LLM-Driven Adaptive Recovery for Multi-Agent Systems in Dynamic Environments

Beril Yalcinkaya [1,2,*], Carlos A. P. Pizzino [2], Micael Couceiro [2], Salviano Soares [1] and António Valente [1]

School of Sciences and Technology-Engineering Department, University of Trás-os-Montes and Alto Douro (UTAD), Portugal
[2]CORE R&D Department, Ingeniarius, Lda., Portugal

beril@ingeniarius.pt, carlos@ingeniarius.pt, micael@ingeniarius.pt, salblues@utad.pt, avalente@utad.pt

Abstract -

Resilience to unexpected failures is critical for multi-agent collaboration in dynamic construction environments. Existing recovery strategies often rely on rigid contingency rules or computationally expensive replanning, which struggle with heterogeneous fleets and complex dependencies. We propose an adaptive recovery framework leveraging Large Language Models (LLMs) to generate executable recovery plans by reasoning over structured task graphs and real-time agent states. The system dynamically adjusts to failures by reassigning subtasks, involving humans, or modifying synchronization. We validate the framework through (i) offline evaluation of 120 plans against seven executability criteria, and (ii) field demonstrations with a heavy-duty autonomous loader and human collaborators. Results demonstrate that the framework produces executable strategies preserving task dependencies, proving the feasibility of LLM-driven resilience in industrial fleet management.

Keywords -

Instructions; Formatting; Authors; ISARC Large Language Models; Autonomous Systems; Adaptive Recovery

1 Introduction

Hybrid multi-agent systems, where autonomous robots and human collaborators coordinate to achieve complex goals, are increasingly prevalent in dynamic domains such as logistics and construction [1, 2, 3]. A defining characteristic of these missions is that tasks are rarely atomic; they consist of interdependent subtasks distributed across agents with heterogeneous capabilities. Consequently, a failure in a single agent or subtask—such as a robot encountering an obstacle—can propagate through the dependency graph, rendering subsequent actions infeasible and stalling the entire operation.

Traditional recovery strategies typically rely on predefined contingency plans or static replanning, which often fail in dynamic scenarios due to their inability to reason about real-time constraints and complex inter-agent dependencies [4]. To address these limitations, we propose an adaptive recovery framework leveraging Large Language Models (LLMs). By exploiting the logical reasoning and generalization capabilities of LLMs [5, 6], our approach interprets real-time execution feedback to generate context-aware recovery strategies. Unlike static rule-based methods, our framework dynamically reasons about task structures and agent availability, including human intervention, to ensure mission continuity in uncertain environments.

Our key contributions include a recovery framework that integrates an LLM within a fleet manager to reason over task dependencies, agent capabilities, and failure feedback, supported by a structured JSON interface that encodes the full execution context for robust prompting. We further propose a methodology for the safe interruption and reassignment of tasks that preserves synchronization and inter-agent dependencies during recovery. Finally, we provide an empirical evaluation across four representative scenarios, comprising an offline validation of 120 LLM-generated recovery plans and real-world demonstrations involving human collaborators and an AGV retrofitted from a heavy-duty loader for autonomous pallet transportation in construction settings.

2 Related Work

Classical multi-agent recovery relies heavily on predefined contingency plans or rule-based strategies, where fallback procedures are explicitly encoded [7]. These systems typically use deterministic logic [4, 8], redundancy where peers take over tasks [9], or market-based auctions for dynamic reassignment [10]. While effective in structured environments, these methods struggle with flexibility and scalability in heterogeneous scenarios, often lacking a machine-parseable global task graph to handle complex dependencies.

To address this rigidity, learning-based approaches leverage reinforcement learning [11] or graph neural networks [12] to model adaptive coordination without explicit rules. However, these methods typically require extensive

training data and reward shaping. In field robotics and construction, the scarcity of failure data and the safety-critical nature of heavy machinery make such "trial-and-error" learning impractical, constraining purely learning-based solutions.

Large Language Models (LLMs) have enabled flexible robotic planning by reasoning over symbolic structures [5]. Approaches generally act as *Scorers*, evaluating actions based on affordances [6, 13], or *Generators*, producing structured plans from descriptions [14, 15]. For instance, SayCan [6] grounds LLM outputs in physical feasibility but primarily focuses on initial planning rather than mid-execution recovery.

Recent works have begun to address execution failures. Inner Monologue [16] uses closed-loop feedback for reasoning, though it may overlook critical constraints. ProgPrompt [17] incorporates assertion checks for verification but remains fixed at execution time. Text2Reaction [18] advances this by enabling plan regeneration upon environmental changes. Crucially, however, these methods are largely framed for single-agent contexts. They do not model cross-agent synchronization (e.g., barriers) or heterogeneous roles. In contrast, our framework reasons over a centralized, mission-level DAG spanning humans and robots, enabling the fleet manager to handle interruptions and reassignments while strictly preserving global causal structures.

3 Background: Fleet Management System

The fleet management system (FMS), built on the ROOSTER framework[1], coordinates heterogeneous teams of robots and humans using unified ROS Actions. The architecture is organized into four layers: **ROS communication** provides standardized topics, services, and actions for all agents; **Agent abstraction** represents each robot or human as a Mobile Executor (MEx) with a common status message (e.g., pose, battery); **Task management** handles task decomposition, dispatching, safe reassignment, and explicit synchronization primitives (events/barriers) to enforce inter-agent ordering; and **User interface** enables operators to inject tasks, monitor execution, and participate directly via graphical dashboards and mobile apps.

In our setup, the robotic agent is a robotized heavy-duty loader, equipped with three ROS Action Servers - `GoTo`, `LoadPallets`, and `UnloadPallets` - for navigation and manipulation. Each Action Server follows the standard ROS interface, exposing both *feedback* and *result* channels. Feedback messages provide real-time progress information (e.g., percentage of action completion), while result messages report the outcome as `SUCCEEDED`, `FAILED`, or `ABORTED`, together with an error string. This structure allows FMS to continuously monitor execution and detect failures at the action level. Humans are integrated through a Unity-based mobile app, which publishes position data and exposes a `HumanInteraction` action server. This allows FMS to issue long-running interaction goals (e.g., clear obstacle) that remain active until the user explicitly confirms completion.

FMS has already been validated in previous EU-funded deployments (details omitted for anonymity), where human agents collaborated with robotic agents (lightweight reconnaissance drones and heavy-lift transport drones) under FMS's coordination. Prior to this work, however, recovery from failures in such missions remained limited: either the operator had to manually terminate the task, or the failed action aborted execution internally.

In this work, we extend FMS with an LLM-driven adaptive recovery framework that dynamically reassigns or modifies tasks in response to failures, injecting recovery without restarting the system or pausing unaffected agents, thereby ensuring continuity in hybrid multi-agent missions.

4 METHODOLOGY

The proposed methodology extends the FMS with a centralized LLM-driven recovery loop (Fig. 1). The framework operates in four stages: (i) agents execute their subtasks defined as Python functions, which are monitored through ROS action servers; (ii) failures detected during execution trigger the *Replan* module, which collects the current task code, agent states, and dependency graph; (iii) the collected information is encoded into a structured JSON prompt for the LLM, which analyzes the context and generates updated recovery functions and coordination primitives; and (iv) the LLM's output (recovery code) is parsed and seamlessly integrated back into the ongoing mission.

This architecture allows failures to be handled adaptively without aborting the entire mission or restarting unaffected agents. Instead, the LLM selectively reasons about which agents are impacted, introduces or preserves synchronization events when necessary, and revises only the parts of the task graph relevant to recovery.

4.1 Task Structure and Failure Monitoring

In the fleet manager, tasks are encoded as Python functions that decompose into subtasks for each participating agent, which may execute in parallel. Each agent is assigned a dedicated function that specifies its sequence of actions. For instance, a loader robot may perform navigation and manipulation actions such as `GoTo`, `Load`, `GoTo`, and `Unload`, while a human agent may contribute with actions such as `OpenBox` or `PrepareBox`. Although defined independently, these functions are synchronized through

[1] https://github.com/ROOSTER-fleet-management

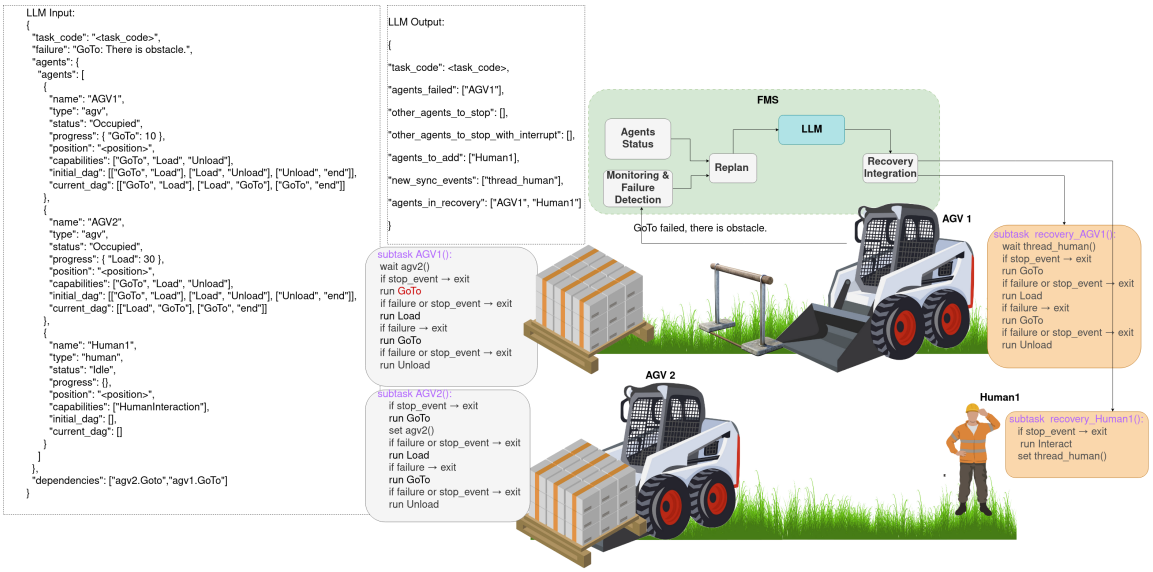

Figure 1. Architecture of the proposed LLM-driven recovery framework in FMS.

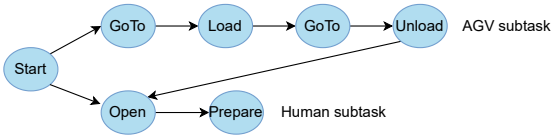

Figure 2. DAG representation of an example hybrid task involving a loader and a human agent.

events to ensure correct ordering. For example, a human may wait for the loader to complete loading before proceeding to open a box.

Both sequential dependencies within an agent and cross-agent interdependencies are captured in a directed acyclic graph (DAG), which provides the structural basis for LLM-driven recovery planning (Fig. 2). The DAG is initialized with all actions as nodes and all dependencies as edges, and is incrementally updated as actions succeed, with completed nodes and their outgoing edges removed.

The execution layer also establishes a uniform monitoring pattern that is later reused for recovery. Each action is wrapped with three control signals:

- **stop_event** — halts an agent when the LLM provides a new recovery plan.
- **failure_event** — triggered when an action fails and the agent cannot continue.
- **action_interrupt** — allows an action to be preempted immediately if required by recovery.

Each subtask follows a simple, reusable template, abstracted here:

Listing 1. Generic action execution pattern with monitoring signals.

```
if stop_event.is_set():
return
if not run_action_with_interrupt(agent_type,
    agent_name, action,
stop_event, failure_event, action_interrupt):
return
```

In our implementation, run_action_with_interrupt executes the bound ROS action and returns `False` if the action reports `FAILED/ABORTED`, or if action_interrupt is raised by the recovery module. Pending actions are maintained in the agent's local queue and are not discarded. Upon successful recovery, the agent resumes at the next node consistent with the updated DAG.

4.2 Replan Module and LLM Input

When an action executed by an agent fails, the **Replan** module is invoked. It gathers the latest agent statuses—including availability, position, and feedback on progress percentage on the running action—along with task information, and packages them into a structured JSON message for the LLM. This message follows a strict snake_case schema with required and optional fields and include integrity checks against the current DAG. Positions are provided in the map frame; when helpful, FMS also injects precomputed distances to the failure site to make proximity comparisons explicit. By consolidating execution context into a structured schema, the LLM can

reason consistently about task dependencies, agent availability, and failure scope. An overview of the input schema is shown below:

Listing 2. Structured JSON schema provided by the Replan module to the LLM.

```
{
  "task_code": "<current_task_code>",
  "failure": "<
      failure_message_from_ROS_action_result>",
  "agents": [
    {
      "name": "<agent_name>",
      "type": "<agent_type>",
      "status": "Occupied/Idle",
      "progress": {"<current_action>": "<
          percentage>"},
      "position": "<position>",
      "capabilities": ["<
          list_of_available_actions>"],
      "initial_dag": "<task_DAG_at_start>",
      "current_dag": "<updated_task_DAG>"
    },
    ...
  ],
  "dependencies": ["<
      array_of_interdependent_nodes>"]
}
```

The LLM backend was accessed through the OpenAI API, enabling programmatic control of inputs and outputs rather than relying on interactive chat sessions. This allowed consistent formatting of recovery prompts, reproducibility across multiple runs, and easier integration with the fleet manager.

4.3 LLM Prompting and Output Design

To transform structured execution data into actionable recovery plans, we design a constrained prompting strategy for the LLM. The key challenge is to ensure that the model's output is both logically grounded in the task state and directly executable within FMS.

4.3.1 Prompt design

The LLM is framed as an adaptive multi-agent planner. Rather than directly requesting code, the prompt enforces a two-stage reasoning process. First, the model explicitly analyzes the failure context by answering targeted assessment questions:

- Is the failure localized to a single agent, or does it require multi-agent coordination?
- Should the strategy be retry, revise, or reassign?
- Which agents must be stopped or interrupted?
- Which actions have already succeeded and must not be repeated?
- How should synchronization events (e.g., thread signals) be preserved?

This step acts as a lightweight chain-of-thought mechanism, forcing the model to reason over the task DAG and agent states before producing a solution.

4.3.2 Output specification

After reasoning, the model generates a structured JSON object containing executable recovery functions in Python syntax and metadata fields describing the context. Key fields include: **task_code** python-like recovery functions specifying revised action sequences; **agents_failed** the agent(s) whose action triggered recovery; **other_agents_to_stop** and **other_agents_to_stop_with_interrupt** distinguishing between graceful and immediate termination of peer agents; **agents_to_add** for newly recruited agents; **new_sync_events** for synchronization; and **agents_in_recovery** listing all participants in the plan.

This structured format ensures that recovery plans are machine-parseable and can be integrated back into the fleet manager without manual post-processing, as detailed in the following section.

4.4 Recovery Execution in FMS

The final stage of the recovery loop is the validated integration of the LLM's output into the execution layer of the FMS. Upon receiving a recovery JSON, the system (i) parses and validates the metadata fields to update the status of running agents; (ii) reconciles the proposed `task_code` with the live DAG; and (iii) applies agent-level control according to the stop/interrupt policy. Agents listed in `agents_failed` or `other_agents_to_stop` are halted gracefully after finishing their current action before switching to their assigned recovery subtasks, whereas agents in `other_agents_to_stop_with_interrupt` are preempted immediately. New synchronization primitives (e.g., `thread_human`) in `new_sync_events` are instantiated when required, ensuring that modified task graphs preserve inter-agent dependencies.

To preserve synchronization and human factors, we use conservative defaults: (a) never interrupt human tasks unless the plan explicitly flags a safety/resource conflict; (b) if a human task's progress exceeds a threshold of 80%, prefer stop (wait-to-complete) over interrupt; (c) interrupt robots only when the recovery would otherwise deadlock or violate resource exclusivity (e.g., shared tool). These rules emerged from our offline/real-world trials and reduce DAG regressions by discouraging unnecessary preemption of ongoing human work.

The recovery functions in `task_code` follow the same Pythonic structure as the original plan and define agent-specific routines, such as `loader_recovery()` or `human1_recovery()`. Each affected or newly recruited

agent is assigned a dedicated thread, launched once the agent becomes available. Pending actions are reconciled with the updated DAG by removing satisfied predecessors and enqueueing only the next valid successors; this ensures that already-completed work is not repeated and that inter-agent dependencies remain intact. New synchronization primitives (events/barriers) are instantiated where the plan adds hand-offs or waits.

Recovery is committed only after schema validation and a dry-run compilation of `task_code`; otherwise, the system re-prompts the LLM with a compact error report. This tight loop enables seamless injection of recovery plans into an ongoing mission-without system restarts and without pausing unaffected agents, while maintaining global causal structure.

5 Experimental Study

We evaluate the proposed recovery framework through two complementary studies. First, we conduct an *offline plan-generation validation*, where the LLM is queried repeatedly under controlled task failures and its outputs are systematically assessed against seven criteria. This analysis quantifies the reliability and limitations of the LLM as a recovery planner. Second, we present a *real-world use case study*, where representative recovery plans are injected into the FMS and executed on an AGV loader in collaboration with a human operator. In the offline study, the LLM consumes structured JSON contexts and returns code that is not executed on hardware; in the real-world study, selected plans are validated and injected into the live FMS to drive actual execution.

5.1 Experimental Scenarios

To evaluate the recovery framework in a controlled manner, we designed four representative scenarios that capture different configurations of agents. In all cases, the AGV loader's nominal task consisted of three sequential actions: `GoTo` → `LoadPallets` → `UnloadPallets`. Failures were consistently induced at the `LoadPallets` step, with the failure message: *"LoadPallets: The robot is stuck because there is an obstacle."*. We fixed the failure point to standardize comparisons across scenarios; varying failure locations is left for future work.

- **Exp 1: Single human idle.** One loader robot and one idle human agent are present. The loader executes its nominal delivery task, while the human is initially unassigned.
- **Exp 2: Two humans idle.** One loader robot and two idle human agents are present. Both are initially unassigned, but differ in proximity to the loader.
- **Exp 3: One human busy, one idle.** One loader robot and two human agents are present. The closer human is already engaged in another subtask (e.g., `AuxTask`) with 10% progress, while the further human is idle and available.
- **Exp 4: Two loaders and one busy human.** Two loader robots and one human agent are present. Both loaders follow the nominal delivery sequence, but Loader 2 depends on Loader 1's completion of `UnloadPallets`. The human agent is initially occupied with another subtask with 10% progress.

Recovery plans were generated using OpenAI's GPT models accessed via the API. Specifically, we employed the `gpt-4.1` model, which supports structured chat-based prompting and code generation. The model was queried with carefully designed system and user messages encoding the recovery instructions and the current task state provided in JSON. To ensure deterministic outputs, we fixed the sampling parameters to `temperature=0.0` and `top_p=0.1`, which minimizes randomness while still allowing limited exploration of alternatives. We also set `max_completion_tokens=2048` to allow full recovery functions to be generated without truncation.

5.2 Evaluation Metrics

To assess the quality of LLM-generated recovery plans in the offline validation study, we define seven binary criteria: **Capability adherence (Cap)** ensures the LLM selects only valid actions from each agent's capability set; **Cross-agent isolation (Iso)** verifies that actions are assigned to the correct agent subtask (e.g., preventing robotic actions like `GoTo` in human subtasks); **DAG preservation (DAG)** respects the causal structure to avoid unnecessary repetition (e.g., resuming from failure rather than restarting); **Synchronization correctness (Sync)** preserves thread events for inter-agent coordination; **Stop/interrupt logic (Stop)** handles agent termination and resumption consistently; **Agent selection accuracy (Sel)** assigns recovery roles based on availability and proximity; and **Executability (Exec)** requires the output to be syntactically valid, directly executable Python code.

These metrics extend beyond standard executability checks used in prior plan-generation benchmarks (e.g., [19, 6]) by explicitly addressing multi-agent dependencies and synchronization, which are critical in hybrid human–robot tasks. We additionally verify that recovery code references only declared agents and that all referenced synchronization events exist in the current DAG.

5.3 Evaluation Procedure

Each criterion was assessed across 120 independent recovery plans using a hybrid approach: **Automatic checks** verified Capability adherence (Cap), Cross-agent isolation (Iso), and Executability (Exec) via syntax parsing; while **Manual inspection** was conducted by two annotators for DAG preservation, Synchronization, Stop/interrupt logic, and Selection accuracy. Due to the strictly objective and binary nature of these criteria, a cross-verification of a representative subset yielded 100% agreement, confirming high annotation reliability against the ground-truth task DAG and JSON context.

5.4 Results

Table 1 summarizes the outcomes of the offline validation across 30 LLM generations for each scenario. Results are reported as success rates for the seven evaluation metrics, along with an overall Success Rate (SR) capturing the fraction of plans that satisfied all criteria simultaneously. With 30 trials per scenario, 95% confidence intervals ranged between ±7–15%, reflecting the limited sample size. This suggests the framework is generally reliable, though Exp 4 shows more variability due to its higher complexity.

Exp 1: Single human idle. The LLM consistently assigned the idle human to remove the obstacle via `HumanInteraction`, preserved synchronization, and resumed the loader correctly from `LoadPallets` → `UnloadPallets`. Plans were executable with no invalid actions. A few times, the LLM misclassified fields in the JSON metadata (e.g., incorrectly marking the failed loader for interruption). Although the recovery logic itself remained valid, such schema errors were treated as execution failures in our evaluation.

Exp 2: Two humans idle. The LLM always selected the closer human to assist, demonstrating correct handling of agent proximity. As in Exp 1, task continuation from the correct DAG state and synchronization were consistently satisfied. Occasional inconsistencies appeared in the handling of the failed loader in the JSON fields (e.g., occasionally flagged for unnecessary stop/interrupt), highlighting the sensitivity of execution to schema correctness.

Exp 3: One human busy, one idle. The model reliably chose the further but idle human, never attempting to stop or interrupt the closer but busy agent. Plans correctly preserved DAG progress, ensured synchronization, and respected agent isolation. As in previous scenarios, in a few cases, the JSON metadata unnecessarily flagged the loader as interrupted.

Exp 4: Two loaders and one busy human. The LLM consistently assigned the human to support recovery but, in some trials, chose to interrupt the ongoing human subtask instead of allowing it to complete first. Although such interruption may be operationally acceptable when progress is low, it exposed a limitation of the current recovery validation pipeline: in 5 out of 30 trials, the interrupted human task was not explicitly reinserted into the updated DAG after the recovery assistance was completed, causing a DAG-preservation failure. This issue did not affect action validity or synchronization correctness, but it highlights the need for stronger guarantees on human-task resumption in future versions of the framework. Importantly, Loader 2 remained unaffected by the failure: it was not stopped and continued executing its own task. Since Loader 2's completion depended on Loader 1 unloading, the LLM correctly preserved the corresponding synchronization event, ensuring that inter-loader dependencies were respected. Action validity, synchronization, and syntax were preserved.

To further illustrate the effect of LLM-driven recovery, Fig. 3 shows the task DAG for Exp 4 before and after recovery. The original plan (blue) encodes both loaders' nominal subtasks, with a dependency edge requiring Loader 2 to wait until Loader 1 completes `UnloadPallets`. The failure at Loader 1's `LoadPallets` step is highlighted in red. The recovery module stops the busy human agent (orange) and adds a new intervention action (green) to remove the obstacle before resuming Loader 1's sequence. Importantly, the dependency for Loader 2 remains intact: the LLM preserves the synchronization event, ensuring that Loader 2 continues its plan unaffected until Loader 1 completes unloading. This visualization highlights how recovery selectively modifies the DAG while respecting inter-agent constraints.

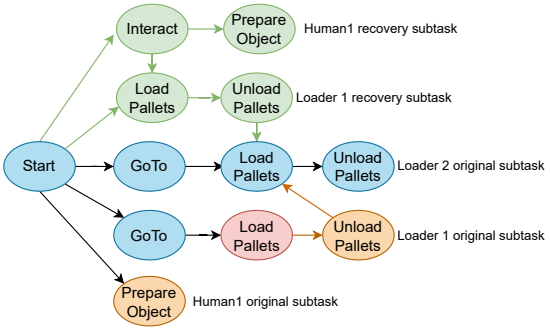

Figure 3. DAG representation of Exp 4. The figure illustrates how the LLM modifies only the affected parts of the task graph while preserving inter-agent dependencies (e.g., Loader 2 waits for Loader 1 to unload).

Table 1. Offline validation results over 30 LLM generations per scenario. Values are success rates (%) with 95% confidence intervals. SR = overall success rate.

Exp.	Cap	Iso	DAG	Sync	Stop	Sel	Exec	SR	95% CI
1	1.00	1.00	1.00	1.00	0.96	1.00	1.00	93%	77–99
2	1.00	1.00	1.00	1.00	0.93	1.00	1.00	90%	73–97
3	1.00	1.00	1.00	1.00	0.96	1.00	1.00	96%	82–99
4	1.00	1.00	0.83	1.00	0.96	1.00	1.00	83%	66–93

Across all scenarios, plan generation times (in Fig. 4) were practical for online use, with medians of 8–10 s and averages under 15 s. Occasional outliers reached 30 s, but this remains shorter than typical human interventions or task execution times.

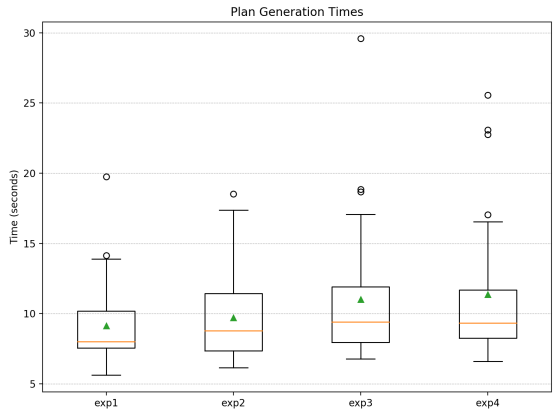

Figure 4. Distribution of LLM response times across the four experimental scenarios (30 trials each).

5.5 Real-World Demonstration

To assess the executability of LLM-generated recovery plans within FMS, we conducted hardware trials with an AGV loader robot. The loader was equipped with ROS Action Servers for `GoTo`, `LoadPallets`, and `UnloadPallets`. Its nominal mission was to navigate to a loading point, load a tank, transport it, and unload at a designated location (Fig. 5).

Figure 5. Nominal mission of the AGV loader: navigate to the loading point, load a tank, transport it, and unload at the destination.

In these trials, failures were induced at the `GoTo` step by placing two boxes directly in the robot's path. The ROS Action Server reported the failure as: *"GoTo: The robot is stuck due to target itself being inside an obstacle."* This triggered the recovery loop.

The four scenarios from the offline validation were replicated in the real system. Additional agents were emulated when needed: a second loader (Exp4) and a second human (Exp2 and Exp 3) were simulated by publishing status and action feedback to FMS. This ensured comparable experimental conditions across both studies.

Across all scenarios, the plans met the success criterion of completing the mission without human override or system restart. The loader successfully resumed its tasks after human assistance was requested through the mobile app, and unaffected agents continued execution without interruption. The only deviation from the offline validation occurred in Exp 4: while the LLM often interrupted the busy human in simulation, in the real-world execution the human was reassigned only after finishing the ongoing subtask (90% progress). Once available, the human cleared the obstacle, and the loader completed its mission as intended.

6 CONCLUSIONS AND FUTURE WORK

This paper introduced an LLM-driven recovery framework for hybrid human–robot teams, enhancing fleet management with adaptive fault handling. Evaluations across 120 offline plans and real-world trials demonstrated that the LLM consistently generates executable, capability-adherent strategies while maintaining agent synchronization. While effective, results from complex multi-agent scenarios suggest that future iterations should include explicit completion guarantees for preempted human tasks. Next steps include exploring broader failure modes, larger teams, and more intricate task graphs across diverse construction environments.

Acknowledgment

This work was supported by the FORTIS project, funded by the European Union's Horizon Europe (GA No. 101135707).

References

[1] Zhe Chen, Javier Alonso-Mora, Xiaoshan Bai, Daniel D Harabor, and Peter J Stuckey. Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. *IEEE Robotics and Automation Letters*, 6(3):5816–5823, 2021.

[2] Jorge Pena Queralta, Jussi Taipalmaa, Bilge Can Pullinen, Victor Kathan Sarker, Tuan Nguyen Gia,

Hannu Tenhunen, Moncef Gabbouj, Jenni Raitoharju, and Tomi Westerlund. Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision. *Ieee Access*, 8:191617–191643, 2020.

[3] Liqun Xiang, Yongtao Tan, Geoffrey Shen, and Xin Jin. Applications of multi-agent systems from the perspective of construction management: A literature review. *Engineering, Construction and Architectural Management*, 29(9):3288–3310, 2022.

[4] Antonín Komenda, Peter Novák, and Michal Pěchouček. Decentralized multi-agent plan repair in dynamic environments. *arXiv preprint arXiv:1202.2773*, 2012.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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

[7] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Online replanning in belief space for partially observable task and motion problems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5678–5684. IEEE, 2020.

[8] Edmund H Durfee and Victor R Lesser. Using partial global plans to coordinate distributed problem solvers. In *Readings in distributed artificial intelligence*, pages 285–293. Elsevier, 1988.

[9] Sergey Bereg, Luis Evaristo Caraballo, José Miguel Díaz-Báñez, and Mario Alberto López. Resilience of a synchronized multi-agent system. *arXiv preprint arXiv:1604.08804*, 2016.

[10] M Bernardine Dias and Anthony Stentz. A market approach to multirobot coordination. Technical report, 2000.

[11] Brijen Thananjeyan, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg. Recovery rl: Safe reinforcement learning with learned recovery zones. *IEEE Robotics and Automation Letters*, 6(3):4915–4922, 2021.

[12] Anthony Goeckner, Yueyuan Sui, Nicolas Martinet, Xinliang Li, and Qi Zhu. Graph neural network-based multi-agent reinforcement learning for resilient distributed coordination of multi-robot systems. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5732–5739. IEEE, 2024.

[13] Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, et al. Grounded decoding: Guiding text generation with grounded models for embodied agents. *Advances in Neural Information Processing Systems*, 36:59636–59661, 2023.

[14] Ishita Dasgupta, Christine Kaeser-Chen, Kenneth Marino, Arun Ahuja, Sheila Babayan, Felix Hill, and Rob Fergus. Collaborating with language models for embodied reasoning. *arXiv preprint arXiv:2302.00763*, 2023.

[15] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.

[16] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[17] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: program generation for situated robot task planning using large language models. *Autonomous Robots*, 47(8):999–1012, 2023.

[18] Zejun Yang, Li Ning, Haitao Wang, Tianyu Jiang, Shaolin Zhang, Shaowei Cui, Hao Jiang, Chunpeng Li, Shuo Wang, and Zhaoqi Wang. Text2reaction: Enabling reactive task planning using large language models. *IEEE Robotics and Automation Letters*, 9 (5):4003–4010, 2024.

[19] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022.