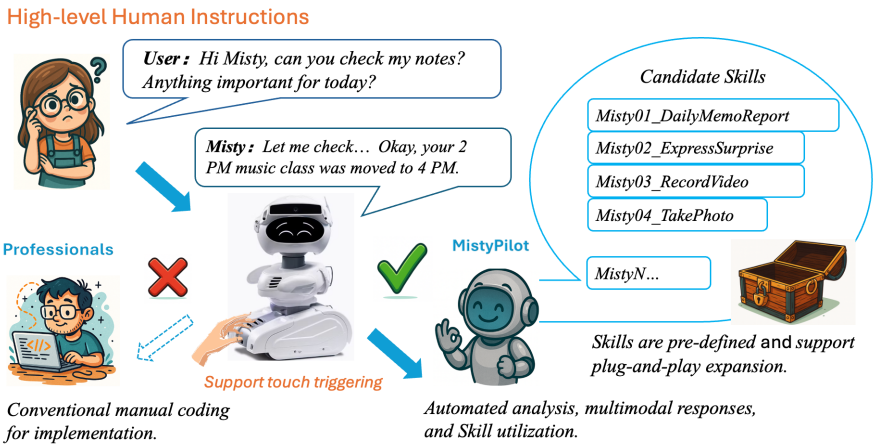

Fig. 1: Overview of MistyPilot workflow for interpreting high-level human instructions. Instead of requiring professionals to hand-code robot behaviors and deploy features on the Misty robot, MistyPilot parses natural-language instructions, analyzes the task, selects and parameterizes skills from its library, and executes them on the Misty Robot.

motion, and touch sensing [5, 30]. However, using these capabilities still requires developers to implement robot behaviors as code by composing APIs into skills, configuring their parameters, binding sensor events to skills, and managing task states at runtime [7]. Real user requests are difficult to enumerate in advance because they are often diverse, open-ended, and subject to change during interaction. In a single session, for instance, a user might first say, "Greet me whenever I tap your head," and then ask the robot, "Tell me a bedtime story," later requesting a different ending and asking to hear the story again. The first is a reactive physical-interaction request that binds a sensor event to a robot skill, whereas the second is a stateful social-interaction request whose execution depends on dialogue history and task progress. Supporting both forms of interaction requires the robot to interpret user instructions and orchestrate the corresponding skills at runtime. Yet current workflows for social robots such as Misty still require developers to anticipate and manually implement these interaction patterns in advance, leaving non-expert users with limited ability to create or modify them at runtime.

These limitations arise from three technical gaps in current social-robot workflows. First, reactive physical interaction requires the robot to invoke skills directly and persist sensor-skill bindings so that they remain available across system restarts. As the skill library grows, integrating new skills typically requires developers to modify the robot's control logic manually. On Misty, AutoMisty [30] reduces the effort of creating individual skills by using an LLM to generate control code, but it does not discover, invoke, or orchestrate those skills

at runtime. General LLM-based tool agents [24, 27, 34] provide flexible mechanisms for selecting and invoking tools from natural-language instructions at runtime. However, general-purpose tool orchestration remains largely underexplored on Misty social robots, where execution must additionally account for robot-specific skills and sensor-triggered interactions. Second, stateful social interaction requires the robot to maintain task progress across dialogue turns. Users may revise an ongoing request, refer to earlier results, or ask the robot to repeat an updated task. Treating each utterance as an independent request makes it difficult to preserve dialogue context and execute evolving multi-step tasks consistently. Current workflows provide limited support for explicit dialogue-oriented task-state management. Beyond state management, social responses should also be generated efficiently and expressed through coordinated robot modalities. Generating similar results from scratch for every request introduces redundant computation and latency. At the same time, basic text-to-speech and weakly coordinated motion can make robot responses feel mechanical. An effective system should therefore reuse suitable prior results when possible, invoke full generation when necessary, and produce context-dependent responses across speech, motion, and facial cues.

To address these limitations, we present MistyPilot, a multi-agent LLM framework that interprets high-level natural-language instructions and orchestrates the corresponding skills on the Misty robot at runtime. Its role-specialized architecture separates reactive physical interaction from stateful social interaction, enabling each type of task to be handled within a dedicated state and skill space. Our main contributions are summarized as follows:

- **A role-specialized architecture for reactive and stateful interaction.** We introduce a multi-agent architecture that explicitly separates reactive physical interaction from stateful social interaction. A *Task Router* directs each instruction to either a *Physically Interactive Agent (PIA)* or a *Social Interaction Agent (SIA)*, avoiding the need to combine routing, sensor-triggered control, and dialogue management within a single agent context.
- **Specialized runtime orchestration for physical and social behaviors.** The *PIA* supports direct skill invocation, persistent sensor-skill bindings, and runtime integration of new skills. The *SIA* maintains explicit task states across multi-turn dialogue, reuses previously generated results when applicable, falls back to full generation otherwise, and produces context-dependent multimodal responses across speech, motion, and facial cues.
- **Component-level evaluation and preliminary user feedback.** We evaluate MistyPilot on the Misty robot using five component-level suites covering routing, sensor-skill binding, task-state parsing, result reuse, and skill extensibility, together with a preliminary in-person user study. Under controlled Misty-specific settings, the system demonstrates reliable component-level performance and positive participant perceptions of usability and interaction quality.

2 Related Work

2.1 Tool-augmented and embodied LLM agents

LLM agents have progressed from pure reasoning to tool use. Toolformer [27] learns to call APIs within text generation, ReAct [34] interleaves reasoning and acting to query external tools, ToolLLM [24] orchestrates many tools through tree-structured search, and Tulip Agent [20] scales to large tool libraries via vector-store retrieval; benchmarks such as API-Bank [15], APIBench [22], and ToolAlpaca [28] standardize their evaluation. In the physical world, AutoRT [1], Odyssey [18] pair tool use with perception and planning. However, these works are not designed for social robots; they mainly target software tool calling or robotic motion planning, leaving their application to social-robot interaction largely unexplored.

2.2 Social Robots for Human–Robot Interaction

Prior work on social robots has explored affect recognition in assistive and mental-health settings [14], but generating emotionally expressive multimodal responses remains underexplored. A related challenge is enabling non-programmers to create and control robot behaviors. AutoMisty [30], the closest work to ours, uses a multi-agent LLM system to generate executable Misty skills for non-programmers, but it does not invoke or orchestrate those skills at runtime. MistyPilot complements AutoMisty by using a given skill library to interpret natural-language instructions and select, bind, and invoke skills on the physical robot, while also managing dialogue task state and generating emotion-conditioned multimodal responses.

3 PROBLEM FORMULATION

To handle the open-ended interaction scenarios faced by social robots, we first formalize a Task Space $\mathcal{T}$. Given a natural language task instruction $T_i \in \mathcal{T}$ from this space, the objective is to construct an end-to-end tool orchestration pipeline that executes the specified task. To handle heterogeneous tasks, MistyPilot first employs a Task Router R to dispatch the initial task T_i to the appropriate tool agent K^*:

$$K^* = R(T_i), \quad K^* \in \{PIA,\ SIA\} \tag{1}$$

Once dispatched to the selected agent branch, the corresponding agent (SIA or PIA) parses the task T_i, automatically selects the required functions f_j from a predefined tool library $\mathcal{F}$, and infers their corresponding parameter configurations $\theta_j \in \Theta_{f_j}$. Since completing a task typically requires invoking a series of functions, the final execution of task T_i is represented as an ordered sequence of parameterized function calls:

$$Agent(T_i) = \big(f_1(\theta_1), f_2(\theta_2), \ldots, f_N(\theta_N)\big) \tag{2}$$

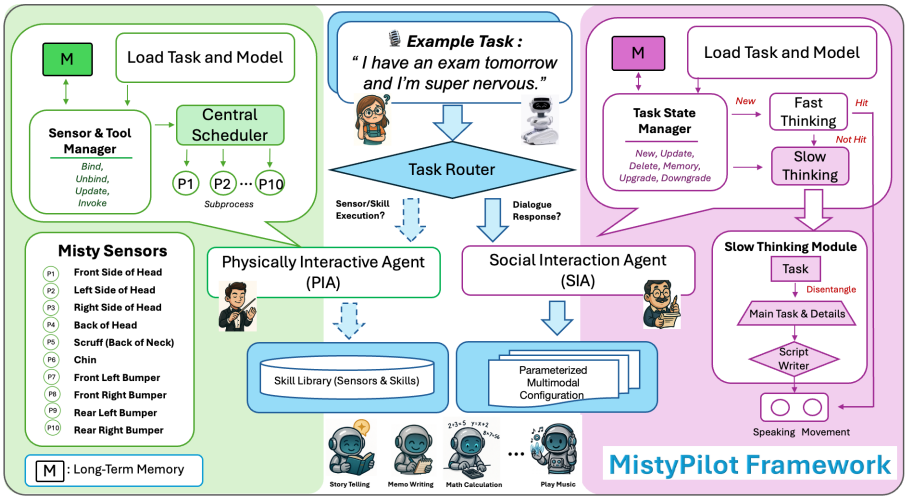

Fig. 2: Overview of the MistyPilot framework. A *Task Router* dispatches each instruction to the *Physically Interactive Agent (PIA)* for sensor-skill binding and skill invocation, or to the *Social Interaction Agent (SIA)* for dialogue. The *SIA* tracks task state and responds through a fast path that reuses stored results, or a slow path that generates emotion-conditioned multimodal output.

Based on this formulation, the core objective of MistyPilot is to infer a valid sequence of function calls that satisfies the task requirements:

$$MistyPilot(T_i) = Agent_{K^*}(T_i) = \left(f_j(\theta_j)\right)_{j=1}^{N} \tag{3}$$

4 METHODOLOGY

To handle open ended human robot interactions, MistyPilot adopts a multi agent framework built on the tool agent paradigm [16, 25, 33], separating task routing, physical interaction, and social dialogue. A tool agent translates high level natural language instructions into parameterized skill calls and invokes registered tools to execute complex tasks, as illustrated in Fig. 2. Crucially, MistyPilot adopts role separation at the architecture level rather than relying only on prompt based specialization. Unlike single agent systems that collapse the full context and all available skills into one agent, each agent in MistyPilot maintains its own localized state and skill space. The following subsections describe the design of the *Task Router*, *SIA*, and *PIA*.

4.1 Task Router

Given a task, MistyPilot first employs the *Task Router*, an LLM agent that acts as the central dispatcher. It analyzes the high-level natural-language instruction

to determine the user's core intent. If the task involves sensor-triggered events or direct skill invocation, the router dispatches it to the *PIA*; if the task is dialogue-oriented, it routes it to the *SIA* to produce an emotion-conditioned multimodal response.

4.2 Social Interaction Agent (SIA)

When a dialogue-oriented task is dispatched to the *SIA*, it is first processed by the *Load Task and Model* module for state parsing and selection of the appropriate LLM version for reasoning. The *Task State Manager (TSM)*, *Fast Thinking*, and *Slow Thinking* modules then coordinate to orchestrate the skills needed to complete the task.

***Load Task and Model*:** This module converts user instructions into a task-state representation via an LLM and a schema-constrained system prompt. It maps natural-language intent to a predefined action set: `NEW`$\langle main\_task, details\rangle$, `UPDATE`$\langle details\rangle$, and `DELETE`$\langle details\rangle$ support task-state tracking and modification; `UPGRADE` and `DOWNGRADE` switch between a higher-capacity and a lightweight LLM for subsequent script generation; and `MEMORY` supports long-term information persistence upon user request.

***Task State Manager (TSM)*:** Prior studies show that LLMs accumulate faithfulness errors during iterative dialogue summarization, particularly in long-context settings [3, 17, 19, 23].To address this and stabilize behavior, we use an explicit, editable external task-state memory to store and manage task states, reducing dependence on full-context recomputation. Each update is atomic and localized, leaving other parts unchanged. Table 1 shows an example user prompt and its TSM state: the details are stored as atomic units $(d_1, ..., d_n)$, so an `UPDATE` that changes the return time to 6:00 PM modifies only d_2 while the rest remain unchanged.

Table 1: An example task status managed by the TSM. An `UPDATE` to one detail modifies only the affected atomic unit (d_2); the other units stay unchanged.

User Input 1: "Hi Misty, I'd like to plan a day trip to New York City for tomorrow. Please create an itinerary that allows me to enjoy the city and return to my hotel by 7:00 PM."

User Input 2: "No, no. I need to return by 6:00 PM."

	TSM State	**After `UPDATE`(return by 6 PM)**
main_task	Plan a day trip to New York City	(unchanged)
details	d_1: date = tomorrow	(unchanged)
	d_2: return by 7:00 PM	d_2: **return by 6:00 PM**
	d_3: goal = enjoy the city	(unchanged)

***Fast and Slow Thinking*:** We adopt a dual-channel strategy inspired by fast and slow cognition in human problem solving [8, 13, 21], with *Fast Thinking* (retrieve and reuse) as the default and *Slow Thinking* (generate from scratch) as a fallback. The reusable memory $\mathcal{M}$ is populated on demand: when the user marks a result worth keeping, the `MEMORY` action stores it, indexed by its *main_task* embedding. In *Fast Thinking*, we encode the current *main_task* into a query embedding $\mathbf{q}$ and compare it against the stored embeddings $\mathbf{x}$ in $\mathcal{M}$ by cosine distance, as defined in Eq. (4). The embedding space can be chosen flexibly; we compare three different embedding spaces, which yield consistent retrieval (Table 7). We set the threshold $\tau = 0.4$ based on the observed distance distribution. If the minimum distance exceeds τ, including when no relevant item has been saved, we treat the task as a retrieval miss and fall back to *Slow Thinking* for reasoning and generation.

$$\min_{\mathbf{x}\in\mathcal{M}} d_{cos}(\mathbf{q}, \mathbf{x}) \leq \tau \tag{4}$$

In *Slow Thinking*, the *Script Writer* orchestrates the task with utterance-level emotion conditioning. Each utterance u_i is assigned a multimodal emotion label from eight categories based on the extended Ekman emotion taxonomy [6]: *Happiness*, *Sadness*, *Anger*, *Fear*, *Disgust*, *Surprise*, *Contempt*, and *Neutral*. Table 2 illustrates this process.

We express emotions consistently across three modalities: text, motion, and voice. Text is generated via context-aware reasoning. For motion, Misty executes expressive behaviors (e.g., arm waving, head tilting, body swaying, and light flashing) using templates aligned with Ekman's basic emotions [6], with LED colors and facial expressions reinforcing discriminability and cross-modal consistency. For voice, we bypass the built-in TTS and use OpenAI-TTS for controllable speech synthesis (timbre, speaking rate, and intonation). It (i) adapts prosody and intonation to text semantics and punctuation, (ii) injects emotion-conditioned style by encoding affective attributes such as arousal, intensity, and prosodic patterns [11], and (iii) controls speaking rate conditioned on emotion labels. Following [11], we discretize speaking rate by arousal: 1.00 (baseline), 0.95 (low arousal), and 1.05 (high arousal), yielding synchronized action–speech expressions. Finally, the *SIA* delivers motion and voice to the *Speaking* and *Movement* modules for simultaneous execution on Misty.

4.3 Physically Interactive Agent (PIA)

Tasks involving sensor-triggered events or direct skill invocation are routed to the *PIA*, which performs reactive, event-driven robot control by mapping sensor events to skills and executing them when triggered. The *PIA* uses the *Load Task and Model* module to interpret user intent and either (i) directly execute a skill or (ii) identify the target sensor and retrieve the corresponding skill from the *Skill Library*. It then uses the *Sensor & Tool Manager* module to complete the sensor bindings and execute the skill, enabling seamless human–Misty interaction.

Table 2: Emotional attunement example by the *Script Writer*

Prompt: "Tell me the story of the Three Little Pigs"		
#	**Text**	**Emotion**
1	Three little pigs left home to build their own houses.	Neutral
2	The first pig quickly built a house of straw.	Contempt
3	The second pig put up a house of sticks with little effort.	Contempt
4	The third pig worked diligently to lay strong bricks for a sturdy home.	Happiness
	...	

***Load Task and Model*:** As in the SIA, this module uses an LLM with a schema-constrained prompt to convert user instructions into sensor- or skill-related commands, mapping natural-language intent to a predefined command set: `BIND`$\langle s, k\rangle$ binds skill k to sensor s; `UPDATE`$\langle s, k\rangle$ rebinds sensor s to a new skill k; `UNBIND`$\langle s\rangle$ clears all skills bound to sensor s; and `INVOKE`$\langle k\rangle$ runs skill k once without binding. Here $s \in S$ is one of Misty's ten sensors (Fig. 2, *Misty Sensors*), and $k \in K$ is a callable skill (e.g., "play relaxing piano music"). The skill set K is loaded at runtime for plug-and-play expansion: at startup, the *PIA* scans the local skill directory, parses each docstring into an inventory of skill names and capabilities, and injects this inventory into its system.

***Sensor & Tool Manager (STM)*:** *STM*, through a *Central Scheduler* module, maintains an independent worker process for each valid sensor and records their status in a process table (*sensor*, *PID*, *bound skill*, *Status*). The *Central Scheduler* performs periodic active checks, and any inactive process is immediately restarted, keeping sensor bindings available. Table 3 presents example entries of the *STM* process table. The fields *sensor*, *PID*, and *bound skill* denote the sensor name, process identifier, and the associated skill, respectively. The *Status* field indicates whether a sensor's PID exists in the system process tree (*Active*) or has terminated unexpectedly (*Inactive*).

Upon MistyPilot startup, *STM* scans the persisted process table, recovers the sensor-skill bindings, and automatically remounts them on Misty, providing persistent state and rapid recovery across sessions.

Table 3: Example of STM process table entries

Sensor	**PID**	**Bound Skill**	**Status**
Head Touch	1023	Greeting	Active
Scruff Touch	1098	DailyMemoReport	Active
Front Left Bumper	1120	NodHead	Inactive

5 Component-Level Evaluation Suites

To evaluate the MistyPilot framework, we curate five evaluation suites that target task routing, sensor binding, task-state parsing, retrieval reuse, and skill extensibility. Their construction follows a hybrid pipeline [32] that synthesizes realistic instructions while mitigating single-model bias: (1) we collect empirically grounded seeds from experts with substantial on-site Misty demonstration experience; (2) we expand these seeds using a multi-LLM ensemble (Gemini 2.5 Pro and GPT-5) under explicitly defined Easy/Hard criteria, with single-feature iterative prompting to control complexity; (3) we enforce diversity via ROUGE-L filtering, removing samples with overlap greater than 0.7; (4) we randomly sample from the instruction pool to form each suite; and (5) human annotators assign the ground-truth routing decisions, target skills, and parameter specifications. The LLM ensemble is used only to generate candidate instructions; all ground-truth labels are assigned by human annotators, enabling quantitative evaluation of MistyPilot's decision accuracy.

Route100: This suite comprises 100 task instructions designed to evaluate the routing capability of MistyPilot, specifically whether a task should be dispatched to the *SIA* for a dialogue-oriented task or to the *PIA* for sensor-triggered events or direct skill invocation. It is relatively balanced across both categories, with 58 *SIA* tasks and 42 *PIA* tasks. Each subset is further divided into *Easy* cases (direct and straightforward instructions) and *Hard* cases (implicit or composite conditions), enabling a fine-grained evaluation of routing accuracy under diverse natural-language inputs.

SensorBind40: This suite contains 40 single-turn instances spanning two categories: (i) sensor-binding commands and (ii) immediate single-skill invocation commands without sensor binding, curated to evaluate the *PIA*'s routing accuracy and sensor-binding capability. The instances are split by difficulty: 20 *Easy* cases that require either a single, immediate sensor-grounded response or the immediate invocation of a single skill (e.g., "touch your head and make a cute sound", or "I'm doing home exercise, play some good workout music for me"), and 20 *Hard* cases that require parsing and registering multiple concurrent sensor-trigger bindings within a single pass (e.g., "when I tap your chin, take a photo; press your forehead to say hi; touch your right side to show sadness"). These multi-binding instructions are challenging because all bindings must be correctly parsed, registered, and executed without omission.

TaskParser256: This suite contains 40 multi-turn dialogues (256 turns total) spanning multiple domains, including daily-life assistance, planning, storytelling, and emotion-supportive interaction. It evaluates the *SIA*'s proficiency in parsing user intent, dynamically managing task states, and executing system-level controls (e.g., switching to a more powerful model) based on explicit user feedback. The suite is split into 7 *Easy* dialogues (28 turns) and 33 *Hard* dialogues (228 turns). *Easy* dialogues contain clear instructions with no more than four turns, while *Hard* dialogues involve longer interactions with ambiguous references (coreference), topic shifts, and interleaved or evolving instructions, posing challenges for long-horizon reasoning and context-aware adaptation.

FastThinking230: This suite targets MistyPilot's *fast-thinking* module and evaluates its retrieval accuracy under paraphrasing. It contains 230 commands derived from 46 canonical tasks, each expanded into five variants that preserve the same core task while varying surface details. The goal is to measure whether the system can correctly retrieve and reuse previously executed implementations under surface-level linguistic variations, thereby reducing latency and computational cost.

SkillExtension100: This suite evaluates how well MistyPilot scales to newly introduced skills, assessing *PIA* skill extensibility under dynamically injected skills. New skills are introduced at four scales (30, 50, 70, and 100 skills), and each skill is provided solely by its docstring; the system is expected to select the appropriate skill based only on these docstring descriptions.

6 Experiments

To evaluate MistyPilot, we conduct experiments on *Task Routing* correctness, *PIA* performance, *SIA* performance, fast-thinking retrieval, and skill extensibility using the corresponding evaluation suites. To account for the stochastic nature of LLM outputs, each configuration is run five times with the temperature fixed at 0.3, and results are reported as mean ± standard deviation. Since MistyPilot's contribution lies in architectural role separation rather than prompt engineering alone, we compare it with a Single-Agent baseline that collapses the *Task Router*, *PIA*, *SIA*, and all skills into one LLM agent. For a fair comparison, the baseline uses the same GPT-5-mini backbone, skill set, inputs, and evaluation metrics as MistyPilot. Thus, the only difference is the system architecture: specialized agents with isolated contexts and restricted skill spaces versus a single agent with all context and skills collapsed together.

6.1 Evaluation of Task Routing Correctness

We evaluate MistyPilot's *Task Router* on the MistyPilot-Route100 suite by measuring routing accuracy across task categories and difficulty levels. Correct routing is critical because an incorrect agent assignment can directly lead to downstream execution failure. As shown in Table 4, the Multi-Agent design matches or outperforms the Single-Agent baseline across all categories, achieving 100% accuracy on SIA–Easy, SIA–Hard, and PIA–Easy, and showing higher and more stable performance on PIA–Hard (96.2%±5.66% vs. 90.4%±15.65%). These results show that both systems achieve strong routing performance on MistyPilot-Route100, while MistyPilot provides a clearer advantage in stability and accuracy, particularly on the more challenging PIA–Hard category. This suggests that architectural role separation can better support reliable routing than collapsing all functions into a single agent.

Table 4: Task routing correctness on Route100.

Condition	MistyPilot (Multi-Agent)	Single-Agent
SIA – Easy	$100\% \pm 0.00\%$	$99.2\% \pm 1.79\%$
SIA – Hard	$100\% \pm 0.00\%$	$100\% \pm 0.00\%$
PIA – Easy	$100\% \pm 0.00\%$	$100\% \pm 0.00\%$
PIA – Hard	$96.2\% \pm 5.66\%$	$90.4\% \pm 15.65\%$

Table 5: PIA performance on SensorBind40.

Subset	MistyPilot (Multi-Agent)	Single-Agent
Easy	$100.00\% \pm 0.00\%$	$100.00\% \pm 0.00\%$
Hard	$100.00\% \pm 0.00\%$	$81.00\% \pm 4.18\%$

6.2 Evaluation of PIA Performance

We evaluate the *PIA* on the MistyPilot-SensorBind40 suite, focusing on whether it can correctly establish dynamic sensor-skill bindings and execute direct skill invocations. We follow the same comparison setting as above. As shown in Table 5, both designs achieve 100% accuracy on the Easy subset, indicating that both can handle straightforward physical-interaction tasks. On the Hard subset, however, MistyPilot performs substantially better and more stably, reaching 100.00%±0% compared with 81.00%±4.18% for the Single-Agent baseline. These results show that while both systems perform well on simple cases, MistyPilot provides stronger reliability for more challenging sensor-skill binding and sensor-free skill invocation tasks.

6.3 Evaluation of SIA Performance

For the *SIA*, accurate *Task State* recognition is critical for multi-turn dialogue, since decisions such as `UPDATE` and `DELETE` directly affect downstream module coordination. We evaluate *Task State* and *System Control* correctness on the MistyPilot-TaskParser256 suite, using the same Single-Agent baseline for comparison. As shown in Table 6, both systems perform well, but MistyPilot achieves higher accuracy with substantially lower variance on both subsets: 99.29%±1.60% vs. 91.43%±13.27% on Easy, and 96.75%±1.15% vs. 93.50%±5.36% on Hard. These results indicate that MistyPilot provides more reliable *SIA* performance in long-horizon interactions involving ambiguous references, topic shifts, and evolving instructions.

6.4 Evaluation of Fast-Thinking Retrieval

Fast Thinking reuses user-saved results for semantically similar requests, retrieving prior implementations instead of regenerating them from scratch to reduce

Table 6: SIA performance on TaskParser256.

Subset	MistyPilot (Multi-Agent)	Single-Agent
Easy	$99.29\% \pm 1.60\%$	$91.43\% \pm 13.27\%$
Hard	$96.75\% \pm 1.15\%$	$93.50\% \pm 5.36\%$

Table 7: Fast-thinking retrieval on FastThinking230: Raw Text vs. Fast Path.

Data	Model	Top-1	Rank1 Dist.	Rank2 Dist.
Raw Text	TE3-L	67.83%	0.365 ± 0.070	0.464 ± 0.071
Raw Text	TE3-S	72.61%	0.342 ± 0.082	0.453 ± 0.066
Raw Text	MiniLM	58.70%	0.397 ± 0.127	0.570 ± 0.066
Fast Path	TE3-L	100.00%	0.003 ± 0.015	0.392 ± 0.216
Fast Path	TE3-S	100.00%	0.003 ± 0.015	0.394 ± 0.211
Fast Path	MiniLM	100.00%	0.000 ± 0.002	0.463 ± 0.248

Note: TE3-L = text-embedding-3-large; TE3-S = text-embedding-3-small; MiniLM = all-MiniLM-L6-v2.

redundant computation and improve response speed. We evaluate this capability on the MistyPilot-FastThinking230 suite. MistyPilot decomposes each task state into *main_task* and *details*, using the *main_task* representation for Fast Path retrieval. As an ablation, we compare this structured representation with a *Raw Text* baseline that retrieves directly from the original user input. We report three metrics: *Top-1 Accuracy*, *Rank1 Dist. Mean*, and *Rank2 Dist. Mean*, where the latter two measure the average embedding-space distance between the query and its top-1 and top-2 retrieved candidates. To test whether the effect is consistent across embedding spaces, we evaluate three embedding models: two proprietary models, `text-embedding-3-large` and `text-embedding-3-small`, and one open-source model, `all-MiniLM-L6-v2`.

As shown in Table 7, MistyPilot's disentangled representation achieves 100.00% Top-1 accuracy across all three embedding models, while the *Raw Text* baseline reaches only 67.83%, 72.61%, and 58.70%, respectively. The disentangled representation also yields a larger gap between *Rank1 Dist. Mean* and *Rank2 Dist. Mean*, indicating better separation between the correct match and the second-best candidate. In addition, the Fast-Thinking path reduces response time from 5.088 ± 2.571 s to 2.263 ± 0.627 s, corresponding to a 55.5% latency reduction.

6.5 Evaluation of Skill Extensibility

A key goal of MistyPilot is plug-and-play skill extension: new skills can be added to the library using only their docstring descriptions. We evaluate whether MistyPilot can still discover newly added skills and select the correct one as the

Table 8: Skill extensibility on SkillExtension.

Scale	30 skills	50 skills	70 skills	100 skills
Correct skill selection	✓	✓	✓	✓

library scales. Specifically, we test libraries with 30, 50, 70, and 100 skills on the SkillExtension suite, repeating each configuration five times. As shown in Table 8, MistyPilot selects the correct skill in all runs across all library sizes, indicating that docstring-based skill selection remains reliable for libraries of up to 100 skills. We observe no degradation within this range, while characterizing the upper scalability limit is left to future work.

Table 9: Subjective evaluation results of the MistyPilot prototype.

Evaluation Dimension and Questionnaire Item	Mean	Std
Interaction Naturalness: Communicating with the robot through natural language to perform specific tasks was intuitive and straightforward.	4.75	0.45
Comprehension Accuracy: The robot accurately understood my intentions and provided responses that met my expectations.	4.42	0.67
Emotional Expressiveness: The robot's speech and reactions were emotionally expressive, giving it a highly anthropomorphic and human-like presence.	4.75	0.45
System Responsiveness: The overall response latency was within an acceptable range.	4.67	0.49
User Satisfaction: Overall, I am satisfied with the multimodal interaction experience provided by the MistyPilot system.	4.75	0.45

Note: All scores are on a 5-point Likert scale (1 = strongly disagree, 5 = strongly agree). Mean is the average rating; Std is the standard deviation.

6.6 Preliminary User Feedback Study

As a preliminary assessment of practical feasibility and user perception, we conducted an in-person user study with 12 volunteers. Each participant interacted with MistyPilot for at least 30 minutes, totaling over 6 hours of interaction. Participants used SIA for open-domain conversations and PIA for 30 utility skills. Afterward, they completed a five-item post-study questionnaire (Table 9). The study was approved by our institution's IRB.

The questionnaire was adapted from established HRI and UX instruments and tailored to our setting [2]. Participants rated five aspects on a 5-point Likert scale. Results are consistently positive, with all mean scores above 4.4/5. Overall satisfaction, interaction naturalness, and emotional expressiveness received the highest ratings (Table 9). Occasional failures were mainly caused by ASR errors on accented speech, leading to misunderstood user intent.

7 Limitations and Future Work

Our evaluation suites target the components we consider most critical, but they are not exhaustive: they do not cover every interaction pattern a social robot may encounter, and designing broader evaluation suites, including composite and longer-horizon tasks, remains future work. In addition, sensor-skill bindings and user-marked results currently persist across sessions. While this supports continuity for individual users, it may raise privacy concerns on shared robots. Future work will introduce user authentication together with per-user isolation of persistent bindings and long-term memory.

8 Conclusion

We presented MistyPilot, a multi-agent LLM framework for natural-language skill orchestration on the Misty social robot. Its role-specialized architecture separates reactive physical interaction from stateful social interaction through the *PIA* and *SIA*. MistyPilot supports sensor-triggered skill execution, dialogue-oriented task-state management, result reuse, multimodal response generation, and extensibility to new skills – capabilities that existing interfaces expose only through hand-written code. Evaluations demonstrate higher component-level accuracy and lower variance than the single-agent baseline, while a preliminary user study indicates positive perceptions of usability and interaction quality.

9 Acknowledgment

This material is based upon work supported under the AI Research Institutes program by the U.S. National Science Foundation and the Institute of Education Sciences, U.S. Department of Education, through Award # 2229873—National AI Institute for Exceptional Education. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, the Institute of Education Sciences, or the U.S. Department of Education.

References

1. Ahn, M., Dwibedi, D., Finn, C., Arenas, M.G., Gopalakrishnan, K., Hausman, K., Ichter, B., Irpan, A., Joshi, N., Julian, R., et al.: Autort: Embodied foundation models for large scale orchestration of robotic agents. arXiv preprint arXiv:2401.12963 (2024)
2. Bartneck, C., Kulić, D., Croft, E., Zoghbi, S.: Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots. International journal of social robotics **1**(1), 71–81 (2009)
3. Chang, Y., Lo, K., Goyal, T., Iyyer, M.: Booookscore: A systematic exploration of book-length summarization in the era of llms. arXiv preprint arXiv:2310.00785 (2023)
4. Chen, S.C., Moyle, W., Jones, C., Petsky, H.: A social robot intervention on depression, loneliness, and quality of life for taiwanese older adults in long-term care. International psychogeriatrics **32**(8), 981–991 (2020)
5. Ciuffreda, I., Amabili, G., Casaccia, S., Benadduci, M., Margaritini, A., Maranesi, E., Marconi, F., De Masi, A., Alberts, J., de Koning, J., et al.: Design and development of a technological platform based on a sensorized social robot for supporting older adults and caregivers: Guardian ecosystem. International Journal of Social Robotics **17**(5), 803–822 (2025)
6. Dalgleish, T., Power, M.: Handbook of cognition and emotion. John Wiley & Sons (2000)
7. Eckert, A., Juvonen, P.: Programming social robots in linguistically heterogeneous classrooms: the case of misty. Digital Experiences in Mathematics Education pp. 1–20 (2025)
8. Evans, J.S.B.: Dual-processing accounts of reasoning, judgment, and social cognition. Annu. Rev. Psychol. **59**(1), 255–278 (2008)
9. Ghafurian, M., Chandra, S., Hutchinson, R., Lim, A., Baliyan, I., Rhim, J., Gupta, G., Aroyo, A.M., Rasouli, S., Dautenhahn, K.: Systematic review of social robots for health and wellbeing: a personal healthcare journey lens. ACM Transactions on Human-Robot Interaction **14**(1), 1–48 (2025)
10. González-Oliveras, P., Engwall, O., Wilde, A.: Social educational robotics and learning analytics: A scoping review of an emerging field. International Journal of Social Robotics pp. 1–16 (2025)
11. Goudbeek, M., Scherer, K.: Beyond arousal: Valence and potency/control cues in the vocal expression of emotion. The Journal of the Acoustical Society of America **128**(3), 1322–1336 (2010)
12. Jackson, J.N., Campbell, J.M.: Teachers' peer buddy selections for children with autism: Social characteristics and relationship with peer nominations. Journal of Autism and Developmental Disorders **39**(2), 269–277 (2009)
13. Kahneman, D.: Thinking, fast and slow. macmillan (2011)
14. Kling, M., Haeussl, A., Dalkner, N., Fellendorf, F.T., Lenger, M., Finner, A., Ilic, J., Smolak, I.S., Stojec, L., Zwigl, I., et al.: Social robots in adult psychiatry: a summary of utilisation and impact. Frontiers in psychiatry **16**, 1506776 (2025)
15. Li, M., Zhao, Y., Yu, B., Song, F., Li, H., Yu, H., Li, Z., Huang, F., Li, Y.: Api-bank: A comprehensive benchmark for tool-augmented llms. In: Proceedings of the 2023 conference on empirical methods in natural language processing. pp. 3102–3116 (2023)
16. Li, X.: A review of prominent paradigms for llm-based agents: Tool use, planning (including rag), and feedback learning. In: Proceedings of the 31st International Conference on Computational Linguistics. pp. 9760–9779 (2025)

17. Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the middle: How language models use long contexts. Transactions of the association for computational linguistics **12**, 157–173 (2024)
18. Liu, S., Li, Y., Zhang, K., Cui, Z., Fang, W., Zheng, Y., Zheng, T., Song, M.: Odyssey: Empowering minecraft agents with open-world skills. arXiv preprint arXiv:2407.15325 (2024)
19. Maynez, J., Narayan, S., Bohnet, B., McDonald, R.: On faithfulness and factuality in abstractive summarization. arXiv preprint arXiv:2005.00661 (2020)
20. Ocker, F., Tanneberg, D., Eggert, J., Gienger, M.: Tulip agent–enabling llm-based agents to solve tasks using large tool libraries. arXiv preprint arXiv:2407.21778 (2024)
21. Pan, Q., Ji, W., Ding, Y., Li, J., Chen, S., Wang, J., Zhou, J., Chen, Q., Zhang, M., Wu, Y., et al.: A survey of slow thinking-based reasoning llms using reinforced learning and inference-time scaling law. arXiv preprint arXiv:2505.02665 (2025)
22. Patil, S.G., Zhang, T., Wang, X., Gonzalez, J.E.: Gorilla: Large language model connected with massive apis. Advances in Neural Information Processing Systems **37**, 126544–126565 (2024)
23. Peysakhovich, A., Lerer, A.: Attention sorting combats recency bias in long context language models. arXiv preprint arXiv:2310.01427 (2023)
24. Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., et al.: Toolllm: Facilitating large language models to master 16000+ real-world apis. arXiv preprint arXiv:2307.16789 (2023)
25. Qu, C., Dai, S., Wei, X., Cai, H., Wang, S., Yin, D., Xu, J., Wen, J.R.: Tool learning with large language models: A survey. Frontiers of Computer Science **19**(8), 198343 (2025)
26. Sawik, B., Tobis, S., Baum, E., Suwalska, A., Kropińska, S., Stachnik, K., Pérez-Bernabeu, E., Cildoz, M., Agustin, A., Wieczorowska-Tobis, K.: Robots for elderly care: review, multi-criteria optimization model and qualitative case study. In: Healthcare. vol. 11, p. 1286. MDPI (2023)
27. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems **36**, 68539–68551 (2023)
28. Tang, Q., Deng, Z., Lin, H., Han, X., Liang, Q., Cao, B., Sun, L.: Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. arXiv preprint arXiv:2306.05301 (2023)
29. Romero-C. de Vaca, A.J., Melendez-Armenta, R.A., Ponce, H.: Using social robotics to identify educational behavior: A survey. Electronics **13**(19), 3956 (2024)
30. Wang, X., Dong, L., Rangasrinivasan, S., Nwogu, I., Setlur, S., Govindaraju, V.: 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). pp. 9194–9201. IEEE (2025)
31. Wang, X., Wang, Y., Han, Z., Duan, Z., Zhang, Z., Alsudais, T.A.: Social robots for child development: research hotspots, topic modeling, and collaborations. Humanities and Social Sciences Communications **12**(1), 1–15 (2025)
32. Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N.A., Khashabi, D., Hajishirzi, H.: Self-instruct: Aligning language models with self-generated instructions. In: Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers). pp. 13484–13508 (2023)

33. Wang, Z., Cheng, Z., Zhu, H., Fried, D., Neubig, G.: What are tools anyway? a survey from the language model perspective. arXiv preprint arXiv:2403.15452 (2024)
34. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. In: International Conference on Learning Representations (ICLR) (2023)
35. Zhao, I.Y., Leung, A.Y.M., Huang, Y., Liu, Y.: A social robot in home care: Acceptability and utility among community-dwelling older adults. Innovation in Aging **9**(5) (2025)