A Unified Framework for Real-Time Failure Handling in Robotics Using Vision-Language Models, Reactive Planner and Behavior Trees

Faseeh Ahmad*[1], Hashim Ismail*[1], Jonathan Styrud[2,3], Maj Stenmark[1] and Volker Krueger[1]

***Abstract*— Robotic systems often face execution failures due to unexpected obstacles, sensor errors, or environmental changes. Traditional failure recovery methods rely on predefined strategies or human intervention, making them less adaptable. This paper presents a unified failure recovery framework that combines Vision-Language Models (VLMs), a reactive planner, and Behavior Trees (BTs) to enable real-time failure handling. Our approach includes pre-execution verification, which checks for potential failures before execution, and reactive failure handling, which detects and corrects failures during execution by verifying existing BT conditions, adding missing preconditions and, when necessary, generating new skills. The framework uses a scene graph for structured environmental perception and an execution history for continuous monitoring, enabling context-aware and adaptive failure handling. We evaluate our framework through real-world experiments with an ABB YuMi robot on tasks like peg insertion, object sorting, and drawer placement, as well as in AI2-THOR simulator. Compared to using pre-execution and reactive methods separately, our approach achieves higher task success rates and greater adaptability. Ablation studies highlight the importance of VLM-based reasoning, structured scene representation, and execution history tracking for effective failure recovery in robotics.**

I. Introduction

Modern robotic systems excel in controlled environments, but struggle with dynamic environments such as small batch manufacturing, particularly in handling execution failures [1]. Failures such as unexpected obstacles, sensor inaccuracies, or misaligned objects disrupt operations, causing costly delays [2]. Unlike repetitive, pre-planned tasks in large-scale production, small batch manufacturing demands adaptability to frequent task variations. Similarly, in collaborative assembly lines, where robots work alongside humans, real time failure handling is crucial for safe and efficient execution [3]. Developing autonomous failure recovery mechanisms that enable robots to detect, identify, and correct failures without human intervention is essential for improving reliability and reducing downtime [4].

To address these challenges, failure recovery methods range from learning based approaches that rely on data driven policies to structured execution frameworks designed for modular and interpretable decision making. Many learning based methods employ end-to-end architectures where robotic control policies are trained directly from data [5], [6]. While effective across diverse tasks, these methods often lack interpretability and verifiability, making them unsuitable for safety critical domains requiring robust, failure resistant execution especially in high stakes environments where errors can damage expensive equipment or disrupt operations.

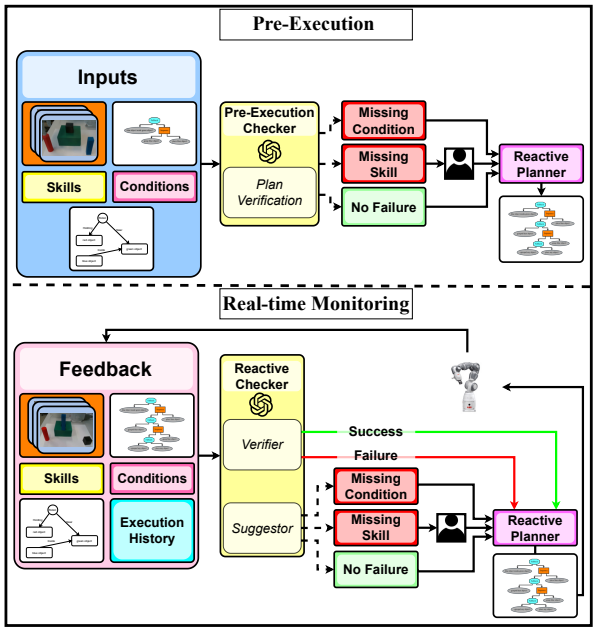

Fig. 1. Overview of our approach, which consists of two phases: pre-execution verification and real-time monitoring. The pre-execution phase verifies the entire planned BT proactively using a VLM based on inputs (images, scene graphs, skills, and conditions). The real-time phase continuously monitors execution, where the VLM verifies preconditions, postconditions, and infers missing preconditions for individual skills using updated inputs and execution history. A reactive planner dynamically generates and adapts the BT as the robot's execution policy.

Structured execution frameworks, such as Behavior Trees (BTs) [7], provide a modular framework for verification, adaptation, and efficient failure recovery. They define execution policies as hierarchical compositions of reusable skills [8], enabling fine-grained monitoring while ensuring compliance with safety standards [9]. Their modularity supports incremental recovery, avoiding the computational cost of full replanning [10]. While BTs can be manually designed, reactive planners automate their generation using a backchaining approach that selects skills based on preconditions and postconditions [11]. This allows robots to construct reactive execution policies that adapt to unexpected conditions in real-time without requiring full replanning.

In our prior work [12], we introduced a failure recovery framework that used a Vision-Language Model (VLM) for pre-execution plan verification. The system analyzed

*Equal Contribution
[1]Lund University, Lund, Sweden. E-mail: firstname.lastname@cs.lth.se
[2]KTH Royal Institute of Technology, Stockholm, Sweden. E-mail: jstyrud@kth.se
[3]ABB Robotics, Västerås, Sweden

input skills, execution conditions, the planned BT, and pre-execution images to assess whether the plan contained sufficient knowledge for successful execution. If critical preconditions or required skills were missing, it suggested modifications to prevent execution errors, reducing failures caused by incomplete task knowledge. However, this approach did not account for failures arising during execution due to unforeseen disturbances, environmental changes, or hardware errors.

While pre-execution verification helps prevent many failures, it cannot predict all possible execution-time issues. A robot may generate a valid pick-and-place plan, yet unexpected events, such as human intervention or object displacement, can still cause grasp failures. Addressing such failures requires real-time monitoring and corrective actions, which is only possible through a reactive mechanism. Without continuous failure monitoring, robots cannot effectively detect and adapt to failures as they occur, making reactive checks essential for robust autonomous execution.

Building on our prior work [12], this paper presents a unified failure recovery framework that extends pre-execution plan verification with real-time execution monitoring (Figure 1) to detect, identify, and correct errors dynamically. Our framework integrates reactive failure handling using a continuously updated execution history, which records skill execution states, timestamps, and scene graph updates for adaptive failure recovery. To improve situational awareness, we incorporate scene graphs that track object-object and robot-object spatial relationships throughout execution. Unlike [13], which generates scene graphs post-execution, our method updates them continuously, enabling immediate detection of environmental changes. Additionally, while [12] suggested missing skills only pre-execution, our approach supports both pre-execution and reactive skill suggestions, ensuring failures are addressed proactively and dynamically. This work makes the following key contributions:

- A unified failure recovery framework integrating Vision-Language Models (VLMs), reactive planners, and Behavior Trees (BTs) for pre-execution failure verification and real-time reactive failure handling.
- Real-time failure detection, identification, and correction using an incrementally updated execution history that tracks skill conditions, execution timestamps, and scene graph updates.
- Experimental validation in AI2-THOR [14] and a real-world ABB YuMi robot, demonstrating improved failure recovery across diverse environments.

II. Related Work

Failure recovery in robotics has been extensively studied, from predefined strategies to modern learning-based techniques and Large Language Models (LLMs) for adaptive failure handling. This section reviews these methodologies and highlights the distinctions between existing works and our approach.

A. Traditional Failure Recovery Strategies

Early methods relied on human intervention, predefined recovery strategies, and automated solutions based on failure mode analysis. While human-in-the-loop strategies offer flexibility, they are labor-intensive and limit scalability [15]. Predefined strategies handle known failure cases well but struggle with novel issues [16]. Systematic failure analysis, such as Failure Mode and Effects Analysis (FMEA), requires expert knowledge and does not generalize to dynamic environments [17]. Automated recovery methods attempt autonomy but remain constrained by predefined failure modes [18], [19]. Unlike these approaches, our framework continuously updates a dynamic execution history for real-time failure detection and adaptation.

B. Learning-Based Failure Recovery

Recent approaches explore reinforcement learning (RL) and imitation learning (IL) to develop recovery strategies from experience [5], [6]. RL-based methods require extensive training in simulations, making real-world deployment difficult [20]. IL-based methods like RACER [21] improve recovery using demonstrations but struggle to generalize. Neuro-symbolic methods combine structured reasoning with learning, improving interpretability but facing scalability challenges [22], [23], [24]. Our approach avoids data-heavy training by leveraging Vision-Language Models (VLMs) for reasoning-based failure recovery, enabling flexible and context-aware corrections in real time.

C. Failure Recovery with Large Language Models (LLMs) and Vision-Language Models (VLMs)

LLMs and VLMs have become integral to robotic failure recovery due to their reasoning capabilities. Several approaches leverage LLM-based reasoning for failure detection and correction, including REFLECT [25], AHA [20], DoReMi [26], ReplanVLM [27], RECOVER [22], and Code-as-Monitor [28]. REFLECT provides hierarchical post-execution summaries but lacks real-time intervention. AHA fine-tunes a VLM for failure detection at task checkpoints but lacks structured execution policies. DoReMi enforces dynamic execution constraints but relies on LLM-generated constraints, introducing variability. ReplanVLM integrates pre-execution validation with execution monitoring using GPT-4V but depends on LLM-driven re-planning rather than structured failure handling.

Unlike these, our framework integrates a reactive planner and Behavior Trees (BTs) for structured, real-time failure handling at both pre-execution and reactive levels. *RECOVER*[22] uses ontology-driven neuro-symbolic reasoning for real-time failure detection but requires domain-specific engineering, limiting adaptability. *Code-as-Monitor*[28] translates natural language constraints into executable monitors for proactive (handling foreseeable failures) and reactive failure detection but lacks explicit recovery mechanisms. Unlike these, our execution history continuously updates skill execution states, enabling VLMs to analyze failures dynamically rather than post-execution.

Compared to *AHA* and *ReplanVLM*, which focus on high-level reasoning or planning corrections, our approach ensures modular and adaptive failure recovery by integrating structured execution policies via BTs and a reactive planner. Additionally, recent work [29] explores intent-based BT planning using LLMs for goal interpretation, whereas our method actively modifies execution policies by suggesting missing preconditions, postconditions, and skills in real time, ensuring robust failure recovery in dynamic environments.

III. Background

In this section, we discuss the relevant concepts that serve as background knowledge for the paper.

A. Behavior Trees

Behavior Trees (BTs) are a hierarchical execution model valued for their modularity, flexibility, and reactivity in robotic decision-making [30], [31]. Originally developed for game AI, BTs now provide interpretable and scalable task execution in robotics [9], [7]. Their structure simplifies behavior design, modification, and debugging while enabling real-time adaptation to dynamic environments [32].

A BT is a directed acyclic graph where execution begins at the root node, propagating tick signals to evaluate and execute behaviors dynamically. Nodes return *Success*, *Failure*, or *Running*, with control-flow nodes (e.g., *Sequence*, *Fallback*) managing execution order and execution nodes (e.g., *action*, *condition*) implementing robot skills. This structured execution enables task decomposition and fine-grained monitoring. Once adapted to handle a failure, the BT becomes a reusable execution policy, reducing reliance on model queries and improving efficiency over time.

B. Reactive Planner

Reactive planners generate Behavior Trees (BTs) dynamically using backchaining, selecting skills that satisfy goal conditions [33]. Starting from the goal, the planner works backward through skill preconditions and postconditions, iteratively expanding the BT until all conditions are met or a termination criterion is reached. This approach enables robots to adapt to environmental changes without full re-planning, leveraging BT modularity for flexible execution [11]. The PDDL-based reactive planner used in this work follows [11], ensuring efficiency by removing redundant nodes and introducing composite subtrees for complex tasks. This facilitates real-time, autonomous failure recovery while maintaining computational efficiency. As backchaining inherently selects skills that achieve required postconditions, explicit VLM-generated postcondition suggestions are unnecessary.

C. Vision-Language Models

Vision-Language Models (VLMs) combine visual perception with language-based reasoning, making them effective for robotic failure recovery [22], [21]. They enable robots to detect, identify, and correct failures by analyzing execution conditions and task states.

In our prior work [12], GPT-4o was used for pre-execution verification, where the VLM assessed if a planned execution contained sufficient knowledge to succeed. It performed three key tasks: failure detection (checking for potential failures based on available conditions), failure identification (diagnosing root causes by analyzing missing or incorrect preconditions), and failure correction (suggesting modifications such as adding missing preconditions or required skills). This approach reduced failures due to incomplete task knowledge but did not address execution-time failures from unforeseen disturbances or environmental changes.

This work extends VLMs to real-time execution monitoring and correction. The VLM continuously analyzes execution states, providing corrective suggestions based on evolving conditions. To improve reasoning, we integrate scene graphs that dynamically track object-object and robot-object relationships, improving failure detection. Additionally, an execution history records skill preconditions, postconditions, and execution timestamps, enabling structured failure analysis. By combining pre-execution checks with reactive real-time monitoring, our framework ensures continuous adaptation to failures, enhancing robustness in autonomous robotic execution.

IV. Approach

To enable real-time robotic failure recovery, our framework integrates a reactive planner, Behavior Trees (BT), and Vision Language Models (VLM). The failure monitoring process is divided into pre-execution failure verification and real-time execution monitoring, each addressing failure detection, identification, and correction. Additionally, we extend the system with a scene graph and execution history to improve failure reasoning and adaptation. All failure handling mechanisms rely on the following key inputs:

- **Images** capturing the scene from multiple angles using two cameras (front and side views) to improve spatial understanding.
- **Skills** with predefined pre- and postconditions.
- **Known conditions** for environment reasoning.
- **Scene graph** representing spatial object relations.
- **Behavior Tree (BT)** defining execution policy.
- **Execution history** (real-time only) tracking past actions and scene updates.

Failure handling follows a three-phase process: *detection* identifies potential failures, *identification* determines the root cause by pinpointing the affected skill and unmet condition, and *correction* modifies the BT through precondition adjustments or skill additions to ensure successful execution. inspired by chain-of-thought [34] reasoning, we structure failure recovery prompt into these three phases. This improves the VLM performance by guiding it step-by-step toward the correct solution. If no failure is detected during the detection phase, the system skips the identification and correction steps, optimizing computational efficiency in both pre-execution and real-time monitoring.

To explain concretely our failure handling process, we use a peg-in-hole task, where the goal is to insert the

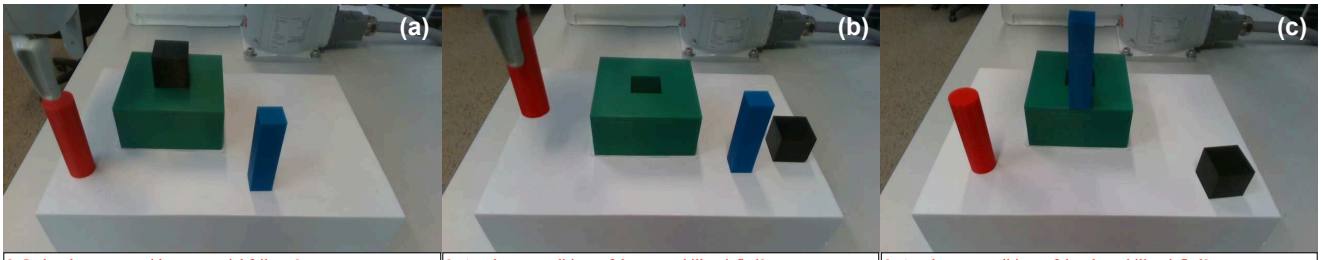

Fig. 2. Three failure instances with corresponding VLM responses. (a) Pre-execution verification detects that the black object blocks the hole, and the VLM suggests adding the missing precondition for the place skill. (b) Precondition verification identifies that the grasp skill fails due to an unmet condition, as the robot is already holding a red object. (c) Postcondition verification detects a failed placement since the blue object is on top of the green object instead of inside. Failure detection (red), identification (orange), and correction (blue) are indicated with corresponding VLM responses in black.

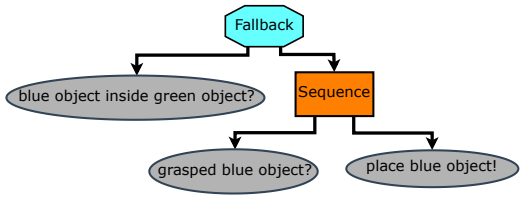

Fig. 3. BT of the peg-in-hole task without failure handling

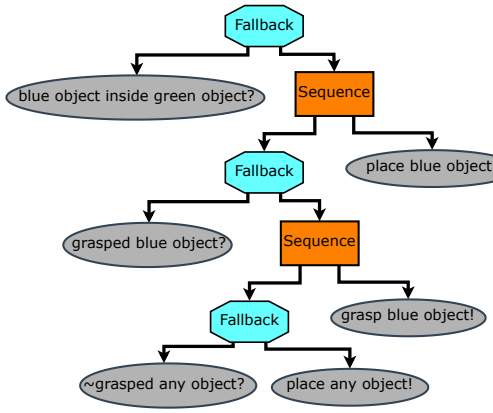

Fig. 4. Extended BT execution where a missing precondition is added, ensuring the gripper is empty before grasping target object.

blue object inside the green object, while red and black objects act as obstacles. Figures 2 and 5 illustrate different failure types with VLM responses. These figures also show various prompts, color-coded to distinguish between failure detection, identification, and correction questions posed to the VLM [1]. From here onward, we will consider a BT for peg-in-hole task execution that does not yet account for failures, as shown in Figure 3, unless specified otherwise.

A. Pre-Execution Failure Verification

Before execution [12], we validate the planned BT by proactively checking for missing preconditions or potential execution failures. This step prevents errors before they occur, reducing failures caused by incomplete task knowledge. A GPT-4o-based VLM performs this verification by analyzing the inputs.

- **Detection**: Flags anomalies where the planned BT may fail based on the current scene. *For example, in the peg-in-hole task, if a black cube blocks the hole, the pre-execution checker detects a potential failure (Figure 2(a)).*
- **Identification**: Pinpoints the failing skill and the root cause, whether due to missing knowledge or an incorrect assumption. *In this case, the VLM identifies that the place skill will fail as the BT does not ensure the hole is unoccupied before placement (Figure 2(a)).*
- **Correction**: Suggests a missing precondition to update the BT and prevent failure. *Here, the system adds ~occupied(hole) as a precondition for place, prompting the reactive planner to remove the black cube before placement (Figure 2(a)).*

B. Real-Time Failure Monitoring

While pre-execution verification minimizes failures, unexpected execution failures may still occur due to sensor inaccuracies, dynamic obstacles, or external disturbances. To handle these, we introduce a real-time failure monitoring module comprising a *Verifier* and a *Suggestor*. Both modules use the same inputs as pre-execution verification but incorporate continuously updated scene graphs, images, and execution history for improved reasoning.

1) Verifier: Ensures that execution aligns with expected conditions by performing *precondition verification before execution* and *postcondition verification after execution*.

a) Precondition Verification: Before executing a skill, the *Verifier* checks if the skill preconditions hold. Consider the BT in Figure 4 with an existing `~grasped any object` precondition in this case.

- **Detection**: Flags an anomaly if the preconditions for the skill in the BT are unmet. *For example, in the peg-in-hole task, if the robot has already grasped a red object but needs to grasp the blue object, the verifier detects an anomaly (Figure 2(b)).This failure can occur if a*

[1]Full prompts and code will be released after the submission process.

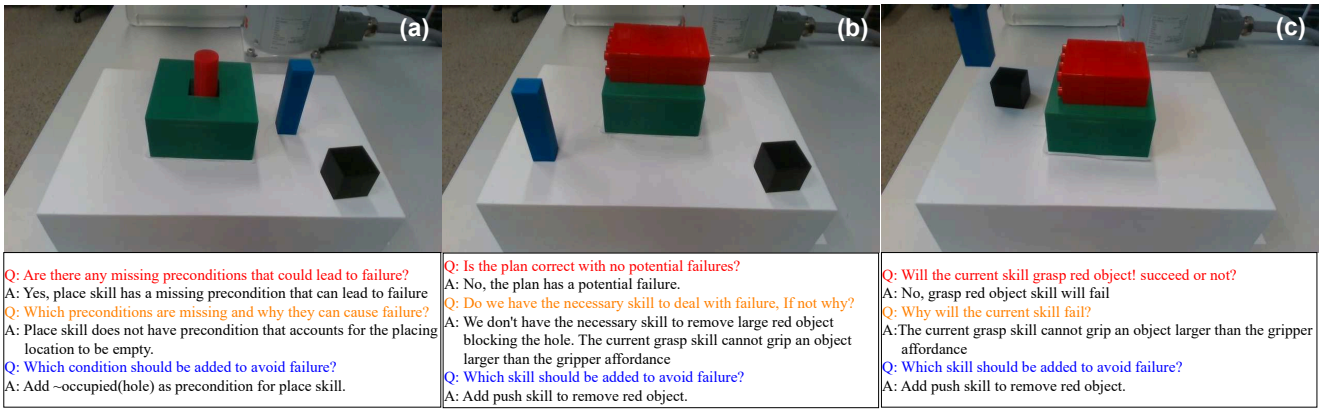

Fig. 5. The figure illustrates three failure scenarios and corresponding VLM responses. (a) Precondition suggestor: The red object inside the green object leads the VLM to identify a missing precondition for the place skill. (b) Pre-execution missing skill generation: The VLM identifies the need for a push skill to remove the red object. (c) Real-time missing skill generation: The VLM suggests generating the push skill during execution. Failure detection (red), identification (orange), and correction (blue) phases are depicted, with VLM responses in black.

human intervenes after the pre-execution failure check by manually placing the red object inside the gripper.

- **Identification**: Determines the violated precondition and the cause of failure. *In this case, it finds that the* `~grasped any object` *precondition of the grasp skill is not satisfied (Figure 2(b)).*
- **Correction**: Prevents execution by marking relevant preconditions as unsatisfied. The reactive planner will then automatically expand the BT to satisfy the marked preconditions. *For instance, the BT adapts by placing the currently held object before attempting the new grasp (Figure 2(b)).*

b) Postcondition Verification: After executing a skill, the *Verifier* checks if expected postconditions hold.

- **Detection**: Flags an anomaly if the executed skill fails to meet its postconditions. *For instance, if the robot places the blue object on top of the hole instead of inside, the verifier detects a failure (Figure 2(c)).*
- **Identification**: Identifies the violated postcondition and the cause of failure. *Here, it finds that the "inside" condition is violated because the object is on top rather than inside (Figure 2(c)).*
- **Correction**: Returns *Failure*, triggering the reactive planner to adjust execution dynamically. *The BT reattempts placement in the next tick (Figure 2(c)).*

2) Suggestor: The *Suggestor* dynamically infers missing preconditions when a skill fails due to unmet conditions.

- **Detection**: Flags an anomaly when a skill is likely to fail due to an unmet precondition. *For example, in the peg-in-hole task, the red object is already occupying the hole (Figure 5(a)).*
- **Identification**: Identifies the missing precondition and the cause of failure. *In this case, it determines that the place skill is missing a precondition ensuring the hole is empty before insertion (Figure 5(a)).*
- **Correction**: Suggests the missing precondition, prompting the BT to update accordingly. *The model suggests ~occupied(hole) as a precondition, allowing the reactive planner to expand the BT accordingly (Figure 5(a)).*

C. Skill Addition

While modifying preconditions can resolve many failures, some cases require introducing new skills. The skill addition can be suggested either pre-execution or reactively depending on when the potential failure case arises. The pre-execution stage implements our prior work in the [12] paper. if no existing skill can address a detected failure, the system suggests a missing skill (Figure 5(b)). In the reactive phase, the VLM checks execution feasibility before executing every skill. If the current skill is predicted to fail, a missing skill is suggested to remove the failure (Figure 5(c)).

- **Detection**: Identifies when the available skills can not resolve a failure. *For example, in the peg-in-hole task, if a non-pickable object blocks the hole and the system detects an unresolved failure (Figure 5(b)).*
- **Identification**: Determines the missing capability and the skill that fails due to this limitation. *In this case, the pick skill fails because the object is non-pickable (Figure 5(b)).*
- **Correction**: The VLM suggests a new skill to resolve the failure, ensuring compatibility with the robot's world model. The suggestion includes:
  - The name of the missing skill.
  - A code template defining the skill.
  - Predefined preconditions and postconditions.

  For example, if a robot cannot grasp an object, the VLM may suggest a "Push" skill as an alternative, providing a skill description with predefined conditions (Figure 5(b)). Figure 5(c) illustrates the reactive version occurring during execution, where the robot first places the blue object on the table before executing the "Push" skill to move the red object. To ensure consistency, the system restricts the VLM to known world model conditions, preventing arbitrary condition generation.

D. Scene Graph Representation

To enable real-time monitoring, our system maintains an evolving scene graph that tracks spatial relationships between

objects and the robot. Unlike REFLECT [13], which regenerates the scene graph from scratch at each timestep, our approach continuously updates it by modifying relationships and adding or removing nodes as needed.

The scene graph is constructed using:

- **RGB-D images and point clouds** for capturing scene depth and object positioning.
- **Grounding DINO** [35] for object detection and **SAM2** [36] for instance segmentation and tracking.
- **RANSAC and PCA-based pose estimation** to estimate 6D object poses.

Continuous updates improve efficiency and ensure execution consistency. *For example, in the peg-in-hole task, when the robot inserts the blue object into the green one, our system updates the scene graph by modifying the "on" relation to "inside" without reconstructing the entire graph.*

E. Execution History

The execution history maintains a log of skill executions, condition verification results, and environmental changes. Instead of explicit failure logging, which assumes perfect execution state knowledge, our approach captures execution traces via changes in the scene graph that help infer failures and inconsistencies.

- **Skill execution records**: Logs executed skills with timestamps.
- **Precondition and postcondition verification**: Tracks whether preconditions were met before execution and if postconditions held afterward.
- **Scene graph updates**: Records object positions and relationships before and after execution to analyze deviations.

For example, in the peg-in-hole task, if the blue peg is placed on top of the green hole instead of inside, the execution history logs the "Place" skill execution with its timestamp. The system records that the precondition was satisfied (e.g., the peg was grasped), but postcondition verification fails as the peg's spatial relation does not match the expected "inside" condition. The scene graph update reflects this deviation, showing the peg as "on" rather than "inside" the hole.

This structured history enables real-time adaptation by detecting execution anomalies, allowing the system to refine failure handling based on observed task progression.

V. Experiments and Results

We evaluate our failure recovery framework through both simulation benchmarks and real-world experiments. In simulation, we use benchmark tasks from REFLECT [13] in AI2-THOR [14], assessing how our system adapts to predefined failure cases. For real-world validation, we implement our framework on a robotic platform to evaluate its effectiveness in handling failures in physical environments.

A. Simulation Experiments

We evaluate our framework on REFLECT benchmark tasks, where failures occur during execution and are corrected post-execution using hierarchical summaries and scene graphs [25]. However, REFLECT lacks real-time adaptation, as failures are only detected and corrected after task completion.

Our approach instead uses a reactive planner and BTs to dynamically generate execution policies, enabling real-time monitoring and immediate failure correction. Unlike REFLECT, which reconstructs a new scene graph per execution, our system continuously updates it. Additionally, while REFLECT relies on LLM-generated post-execution corrections without correctness guarantees, our reactive planner ensures correctness through structured preconditions and postconditions.

We successfully applied our framework to all REFLECT benchmark tasks, achieving a 100% success rate across multiple runs. Real-time monitoring was sufficient, making pre-execution checks unnecessary. The *Verifier* ensured execution correctness, while the *Suggestor* resolved missing preconditions. Unlike REFLECT, which evaluates explanation, localization, and replanning success, we assess overall task completion. Since failures are proactively verified and reactively corrected during execution, post-execution replanning is unnecessary, reducing reliance on retrospective reasoning.

Key differences between REFLECT and our approach are summarized in Table I.

TABLE I
QUALITATIVE COMPARISON OF REFLECT AND OUR APPROACH

Feature	REFLECT	Our Approach
Execution Plan	Manually designed	Reactive BT
Failure Handling	Post-execution	Real-time
Scene Graph Update	Reconstructed post-execution	Maintained incrementally
Failure Detection	Post-execution	Real-time
Plan Correction	LLM-generated	Reactive BT

B. Real-World Experiments

For real-world validation, we deployed our framework on an ABB YuMi robot equipped with an RGB-D camera. We assessed its failure recovery capabilities across three tasks:

- **Peg-in-hole**: Inserting a peg into a hole with varying initial placements.
- **Object Sorting**: Sorting objects by color into designated locations.
- **Drawer Placement**: Placing an object inside a drawer.

Failures were introduced by modifying object placements, adding obstructions, or altering task constraints. Additionally, human intervention was used to induce failures during execution.

1) Baseline Approaches: We compared our approach against two baselines to assess the benefits of integrating pre-execution and reactive failure recovery mechanisms:

- **Pre-execution**: Check for plan verification [12].
- **Reactive**: Detect and correct failures during execution.

TABLE II
COMPARISON OF FAILURE RECOVERY BASELINES

Metric	Pre-execution	Reactive	Pre-execution + Reactive (Ours)
Task Success Rate	31.25%	100%	**100%**
Failure Detection Rate	31.25%	100%	**100%**
Failure Identification Rate	100%	100%	**100%**
Correction Success Rate	100%	100%	**100%**
Skill Suggestion Accuracy	50%	100%	**100%**

- **Our Approach (Pre-execution + Reactive)**: Combine pre-execution validation and real-time monitoring to prevent and correct failures dynamically.

Table II provides a quantitative comparison, showing that *Pre-execution + Reactive* achieves the highest performance. While *Reactive* matches its failure handling, it is significantly more expensive due to increased VLM queries, additional skill executions, and longer execution times. For instance, without pre-execution checks, a robot may start execution only to realize mid-task that a required object is missing, forcing backtracking and reactive correction causing delays and inefficiencies [2]. In contrast, our approach resolves pre-execution failures proactively whenever possible, reducing computational and execution overhead. Meanwhile, *Pre-execution* has lower accuracy as it addresses failures only before execution but remains the most efficient, avoiding costly real-time interventions. This highlights the trade-off between execution success and efficiency, where pre-execution handling is computationally cheaper but insufficient for real-time failures.

2) Evaluation Metrics: We evaluated our framework's ability to detect, identify, and correct failures across 16 pre-recorded failure cases, repeating each experiment 10 times. To assess false positives, we also ran each task 10 times without introducing failures.

We measured the system's accuracy in detecting failures, correctly identifying their root causes, and successfully correcting them. Additionally, we analyzed the proportion of pre-execution failures handled versus those requiring real-time intervention and assessed the accuracy of skill suggestions.

Table II summarizes the performance across these metrics. Our framework achieved a perfect 100% accuracy on all tasks, demonstrating strong failure recognition and reasoning capabilities. No false positives occurred when running the tasks without failures. Given that failure recovery systems are designed to achieve near-perfect accuracy, these results align with expectations. Future work should focus on evaluating the framework on more complex benchmarks to further assess its scalability and robustness.

3) Ablation Studies and Summary of Findings: To evaluate key components, we conducted ablation studies by selectively removing elements and analyzing their impact on failure recovery.

- **VLM vs. LLM**: Removing vision input weakens spatial and scene-aware failure detection, limiting object relation reasoning. Success drops from 100% (2 images) to 98% (1 image) and 95% (no images), assuming scene graph accuracy, which is not always guaranteed.
- **Scene Graph Contribution**: Assists spatial reasoning and removes scene ambiguity. Without it, success drops to 91.25%, highlighting its role in structured failure prediction.
- **Execution History Effectiveness**: Omitting execution history tracking did not significantly impact results, as we observed similar success rates with and without it. However, this does not imply that execution history is ineffective; its benefits may become more evident in more complex benchmarks.

Our findings confirm that combining pre-execution and reactive failure handling improves task success. Pre-execution checks prevent plan failures, while real-time monitoring improves adaptability. VLM-based reasoning strengthens failure detection and correction, and scene graphs with execution tracking improve system reliability by maintaining structured environmental context. These results validate our framework's effectiveness in autonomous failure recovery across diverse robotic tasks.

VI. CONCLUSION AND FUTURE WORK

This paper presented a unified failure recovery framework integrating VLMs, a reactive planner, and Behavior Trees (BTs) for pre-execution failure detection and reactive recovery in robotic execution. By incorporating a scene graph for structured perception and execution history for real-time monitoring, our approach dynamically adapts to failures, minimizing execution disruptions. Experimental validation on an ABB YuMi robot and simulation benchmarks demonstrated its superiority over using pre-execution and reactive methods separately. Ablation studies confirmed the importance of VLMs, structured scene understanding, and execution summaries in enhancing system reliability.

In the future, we aim to enhance our framework by integrating video and audio inputs for improved context-aware task monitoring. We plan to fine-tune open-source multimodal models for failure handling, reducing computational costs and improving efficiency. Additionally, we will leverage Vision-Language Action (VLA) models for autonomous skill generation with structured preconditions and postconditions, ensuring quality through static and integration checks. To extend real-time monitoring, we will incorporate holding conditions for proactive failure checking during execution. These advancements will enhance autonomous failure recovery, making robotic systems more adaptable and self-sufficient.

VII. ACKNOWLEDGEMENTS

We thank Jialong Li for valuable discussions. This work was supported by the Wallenberg AI, Autonomous Systems,

[2]See the video submission for details.

and Software Program (WASP) through the Knut and Alice Wallenberg Foundation and by Vinnova (NextG2Com, ref. no. 2023-00541). Experiments were partly conducted at ABB Corporate Research Center, Västerås, Sweden, with financial support from WASP. Generative AI tools were used for editing, including grammar and sentence structuring.

REFERENCES

[1] M. Löfving, P. Almström, C. Jarebrant, B. Wadman, and M. Widfeldt, "Evaluation of flexible automation for small batch production," *Procedia Manufacturing*, vol. 25, pp. 177–184, 2018, proceedings of the 8th Swedish Production Symposium (SPS 2018). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978918305912

[2] R. Liu, G. Wan, M. Jiang, H. Chen, and P. Zeng, "Autonomous robot task execution in flexible manufacturing: Integrating pddl and behavior trees in ariac 2023," *Biomimetics*, vol. 9, no. 10, 2024. [Online]. Available: https://www.mdpi.com/2313-7673/9/10/612

[3] E. Sharma, C. Henke, A. Mitrevski, and P. G. Plöger, "Adaptive compliant robot control with failure recovery for object press-fitting," 2023. [Online]. Available: https://arxiv.org/abs/2307.08274

[4] R. Wu, S. Kortik, and C. H. Santos, "Automated behavior tree error recovery framework for robotic systems," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 6898–6904.

[5] S. Kobayashi and T. Shibuya, "Reinforcement learning to efficiently recover control performance of robots using imitation learning after failure," in *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2022, pp. 1147–1154.

[6] J. Booher, K. Rohanimanesh, J. Xu, V. Isenbaev, A. Balakrishna, I. Gupta, W. Liu, and A. Petiushko, "Cimrl: Combining imitation and reinforcement learning for safe autonomous driving," 2024. [Online]. Available: https://arxiv.org/abs/2406.08878

[7] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. Chapman & Hall/CRC Press, 2017.

[8] F. Ahmad, M. Mayr, S. Suresh-Fazeela, and V. Krueger, "Adaptable recovery behaviors in robotics: A behavior trees and motion generators (btmg) approach for failure management," in *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2024, pp. 1815–1822.

[9] O. Biggar, M. Zamani, and I. Shames, "On modularity in reactive control architectures, with an application to formal verification," *ACM Transactions on Cyber-Physical Systems (TCPS)*, vol. 6, no. 2, pp. 1–36, 2022.

[10] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a unified behavior trees framework for robot control," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 5420–5427.

[11] J. Styrud, M. Mayr, E. Hellsten, V. Krueger, and C. Smith, "Bebop–combining reactive planning and bayesian optimization to solve robotic manipulation tasks," in *2024 International Conference on Robotics and Automation (ICRA)*. IEEE, 2024.

[12] F. Ahmad, J. Styrud, and V. Krueger, "Addressing failures in robotics using vision-based language models (vlms) and behavior trees (bts)," *arXiv preprint arXiv:2411.01568*, 2024, accepted at European Robotics Forum (ERF) 2025.

[13] Z. Liu, A. Bahety, and S. Song, "Reflect: Summarizing robot experiences for failure explanation and correction," *arXiv preprint arXiv:2306.15724*, 2023.

[14] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "AI2-THOR: An Interactive 3D Environment for Visual AI," *arXiv*, 2017.

[15] S. Itadera and Y. Domae, "Motion priority optimization framework towards automated and teleoperated robot cooperation in industrial recovery scenarios," 2024. [Online]. Available: https://arxiv.org/abs/2308.15044

[16] R. Wu, S. Kortik, and C. H. Santos, "Automated behavior tree error recovery framework for robotic systems," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6898–6904.

[17] F. Jusuf, A. Susanto, A. Waluyo, and N. Siwhan, "Review on defenses against common cause failures on digital safety system," in *AIP Conference Proceedings*, vol. 2374, no. 1. AIP Publishing, 2021.

[18] Y. Lei, J. Wilch, B. Rupprecht, and B. Vogel-Heuser, "Artificial intelligence planning of failure recovery strategies in discrete manufacturing automation," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2023, pp. 1–8.

[19] L. V. Alves and P. N. Pena, "Secure recovery procedure for manufacturing systems using synchronizing automata and supervisory control theory," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 1, pp. 486–496, 2020.

[20] J. Duan, W. Pumacay, N. Kumar, Y. R. Wang, S. Tian, W. Yuan, R. Krishna, D. Fox, A. Mandlekar, and Y. Guo, "Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation," 2024. [Online]. Available: https://arxiv.org/abs/2410.00371

[21] Y. Dai, J. Lee, N. Fazeli, and J. Chai, "Racer: Rich language-guided failure recovery policies for imitation learning," 2024. [Online]. Available: https://arxiv.org/abs/2409.14674

[22] C. Cornelio and M. Diab, "Recover: A neuro-symbolic framework for failure detection and recovery," 2024. [Online]. Available: https://arxiv.org/abs/2404.00756

[23] O. Bougzime, S. Jabbar, C. Cruz, and F. Demoly, "Unlocking the potential of generative ai through neuro-symbolic architectures: Benefits and limitations," 2025. [Online]. Available: https://arxiv.org/abs/2502.11269

[24] X. Zhang and V. S. Sheng, "Neuro-symbolic ai: Explainability, challenges, and future trends," 2024. [Online]. Available: https://arxiv.org/abs/2411.04383

[25] Z. Liu, A. Bahety, and S. Song, "Reflect: Summarizing robot experiences for failure explanation and correction," 2023. [Online]. Available: https://arxiv.org/abs/2306.15724

[26] Y. Guo, Y.-J. Wang, L. Zha, and J. Chen, "Doremi: Grounding language model by detecting and recovering from plan-execution misalignment," 2024. [Online]. Available: https://arxiv.org/abs/2307.00329

[27] J. Wang, Z. Wu, Y. Li, H. Jiang, P. Shu, E. Shi, H. Hu, C. Ma, Y. Liu, X. Wang, Y. Yao, X. Liu, H. Zhao, Z. Liu, H. Dai, L. Zhao, B. Ge, X. Li, T. Liu, and S. Zhang, "Large language models for robotics: Opportunities, challenges, and perspectives," 2024. [Online]. Available: https://arxiv.org/abs/2401.04334

[28] E. Zhou, Q. Su, C. Chi, Z. Zhang, Z. Wang, T. Huang, L. Sheng, and H. Wang, "Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection," 2024. [Online]. Available: https://arxiv.org/abs/2412.04455

[29] X. Chen, Y. Cai, Y. Mao, M. Li, W. Yang, W. Xu, and J. Wang, "Integrating intent understanding and optimal behavior planning for behavior tree generation from human instructions," *arXiv preprint arXiv:2405.07474*, 2024.

[30] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[31] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," 2020.

[32] J. Styrud, M. Iovino, M. Norrlöf, M. Björkman, and C. Smith, "Automatic behavior tree expansion with llms for robotic manipulation," 2024. [Online]. Available: https://arxiv.org/abs/2409.13356

[33] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive planning and acting using behavior trees," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8839–8845.

[34] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. H. Chi, Q. V. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, 2022.

[35] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, Q. Jiang, C. Li, J. Yang, H. Su, J. Zhu, and L. Zhang, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," 2024. [Online]. Available: https://arxiv.org/abs/2303.05499

[36] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer, "Sam 2: Segment anything in images and videos," 2024. [Online]. Available: https://arxiv.org/abs/2408.00714