Resilience Matters for Embodied Agents System: New Metrics, Systematic Evaluation, and Optimization

Yapeng Liu[1,2,3], Yuanzhao Zhai[1,2], Xudong Gong[1,2], Dawei Feng[1,2]
Bo Ding[1]✉, Addison Lin Wang[3]✉, Huaimin Wang[1,2]

[1]PDL Lab, College of Computer Science and Technology,
National University of Defense Technology
[2]State Key Laboratory of Complex & Critical Software Environment,
Changsha 410073, Hunan, China
[3]EmPACT Lab, Nanyang Technological University, Singapore

Code: `EAS-Resilience-Evaluation` | **Web:** `EAS-Resilience-Evaluation`

Abstract

Embodied Agents System (EAS) are increasingly deployed in open-world physical domains, where reliability directly dictates deployment quality and human-agent trust. However, existing evaluations rely on outcome-centric metrics as success rate or safety scores that collapse diverse execution trajectories into coarse scores, obscuring the dynamic processes underlying agent behavior. Therefore, they ignore a critical property of EAS – which we define as the **Resilience** – that reflects how EASs *recover*, *stabilize*, and *extend* under perturbations and across iterative updates. The lack of resilience is particularly critical in open-world environments due to continuous unexpected disruptions, thus directly affecting the quality of EAS deployment. To address this problem, we gain insight from the resilience-engineering concepts to EAS groundings and propose a novel resilience evaluation framework that can be flexibly applied to any EAS. Specifically, we define the first comprehensive resilience metrics suite for EASs system that exposes *Rebound*, *Stability*, and *Graceful Extensibility* across embodied tasks execution, providing a practical grounding for EAS resilience analysis. We further implement the resilience evaluation layer that transforms execution evidence into assessments for comparison, diagnosis and optimization. Across 400 household tasks with 10 representative EAS baselines, our evaluation reveals process-level distinction hidden by outcome metrics, including recovery cost differences among successful episodes ($\Delta C_{\text{rec}} = 25.2$), increased instability under semantic perturbations, and task-family-specific degradation under stress. Metrics-guided optimizations reduce recovery cost by 42.94%, increase stability by 19.87%, and improve graceful extensibility completion by 10.08%, showing the diagnostic effect of resilience evaluation. Beyond these targeted gains, our results reveal trade-offs among resilience characteristics, suggesting that a resilient EAS construction should be evaluated and configured according to deployment-specific requirements.

1 Introduction

Embodied Agents System (EAS) shows great potential in domains where physical interaction and long-term task completion are essential, especially for the collaboration domain [21, 4]. Recent advances in large language models (LLMs) further strengthen EAS capabilities of reasoning and interaction, along with increased complexity in reliability evaluation [36, 34, 6]. As illustrated in

✉Corresponding authors: Bo Ding (`dingbo@aliyun.com`) and Addison Lin Wang (`linwang@ntu.edu.sg`).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

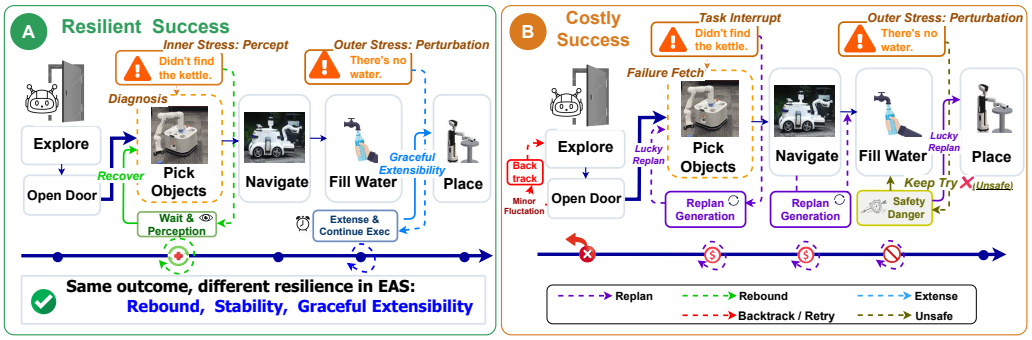


Figure 1: Outcome hide process resilience difference in EAS. (A) A resilient success trajectory handles inner and outer stress through resilient mechanisms, (B) A costly success relies on repeated backtrack, replan and unsafe behaviors.

Fig. 1, two EASs receive the same household instruction and achieve the same outcome, but their thought and execution process can be fundamentally different. This difference matters because EAS operates in dynamic environments where disruptions have continuously emerged [22, 9], indicating that reproducible reliability is critical for physical tasks and collaboration trust during execution [10, 42]. However, current EAS evaluation largely remains outcome or safety metrics, such as success rate or aggregated safety triggers [6, 41, 19, 34]. They collapse different execution processes into the same score, making brittle success indistinguishable from trustworthy adaptation success [25, 26]. This limitation points to a missing property of EAS – which we define as the ***Resilience*** in EAS – that reflects how EASs *recover*, *stabilize*, and *extend* under stress.

To address this problem, we gain insights from the resilience-engineering concepts to EAS groundings. System resilience evaluation captures the intrinsic property of a system, which maintains its acceptable execution and effective recovery when unexpected disturbances happen [12, 3]. Woods further decomposes system resilience into four capabilities, including rebound, robustness, graceful extensibility under stress, and sustained adaptability [37]. Specifically, we define the first comprehensive **Resilience Metrics** for EAS that expose *Rebound*, *Stability*, and *Graceful Extensibility* throughout execution: recovery cost C_{rec}, stability sensitivity β, stress response mapping $M_f(\lambda)$ and stress capacityλ_f^*. We further implement the **Resilience Evaluation Layer** that extracts signals from trajectories, monitors, and LLM judge, aggregates them through stage baselines and controlled stress intensities. This layer non-intrusively enables comparison, diagnosis and optimization guidance for EAS resilience evaluation.

We conduct experiments on Habitat-Sim with 400 household tasks and 10 representative EAS methods. Focus on EAS resilience Metrics Validity, Benchmark and Practical Optimization, our results show that resilience evaluation reveals process-level disparities, benchmarking EAS methods and has practical diagnosis effectiveness. Among success episodes, physical perturbations increase recovery cost from 11.6 to 36.3, semantic perturbations raise instability to $\Delta\beta = 0.32$, and stress tests expose degradation, recovery cost and stress intensities are strongly correlated (Spearman $\rho = 0.82$). For methods, Reflexion and CoPAL achieve comparable SR (56.3% vs. 55.8%) but differ by $4.8\times$ in recovery cost, showing that EASs have distinct resilience signatures. We also find that there is usually a trade-off in resilience aspects for EAS method design. Metric-guided optimizations further improve resilience performance, demonstrating that our evaluation layer is both diagnostic and actionable. In summary, our work makes three major contributions.

1. **Embodied Resilience Metrics and Evaluation Layer:** We formalize resilience metrics as three aspects: *Rebound*, *Stability*, and *Graceful Extensibility*. Then we complement EAS evaluation with the resilience evaluation layer, a practical grounding for measuring EAS resilience property quantification.
2. **Evidence and benchmarking:** We perform a experimental study on 10 representative EAS methods, demonstrating that our metrics successfully distinguish between brittle and resilient behaviors (Sec 3.2.2), we benchmark them, and show their resilience signatures under our resilience evaluation (Sec 3.2.3).
3. **Resilience evaluation provides practical optimization:** We provide quantitative evidence linking specific diagnosis to resilience improvements (Sec 4), showing that resilience-guided diagnosis can support practical optimization, improving EAS resilience while holding the resilience trade-off design principle.

2 Related Works

LLM-Enhanced EAS Evaluation: From Outcome to Resilience Metrics. LLM-enhanced EAS close the perception-execution loop by taking language-based reasoning into embodied actions generation, and they evolved from instruction grounding to feedback reasoning and self-correction [2, 20, 14, 32, 33, 11]. Recent works [15, 23, 44, 35] further explore adaptation, consistency, and open-ended skill acquisition to build trustworthy EAS. However, classical EAS evaluation remains largely outcome-centric as PARTNR emphasizing success metrics [42, 45, 5, 27]. Under embodied scenarios – physical irreversibility, partial observability, and dynamic constraints, such outcomes cannot distinguish robust adaptation from brittle success [21, 6, 34].

Focused on EAS reliability evaluation, recent works construct perturbation test to measure safety-aware planning [41, 39, 7]. They provide reliability diagnostics, but mainly whether an agent violates constraints or approaches a hazardous state in perturbation tests rather than normal process-level execution [19, 46]. We present the resilience metrics to evaluate *EAS Resilience* as an intrinsic property of EAS, measuring how the system behaves from inner stress or perturbations. In this way, we complement EAS evaluation with our resilience evaluation layer to measure how the resilience of EAS exhibits and provide practical optimization guidance.

Resilience Concepts Practical Evaluation in EAS Execution. In systems engineering, resilience emphasizes the capability to endure, adapt to, and recover from disruptions [28, 8]. Woods organizes the technical expression of resilience into four concepts: rebound, robustness, graceful extensibility and sustained adaptability [37]. This perspective holds that handling open-world complexities relies not only on failure prevention, but on continuous, dynamic adaptation during execution [13]. Classical resilience evaluation works evaluate loss severity and recovery latency, but their reliance on exact state matching or crash detection proves insufficient for embodied scenarios [17, 31, 27]. Recently, resilience-relevant evaluating methods for EAS emerged, they evaluate on semantic safety under hazardous instructions and physical cases [47, 43, 22]. However, these methods mix EAS resilience with adversarial security or perturbation stability. They evaluate the failure under static perturbation, but rarely capture the process-aware dynamics like recovery or extensibility.

Grounding with resilient EAS construction, we implement the resilience concepts into three resilient aspects. *Rebound* captures the burden of closed-loop recovery; *Stability* captures robustness as decision consistency; and *Graceful Extensibility* captures bounded degradation and remaining operational margin under stress. Sustained adaptability is treated as a longitudinal extension because it is less directly observable within dynamic EAS execution. Our resilient evaluation realize the practical evaluation layer, providing the validated instrument that reveals the EAS resilience property.

3 The Proposed EAS Resilience Evaluation Framework

Overview. We formalize the Resilience **Metrics** and Resilience **Evaluation Layer** for EAS, which goal is to evaluate system resilience from execution dynamics. Given an episode artifact $(\tau, \mathcal{L}, \mathcal{C}, \mathcal{J})$, where $\tau = \{(o_t, a_t, r_t)\}_{t=1}^{T}$ is the execution trajectory, $\mathcal{L}$ is the runtime log, $\mathcal{C}$ is the monitor, and $\mathcal{J}$ is the LLM Judge signal, the evaluation layer maps execution dynamics to three resilience aspects:

$$\mathcal{M} : (\tau, \mathcal{L}, \mathcal{C}, \mathcal{J}) \mapsto (\mathcal{M}_{\text{rebound}}, \mathcal{M}_{\text{stability}}, \mathcal{M}_{\text{GE}}).$$

For episode i, let t denote a runtime step. At each step, we record task progress $p_t \in [0, 1]$, proposition completion $q_t \in [0, 1]$, and execution mode m_t. We define a local execution anchor as $a_t := (f, b_t^p, b_t^q, m_t)$, where f is the task family, and b_t^q is the residual proposition load. For each anchor, we estimate a local **Stage Baseline** from clean or reference episodes under the same a_t:

$$\mathcal{N}(a_t) := \big(\tau^*(a_t), W^*_{\text{rem}}(a_t), \bar{T}^{\text{cog}}(a_t), \bar{N}^{\text{phy}}(a_t), \bar{\Delta p}(a_t), \bar{\Delta q}(a_t), \bar{r}(a_t)\big).$$

Here, $\tau^*(a_t)$ is the nominal cycle time, $W^*_{\text{rem}}(a_t)$ is the estimated remaining work from the anchor, $\bar{T}^{\text{cog}}(a_t)$ and $\bar{N}^{\text{phy}}(a_t)$ summarize cognitive and physical effort, $(\bar{\Delta p}(a_t), \bar{\Delta q}(a_t))$ summarize task progress and proposition completion, $\bar{r}(a_t)$ summarizes nominal risk load. The Stage Baseline provides a common local reference for resilience metrics computation and the reliability of statistical generalization on resilience metrics. In parallel, we take LLM-as-a-Judge to densify sparse execution feedback. The LLM judges our process from instruction, state, trajectory, and env feedback, and then produces structured reward for goal progress, rational action, and efficiency (detailed in Appendix E.2). Based on this shared evidence substrate, we instantiate system resilience from three complementary aspects: *Rebound*, *Stability*, and *Graceful Extensibility* as Table 1.

Table 1: Overview of resilience metrics with aspect, type (**D**: deterministic outcome, **S**: statistical, **A**: analysis, **T**: time-series), and data source.

Sym.	Metric Name	Calculation	Aspect	Type	Data Source
C_{rec}	Recovery Cost	$C_{\text{rec}} = \frac{1}{W^*_{\text{rem}}(a_{t_d})} \sum_{t_d}^{t_r} \frac{\Delta\tau_t}{\tau^*(a_t)} \cdot g_t^{\text{rec}}$	Rebound	D, S	System Logs
β	Policy Stability	$\beta = \sup_{z,\rho} \frac{\lvert\ell(\pi,\rho(z))-\ell(\pi,z)\rvert}{\lVert\rho\rVert}$	Stability	S	Sensitivity Test
$M_f(\lambda)$	Stress Curve Mapping	$M_f(\lambda) = \frac{1}{K}\sum_{j=1}^{K} \tilde{r}_{(j),f}(\lambda) - 1$	Extensibility	D, S	Stress Test
λ_f^*	Stress Capacity	$\lambda_f^* = \sup\{\lambda \in [0,1] \mid M_f(\lambda) \geq 0\}$	Extensibility	S, A	Stress Grid

3.1 Metrics Design

1) Rebound Metrics: Recovery Cost to Availability. Rebound measures how much extra work the agent must expend to return the acceptable execution after disruption. We use the **Recovery Cost** as the primary indicator, revealing the efficient rebound and brute-force rebound. For detailed implementation, please refer to Appendix A.1. For a rebound window $[t_d, t_r]$, t_d is the first step at which execution departs from the nominal continuation under the local anchor a_t, and t_r is the first step at which the task re-enters the acceptable execution. Execution *StageBaseline* $\mathcal{N}(a_t)$ (execution local reference, refer to Sec 3.2) provides the nominal local reference, including the remaining nominal work $W^*_{\text{rem}}(a_t)$, and the expected local execution statistics.

We decompose recovery cost into three aligned parts: **Cognitive recovery**, **Physical recovery**, and **State debt** in Eq. 1. We aggregate their excess variables as

$$\mathbf{z}_t^{\text{cog}} = \frac{(T_t^{\text{cog}} - \bar{T}^{\text{cog}}(a_t))_+}{\bar{T}^{\text{cog}}(a_t) + \epsilon}, \quad \mathbf{z}_t^{\text{phy}} = \frac{(N_t^{\text{phy}} - \bar{N}^{\text{phy}}(a_t))_+}{\bar{N}^{\text{phy}}(a_t) + \epsilon}, \quad \mathbf{z}_t^{\text{deb}} = \frac{(r_t - \bar{r}(a_t))_+}{\bar{r}(a_t) + \epsilon}. \tag{1}$$

where $\mathbf{z}_t^{\text{cog}}$ measures excess planning, perception effort; $\mathbf{z}_t^{\text{phy}}$ measures excess navigation, retry effort; and $\mathbf{z}_t^{\text{deb}}$ measures lagged progress leakage, risk load. $(\cdot)_+$ indicates the ReLU function, we record the excess costs while discarding pre-complete [1]. Detailed illustration refers to appendix Eq.1. To make the three channels comparable under the same anchor-conditioned regime, we convert each local excess vector into a covariance-adjusted deviation as Mahalanobis Distance[24]:

$$g_t^u = \sqrt{\mathbf{z}_t^{u\top} \left(\Sigma_u(a_t) + \eta I\right)^{-1} \mathbf{z}_t^u}, \qquad u \in \{\text{cog}, \text{phy}, \text{deb}\}, \tag{2}$$

where $\Sigma_u(a_t)$ is the covariance estimated from standard stage baselines, ηI is the regularizer. This step removes scale inconsistency and accounts for correlation within each recovery channel. We then aggregate the three channel deviations into recovery intensity $g_t^{\text{rec}} = \sqrt{\frac{1}{U}\sum_u^U (g_t^u)^2}$. Finally, for episode i, the *Rebound Cost* is defined as

$$C_{\text{rec}}^{(i)} = \frac{1}{W^*_{\text{rem}}(a_{t_d}) + \epsilon} \sum_{t=t_d}^{t_r} \frac{\Delta\tau_t}{\tau^*(a_t) + \epsilon} \cdot g_t^{\text{rec}}. \tag{3}$$

This quantity measures the total extra work required to restore acceptable execution after disruption. Normalization by $W^*_{\text{rem}}(a_{t_d})$ ensures that recovery is judged against the remaining nominal work at the disturbance point. We aggregate rebound signature through statistics in Appendix A.1.

2) Stability Metrics: Execution Fluctuations. Stability quantifies the system's ability to remain *consistent* under stresses, whether arising from stochasticity (Execution), perturbations (Sensitivity), or self-evolution. It is reflected in execution as frequent replanning, unstable progress, and abrupt shifts in the progress. We take *Policy Stability*(β) as the primary indicator, because it directly measures whether small changes in task instruction or execution condition amplify into process fluctuations. With implementation details like value networks refer to Appendix A.2, we describe how β is computed from execution representations.

We use the value function $V(\cdot)$ for each EAS execution representation, which is used to diagnose execution progress changes trend. Given the encoded state s_t, the value function is

$$V(s_t) = \mathbb{E}_\pi \left[\sum_{u=t}^{T} \gamma^{u-t} \tilde{r}_u \mid s_t \right], \tag{4}$$

where π is the agent policy and $\tilde{r}_u$ is the shaped reward by LLM judge and W^*_{rem}. For each state transition, we compute the one-step temporal-difference residual $\delta_t = \tilde{r}_t + \gamma(1-d_t)V(s_{t+1}) - V(s_t)$,

where d_t is the termination signal. We further compute the generalized advantage estimate A_t as an exponentially decayed accumulation of TD residuals: $A_t = \sum_{\ell=0}^{T-t} (\gamma\lambda)^\ell \delta_{t+\ell}$. Then our resilience loss for shape reward update is

$$\ell(\pi, z_i) = \frac{1}{T_i} \sum_{t=1}^{T_i} \sqrt{\mathbf{x}_t^\top \left(\Sigma(a_t) + \eta I\right)^{-1} \mathbf{x}_t}, \tag{5}$$

where z_i denotes an episode, $\Sigma(a_t)$ is the standard stage baselines covariance, and ηI is regularizer. This loss increases when the trajectory $\mathbf{x}_t$ contains large value variance, TD spikes, and advantages. We define β-*Stability* as this stress sensitivity to execution progress:

$$\beta_f = \sup_{z \in \mathcal{Z}_f, \rho \in \mathcal{P}_f} \frac{|\ell(\pi, \rho(z)) - \ell(\pi, z)|}{\|\rho\| + \epsilon}. \tag{6}$$

Here, $\mathcal{Z}_f$ is the episode set from task family f, $\mathcal{P}_f$ is the controlled perturbations set. A low β_f means that similar conditions lead to similar results, while high β_f means more replanning, value fluctuation, and progress instability. Although Eq. 6 holds the similar form on outcome-centric ML robustness, the resilience stability evaluates EAS execution dynamics. The EAS evaluation constructs the episode fluctuation loss $\ell(\pi, z_i)$ from monitor records and value-function signals $V(s_t)$, δ_t, and A_t, and then measures its change between clean and stress executions.

3) Graceful Extensibility Metrics: Stress Response & Adaptative Gains. Graceful Extensibility measures how an EAS performance extends as stress increases. It ensures that when stressors exceed the system's adaptive capacity, performance declines *gracefully* and predictably rather than collapsing catastrophically. This is a dynamic stress-response, revealing the resilience property of EAS. We therefore model Graceful Extensibility as a stress-response mapping $\lambda \mapsto M_f(\lambda)$, where λ is the stress severity and $M_f(\lambda)$ is the operational margin.

Let $\mathcal{T}_\lambda$ denote a stress operator with severity $\lambda \in [0, 1]$, which is related to perturbations and EAS self-evolvement. In our resilience evaluation, we unify external perturbation and internal evolution by the same stress-response in Appendix A.3. For an episode i from task family f under stress $\mathcal{T}_\lambda$, we first define a ***hard*** boundary monitor:

$$m_i(\lambda) = \min \left\{ \frac{p_{\mathcal{T},i}(\lambda)}{\tau_p}, \frac{q_{\mathcal{T},i}(\lambda)}{\tau_q}, \frac{\tau_{\text{rec}}}{C_{\text{rec},i}(\lambda) + \epsilon}, \frac{\tau_{\text{stab}}}{\beta_{\text{stab},i}(\lambda) + \epsilon}, \frac{\tau_{\text{safe}}}{U_i(\lambda) + \epsilon} \right\} - 1. \tag{7}$$

where U_i is the constraints violation load. The thresholds define the minimal acceptable task contract, $m_i(\lambda) < 0$ means that the contract is violated.

After accumulation of stage baselines from the task family f, we aggregate the statistical extensibility of the operation. For compact notation, let $Q_\alpha^f[X](\lambda) \triangleq Q_\alpha(X(\lambda) \mid f)$, where $Q_\alpha^f[X](\lambda)$ denotes the α-quantile of variable X over episodes from family f under stress λ. We construct the statistical extensibility vector

$$\mathbf{r}_f(\lambda) = \left[\frac{Q_\alpha^f[p_T](\lambda)}{\tau_p}, \frac{Q_\alpha^f[q_T](\lambda)}{\tau_q}, \frac{\tau_{\text{rec}}}{Q_{1-\alpha}^f[C_{\text{rec}}](\lambda) + \epsilon}, \frac{\tau_{\text{stab}}}{Q_{1-\alpha}^f[S_{\text{stab}}](\lambda) + \epsilon}, \frac{\tau_{\text{safe}}}{Q_{1-\alpha}^f[U](\lambda) + \epsilon} \right]^\top. \tag{8}$$

We take lower and upper quantiles for benefit variables and cost variables, which makes our metrics sensitive to graceful extensibility. Let $r_{(1),f}(\lambda) \le \cdots \le r_{(K_{\max}),f}(\lambda)$ be the sorted entries of $\mathbf{r}_f(\lambda)$. The statistical operational margin is defined as

$$M_f(\lambda) = \frac{1}{K} \sum_{j=1}^{K} r_{(j),f}(\lambda) - 1, \qquad 1 < K \le K_{\max}. \tag{9}$$

This bottom-K aggregation preserves the semantics of extensibility, but avoids the rigid behavior of a minimum optimization. For practical diagnosis, dynamic ***mapping*** between $M_f(\lambda)$ extensibility and execution performance is an important indicator to analyze EAS resilience. Further, the quantitative Graceful Extensibility indicator is the largest stress level where the EAS remains acceptable execution:

$$\lambda_f^* = \sup \{\lambda \in [0, 1] \mid M_f(\lambda) \ge 0\}. \tag{10}$$

We call λ_f^* the *stress capacity*. It gives the operational boundary of task family f: the maximum stress severity at which the system can still preserve its positive task progress in task family f.

Algorithm 1 Resilience Metrics Evaluation Pipeline

Require: Task family set $\mathcal{F}$; episode artifacts $\{(\tau_i, \mathcal{L}_i, \mathcal{C}_i, \mathcal{J}_i)\}_{i=1}^{N}$; StageBaseline estimation; perturbation set $\mathcal{P}_f$; stress grid $\Lambda = \{\lambda_j\}_{j=1}^{m}$; fixed thresholds and windows.
Ensure: $\mathcal{M}_{\text{rebound}}, \mathcal{M}_{\text{stability}}, \mathcal{M}_{\text{GE}}$

Estimate Stage Baselines $\mathcal{N}(a)$ from executions.
for each task family $f \in \mathcal{F}$ **do**
  for each stress level $\lambda \in \Lambda$ **do**
    for each episode i, artifact $(\tau_i, \mathcal{L}_i, \mathcal{C}_i, \mathcal{J}_i)$ **do**
      Extract stepwise signals: progress p_t, q_t, execution mode m_t, execution $\mathcal{L}_i$, critics $\mathcal{C}_i, \mathcal{J}_i$.
      Assign local execution anchors $a_t = (f, b_t^p, b_t^q, m_t)$.
      if Detect **Rebound** window $[t_d, t_r]$ **then**
        Compute $\mathbf{z}_t^{\text{cog}}$, $\mathbf{z}_t^{\text{phy}}$ and $\mathbf{z}_t^{\text{deb}}$.
        Compute rebound cost $C_{\text{rec}}^{(i)}$ from g_t^{rec}.
        Return to acceptable $\mathcal{N}(a_t)$.
      end if
      Collect **Stability** values $V(s_t)$, TD residuals δ_t, and GAE advantages A_t.
      Aggregate to vector $\mathbf{x}_t$ and β_i.
      Backpropagation Critic $\ell_{\text{stab}}(\pi, z_i)$ and Stage Baselines $\mathcal{N}_f(a)$.
    end for
    Retrieve stressed executions under $\mathcal{T}_\lambda$.
    Compute hard boundary monitors $m_i(\lambda)$ in family f.
    Aggregate episode statistics $\mathbf{r}_f(\lambda)$.
    Compute stress mapping: $M_f(\lambda) \leftarrow \frac{1}{K}\sum_{j=1}^{K} r_{(j),f}(\lambda) - 1$.
  end for
  Compute Graceful Extensibility stress capacity: $\lambda_f^* \leftarrow \sup\{\lambda \in \Lambda \mid M_f(\lambda) \geq 0\}$.
  Update Stage Baseline $\mathcal{N}$ and Statistical Aggregation Metrics
end for
return C_{rec}, β, M_f, λ_f^*, trajectory.

Based on the stress-response extensibility analysis, we could further compare EAS resilience profiles under matched execution conditions. We take adaptation mechanism on and off for analysis: Higher margins where both configurations remain acceptable provide additional operating headroom (*margin gain*). Conditions satisfying $M_{f,\text{on}}(\lambda) \geq 0,\ M_{f,\text{off}}(\lambda) \leq 0$ become *newly acceptable*.
We analyze stress capacity λ_f^* on experiments Sec. 3.2.3, and detail margin gains in Appendix A.3. The area under the shifted margin curve (AUDC) summarizes resilience adaptation performance across stress levels, and reliable comparisons of EAS resilience are represented by statistical execution data collection.

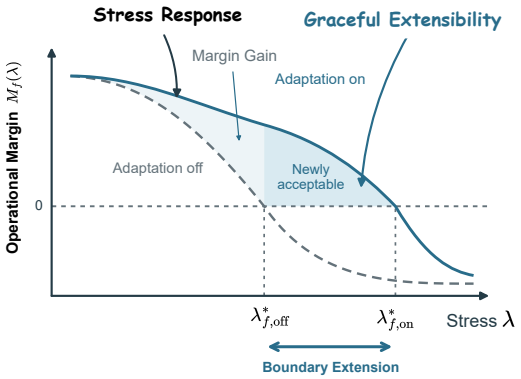


Figure 2: Stress response and adaptive gains. The increase from $\lambda^*_{f,\text{off}}$ to $\lambda^*_{f,\text{on}}$ marks boundary extension.

3.2 Resilience Evaluation Benchmark

The Resilience Evaluation Layer Construction We construct the Resilience Evaluation Layer as a non-intrusive evaluation instrument that complements the EAS reliability evaluation process. We present our resilience evaluation pipeline in Alg 1, it attaches probes during the EAS execution, then computes collected signals to resilience metrics. This design allows resilience to be evaluated as an execution property rather than an additional task objective.

3.2.1 Resilience EAS Testbed Setup

We conduct experiments on *Habitat-Sim 3.0* [29] with PARTNR-style [6] collaborative household tasks, including navigation, object manipulation, and object transportation scenarios. To expose different EAS resilience, we introduce controlled perturbations over physical states, semantic instructions, object states, and stress severity. The full perturbation specification, evaluated EAS methods, and dataset setting are reported in Appendix B.3, Appendix C.3, and Appendix B.1.

3.2.2 Resilience Metrics Validation: Execution Differences Hidden by Outcomes

We first validate whether resilience metrics ($\mathcal{M}_{\text{rebound}}, \mathcal{M}_{\text{stability}}, \mathcal{M}_{\text{GE}}$) capture distinct resilience property hidden by outcome-centric evaluation, and verify them as the measurement instruments.

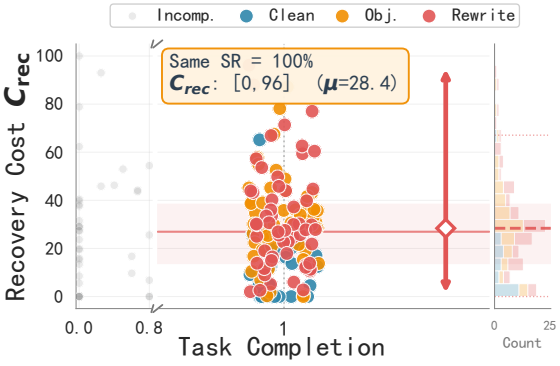


Figure 3: Different C_{rec} under outcomes.

Resilience metrics disperse differences in same outcome. Resilience metrics reveal significant performance differences that both SR and safety scores fail to capture, demonstrating the discrimination capability for EAS with similar success rates and safety scores. We take C_{rec} recovery cost as an example, Fig. 3 illustrates the "same outcome, different resilience" effect. This difference highlights the effectiveness of resilience evaluation and our resilience metrics.

Resilience Metrics Construct Validity and Sensitivity. Fig. 4 further confirms the construct validity of the three metric families. Rebound C_{rec} separates executions by recovery burden: perturbations increase recovery cost from 11.6 in clean episodes to 36.3 in average, with a significant distributional shift (Mann–Whitney $p < 0.001$). Stability β captures sensitivity to semantic-preserving instruction changes: semantic perturbations inflate instability to $\beta = 0.32$, mainly driven by divergent action generation instability ($\beta_{\text{out}} = 0.42$). Graceful Extensibility characterizes degradation as a stress-response curve rather than a binary failure event: as stress severity increases over $\lambda \in [0.2, 1.0]$, the relative margin ΔM decreases while recovery cost rises, with strong correlation (Spearman $\rho = 0.82$). These results show that the metrics measure complementary resilience properties and are not reducible to SR, safety, or completion alone. We conduct further resilience metrics statistical scalability analysis in Appendix D.3.

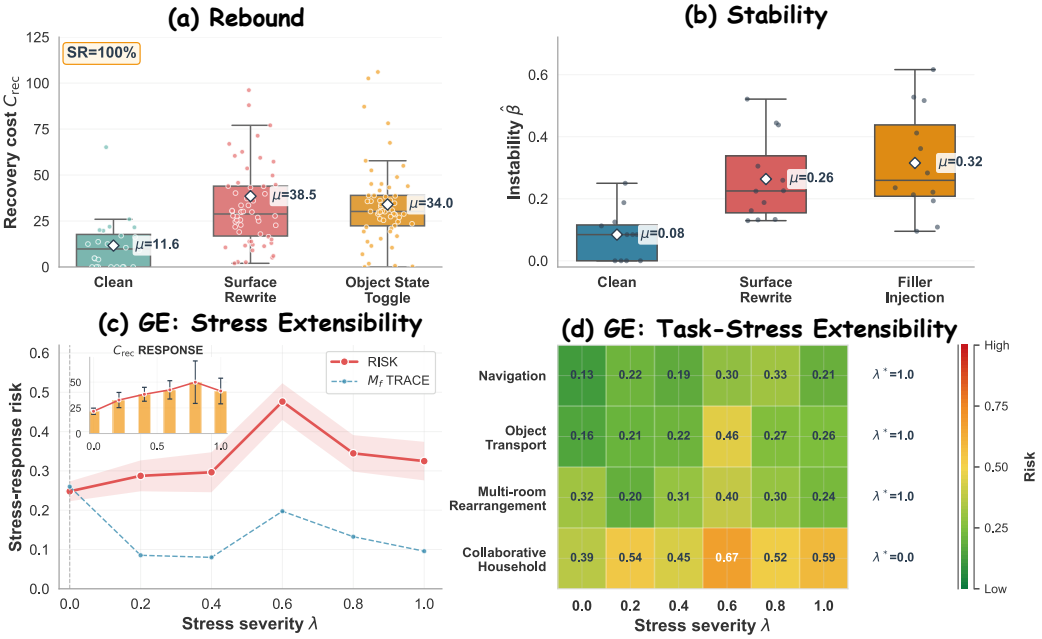


Figure 4: Resilience metrics construct validity results (*zoom in for better view*).

Additionally, we show the statistical reliability and sensitivity of the analysis on embodied agents system resilience metrics in Appendix D.3.1. This indicates that resilience metrics are related to EAS systematic inherent property, which maintains the consistency across embodied tasks execution.

Table 2: Comparative EAS resilience benchmark results in resilience aspects.

Method	Resilience Metrics			Outcome Ref.		Sim.
	C_{rec} (↓)	β (↓)	Stress Cap. λ^* (↑)	SR	Comp.	Steps
ReAct	6.5 ± 0.46	0.239 ± 0.011	0.50 ± 0.042	$38.5\% \pm 3.1\%$	$57.8\% \pm 4.6\%$	3374 ± 285
Reflexion	57.3 ± 5.07	0.299 ± 0.021	0.66 ± 0.035	$56.3\% \pm 4.2\%$	$75.0\% \pm 3.8\%$	4390 ± 312
CycleVLA	35.5 ± 2.91	0.400 ± 0.026	0.72 ± 0.047	$22.2\% \pm 2.8\%$	$47.5\% \pm 4.1\%$	4688 ± 420
CLARE	21.6 ± 1.68	0.344 ± 0.016	0.72 ± 0.052	$28.4\% \pm 3.5\%$	$46.0\% \pm 4.5\%$	4249 ± 360
SayCan	38.5 ± 2.49	0.283 ± 0.013	0.78 ± 0.046	$57.0\% \pm 4.1\%$	$72.6\% \pm 3.7\%$	4682 ± 380
AgentEvolver	19.2 ± 1.24	0.264 ± 0.012	0.21 ± 0.015	$48.8\% \pm 3.9\%$	$66.8\% \pm 4.2\%$	3084 ± 250
InnerMono	43.5 ± 2.69	0.149 ± 0.007	0.53 ± 0.026	$49.5\% \pm 4.0\%$	$67.5\% \pm 4.3\%$	3670 ± 290
CoPAL	11.9 ± 1.02	0.302 ± 0.017	0.85 ± 0.078	$55.8\% \pm 4.5\%$	$72.6\% \pm 3.9\%$	4846 ± 410
SMART-LLM	28.2 ± 2.20	0.339 ± 0.018	0.40 ± 0.036	$55.7\% \pm 3.8\%$	$74.3\% \pm 4.1\%$	5085 ± 395
Baseline	25.0 ± 2.03	0.200 ± 0.009	0.79 ± 0.042	$53.0\% \pm 3.6\%$	$69.7\% \pm 3.5\%$	3671 ± 275

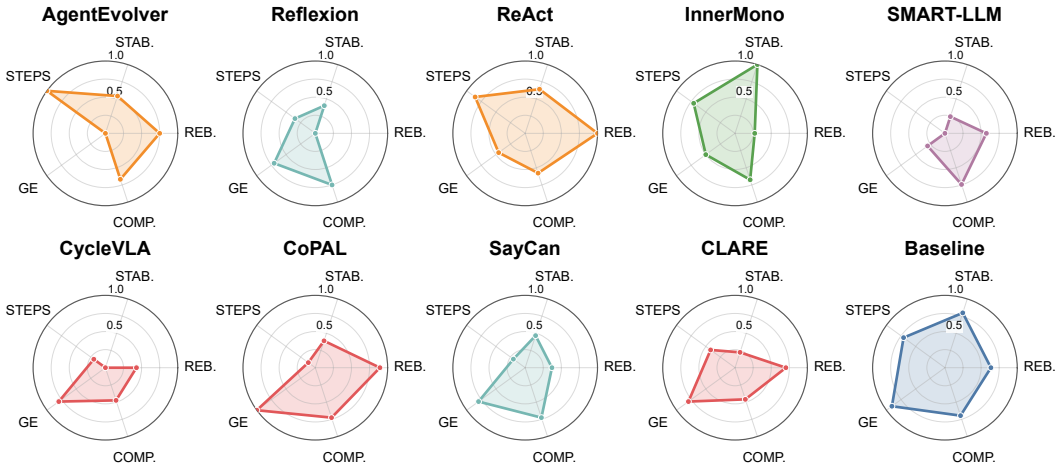


Figure 6: The Resilience Diagnosis. Visualizing the distinct behavioral archetypes of EASs.

We also conduct the scalability sensitivity analysis across Qwen3.5-9B, Qwen3-8B-Instruct, Qwen2.5-7B-Instruct, and Llama3.1-8B-Instruct LLM backbones in Appendix D.3, which shows that stronger backbones improve baseline task performance while preserving similar resilience profile patterns. This suggests that resilience is the system-level property in Appendix D.4. We report our LLM configuration and prompt templates in Appendix E.

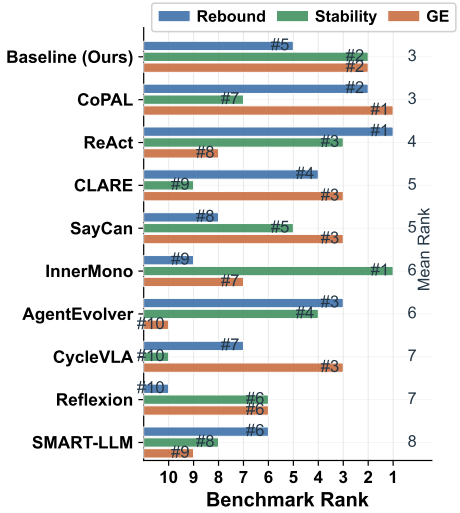


Figure 5: Resilience benchmark landscape.

3.2.3 Benchmarking EAS Resilience Results

We apply the validated resilience metrics to benchmark representative EAS methods under identical environments and perturbation settings. Table 2 reports the resilience benchmark scores, and Fig. 5 summarizes the normalized resilience profiles. For example, Reflexion and CoPAL achieve comparable SR (56.3% vs. 55.8%), but Reflexion incurs $4.8\times$ higher recovery cost than CoPAL (57.3 vs. 11.9). This indicates that EAS take different recovery processes, and resilience evaluation reveals these features.

This benchmark also shows that no current method dominates all resilience aspects. CoPAL achieves a favorable rebound–extensibility trade-off, with low recovery cost ($C_{\text{rec}} = 11.9$) and the highest stress capacity ($\lambda^* = 0.85$), but it requires long simulated execution steps. InnerMono has the best stability profile ($\beta = 0.149$), yet its recovery cost remains high ($C_{\text{rec}} = 43.5$). AgentEvolver has

Table 3: Comparative results of metric-guided targeted improvements.

Target	Methods	Rebound $C_{\text{rec}} \downarrow$	Rec. Win. $\downarrow$	$\beta_{\text{step}} \downarrow$	$\beta \downarrow$	GE. Comp. $\uparrow$	Formal GE
Rebound	Reflexion	59.11	3.13	0.2881	0.6834	0.887	–
	+Optimized	**33.73** ↓42.94%	**2.38** ↓23.96%	0.2079 ↓27.84%	0.6870 ↑0.53%	0.819 ↓7.67%	–
Stability	CoPAL	51.01	2.63	0.3185	0.6788	0.903	–
	+Optimized	43.01 ↓15.68%	3.00 ↑14.07%	**0.2552** ↓19.87%	**0.6772** ↓0.24%	0.750 ↓16.94%	–
GE	SMART-LLM	36.70	1.33	1.4594	0.6917	0.833	Incomplete
	+Optimized	99.59 ↑171.36%	2.25 ↑69.17%	0.7215 ↓50.56%	0.6371 ↓7.89%	**0.917** ↑10.08%	Complete†

† Formal GE is reported only for GE stress-tests, since it requires a stress-response curve over λ grid.

relatively low recovery cost ($C_{\text{rec}} = 19.2$), yet its stress capacity is the weakest among all methods ($\lambda^* = 0.21$), showing limited graceful extensibility. These results demonstrate that resilience should be interpreted as a multi-dimensional execution profile rather than a single scalar ranking.

EAS Methods Resilience Signatures Under the same benchmark protocol, we analyze distinct behaviors (performance, simulation steps, rebound, stability, and graceful extensibility) among the methods in Fig. 6. The shape of the radar plot serves as a visual fingerprint for the EAS resilience profile, and exhibits different resilience profiles, strength, and weakness. Some methods continue execution through costly recovery, some maintain stable execution, some fluctuate during planning, and some show graceful extensibility under stress. These profiles provide a resilience overview of current EAS methods and serve as a diagnostic handbook for optimization experiments. We conclude resilience benchmark findings as below:

Resilience Evaluation Benchmark Findings

1) Outcome-equivalent executions can be resilience-inequivalent. Success rate and completion describe whether an EAS finishes the task, but they do not reveal how much recovery, replanning instability, or stress margin is consumed during execution. Resilience metrics expose these hidden process differences.
2) There is no current EAS method dominates all resilience aspects. Some methods recover efficiently, some remain stable under semantic variation, and some tolerate stress, but no method dominates all dimensions. This indicates that EAS deployment should be matched to the target failure regime: difficult tasks require stronger Graceful Extensibility, while sensitive tasks require stronger Stability.
3) Resilience metrics provide actionable diagnosis. Abnormal metric patterns can be traced back to concrete execution events, such as repeated recovery loops, action oscillation, or stress-boundary collapse. This makes the evaluation layer not only a benchmark tool, but also a diagnostic basis for the metric-guided optimizations in Sec. 4.

4 EAS Optimization: Metric-Guided Resilience Diagnostics

Based on the resilience metrics and the EAS resilience profiles, we introduce a unified *metric-guided resilience optimization* to the EAS execution. The goal is not to replace the original planner, but to use our resilience metrics as diagnostic signals that indicate where the execution loop should be minimally repaired. We implement three targeted optimizations for Rebound, Stability, and Graceful Extensibility, with detailed implementation in Appendix C.5. We compare the *Original* and *Optimized* variants under identical evaluation conditions. Table 3 shows that the resilience metrics are not only descriptive, but can also guide practical resilience improvements. What's more, targeted optimization of a single resilient aspect often leads to compromises in other resilience aspects (optimizing GE results in high recovery cost C_{rec}). This confirms our findings: resilience is a multi-dimensional structural balance, and our resilient evaluation layer could capture this trade-off.

Opt1 Recovery: Feedback-Guided Recovery Implementation High recovery cost C_{rec} and long recovery windows indicate that the agent often spends extra reasoning or physical actions after local failures. To address this failure mode, we add a feedback-guided recovery layer on top of the centralized LLM planner. The layer monitors World-Graph evidence, recent execution feedback, and task-progress stagnation, and diagnoses recoverable faults such as missing graph nodes, stale object locations, repeated tool failures, or invalid object references. Instead of overriding the planner with a separate recovery policy, we get *Rebound Guidance* Π_{t+1} into the next planning context, including perception refresh, reflection, state summarization, or belief rollback. Then we formalize it as Habitat skill tools for better use. This keeps the original action space unchanged while converting low-level execution feedback $\mathcal{F}_t$ into planner recovery hints. As shown in Table 3, this optimization

reduces C_{rec} from 59.11 to 33.73 and the recovery window from 3.13 to 2.38, showing that the metric-guided feedback loop directly reduces redundant recovery effort.

Opt2 Stability: Consistency State Record High β values reveal unstable replanning behavior, where the agent adapts to new feedback but may oscillate between similar actions, repeat completed subtasks, or drift away from the current execution c_t. We therefore introduce a consistency record for replanning. At each step, it constructs a read-only phase guidance from the task goal, current phase, World-Graph state, current room and holding state, completed and pending subtasks, and recent action-response history. This design improves consistency without making the policy brittle. The optimized variant reduces β_{step} from 0.3185 to 0.2552 and improves β from 0.6788 to 0.6772, indicating that the stability metric captures and guides reductions in local replanning variance.

Opt3 GE: Boundary Long-tail Control for Difficulties Graceful Extensibility focuses on whether performance degrades smoothly as stress increases, rather than collapsing or entering unbounded long-tail behavior. In high-stress cases, we observed that the planner may repeatedly wait or receive near-duplicate execution feedback without making progress. Such behavior inflates execution cost and makes the stress-response curve insensitive to true adaptive capacity. To control this boundary effect, we add a long-tail monitor that detects dual-wait patterns, semantically repeated feedback, and stagnant task progress. When a boundary stall B_t is detected, the system injects a bounded cognitive reset that discourages further non-progressive waiting and requests either a concrete productive action or termination. This converts uncontrolled long-tail execution into measurable boundary execution.

5 Conclusion and Future Work

We presented the first Resilience Evaluation Framework for EASs. By designing novel metrics: *Rebound*, *Stability*, and *Graceful Extensibility*, our framework can expose resilience hidden pathologies, such as high-cost recovery and catastrophic forgetting, which remain invisible to outcome-centric metrics. Empirical analysis across representative baselines identified distinct Resilience Archetypes, confirming that architectural choices involve quantifiable trade-offs between execution, stability, and adaptability. The proposed optimization framework can help EASs developers to diagnose root causes and enforce non-degradation contracts to build a resilient and trustworthy EAS.

Future Work Future work can extend resilience evaluation along three directions: *Sim2Real* validation of whether simulation-derived resilience profiles transfer to physical robots, *theoretical analysis of trade-offs* among resilience aspects, and *deployment-aware optimization* that uses benchmarked resilience signatures to improve EASs for target application conditions. These directions can move resilience evaluation toward more principled, deployable, and balanced EAS design.

Acknowledgment

This work was partially supported by the Hunan Provincial Science and Technology Innovation Leading Talent Project (Grant No. 2023RC1005).

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] 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.

[3] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

[4] Rajkumar Buyya et al. Agentic artificial intelligence (ai): Architectures, taxonomies, and evaluation of large language model agents. *arXiv preprint arXiv:2601.12560*, 2026.

[5] Matthew Cavorsi, Lorenzo Sabattini, and Stephanie Gil. Multirobot adversarial resilience using control barrier functions. *IEEE Transactions on Robotics*, 40:797–815, 2023.

[6] Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, et al. Partnr: A benchmark for planning and reasoning in embodied multi-agent tasks. *arXiv preprint arXiv:2411.00081*, 2024.

[7] Zixing Chen, Yifeng Gao, Li Wang, Yunhan Zhao, Yi Liu, Jiayu Li, Xiang Zheng, Zuxuan Wu, Cong Wang, Xingjun Ma, et al. Hazardarena: Evaluating semantic safety in vision-language-action models. *arXiv preprint arXiv:2604.12447*, 2026.

[8] Richard I Cook and Beth Adele Long. Building and revising adaptive capacity sharing for technical incident response: A case of resilience engineering. *Applied ergonomics*, 90:103240, 2021.

[9] Pascale Fung, Yoram Bachrach, Asli Celikyilmaz, Kamalika Chaudhuri, Delong Chen, Willy Chung, Emmanuel Dupoux, Hongyu Gong, Hervé Jégou, Alessandro Lazaric, et al. Embodied ai agents: Modeling the world. *arXiv preprint arXiv:2506.22355*, 2025.

[10] Cuiyun Gao, Xing Hu, Shan Gao, Xin Xia, and Zhi Jin. The current challenges of software engineering in the era of large language models. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30, 2025.

[11] Yanjiang Guo, Yen-Jen Wang, Lihan Zha, and Jianyu Chen. Doremi: Grounding language model by detecting and recovering from plan-execution misalignment. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12124–12131. IEEE, 2024.

[12] Crawford S Holling et al. Resilience and stability of ecological systems, 1973.

[13] Erik Hollnagel. *Safety-I and safety-II: the past and future of safety management*. CRC press, 2018.

[14] 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.

[15] Frank Joublin, Antonello Ceravola, Pavel Smirnov, Felix Ocker, Joerg Deigmoeller, Anna Belardinelli, Chao Wang, Stephan Hasler, Daniel Tanneberg, and Michael Gienger. Copal: corrective planning of robot actions with large language models. In *2024 ieee international conference on robotics and automation (ICRA)*, pages 8664–8670. IEEE, 2024.

[16] Shyam Sundar Kannan, Vishnunandan LN Venkatesh, and Byung-Cheol Min. Smart-llm: Smart multi-agent robot task planning using large language models. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12140–12147. IEEE, 2024.

[17] Philip Koopman and Michael Wagner. Challenges in autonomous vehicle testing and validation. *SAE International Journal of Transportation Safety*, 4(1):15–24, 2016.

[18] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[19] Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Erran Li Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37:100428–100534, 2024.

[20] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

[21] Yang Liu, Weixing Chen, Yongjie Bai, Xiaodan Liang, Guanbin Li, Wen Gao, and Liang Lin. Aligning cyber space with physical world: A comprehensive survey on embodied ai. *IEEE/ASME Transactions on Mechatronics*, 2025.

[22] Xiaoya Lu, Zeren Chen, Xuhao Hu, Yijin Zhou, Weichen Zhang, Dongrui Liu, Lu Sheng, and Jing Shao. Is-bench: Evaluating interactive safety of vlm-driven embodied agents in daily household tasks. *arXiv preprint arXiv:2506.16402*, 2025.

[23] Chenyang Ma, Guangyu Yang, Kai Lu, Shitong Xu, Bill Byrne, Niki Trigoni, and Andrew Markham. Cyclevla: Proactive self-correcting vision-language-action models via subtask backtracking and minimum bayes risk decoding. *arXiv preprint arXiv:2601.02295*, 2026.

[24] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Sankhyā: The Indian Journal of Statistics, Series A (2008-)*, 80:S1–S7, 2018.

[25] Viacheslav Moskalenko, Vyacheslav Kharchenko, Alona Moskalenko, and Borys Kuzikov. Resilience and resilient systems of artificial intelligence: taxonomy, models and methods. *Algorithms*, 16(3):165, 2023.

[26] Fei Ni, Min Zhang, Pengyi Li, Yifu Yuan, Lingfeng Zhang, Yuecheng Liu, Peilong Han, Longxin Kou, Shaojin Ma, Jinbin Qiao, David Gamaliel Arcos Bravo, Yuening Wang, Xiao Hu, Zhanguang Zhang, Xianze Yao, Yutong Li, Zhao Zhang, Ying Wen, Ying-Cong Chen, Xiaodan Liang, Liang Lin, Bin He, Haitham Bou-Ammar, He Wang, Huazhe Xu, Jiankang Deng, Shan Luo, Shuqiang Jiang, Wei Pan, Yang Gao, Stefanos Zafeiriou, Jan Peters, Yuzheng Zhuang, Yingxue Zhang, Yan Zheng, Hongyao Tang, and Jianye Hao. Embodied arena: A comprehensive, unified, and evolving evaluation platform for embodied ai, 2025.

[27] Barak Or. Mttr-a: Measuring cognitive recovery latency in multi-agent systems. *arXiv preprint arXiv:2511.20663*, 2025.

[28] Amanda Prorok, Matthew Malencia, Luca Carlone, Gaurav S Sukhatme, Brian M Sadler, and Vijay Kumar. Beyond robustness: A taxonomy of approaches towards resilient multi-robot systems. *arXiv preprint arXiv:2109.12343*, 2021.

[29] Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, et al. Habitat 3.0: A co-habitat for humans, avatars and robots. *arXiv preprint arXiv:2310.13724*, 2023.

[30] Ralf Römer, Yi Zhang, and Angela P Schoellig. Clare: Continual learning for vision-language-action models via autonomous adapter routing and expansion. *arXiv preprint arXiv:2601.09512*, 2026.

[31] Neetesh Sharma, Armin Tabandeh, and Paolo Gardoni. Resilience analysis: A mathematical formulation to model resilience of engineering systems. *Sustainable and Resilient Infrastructure*, 3(2):49–67, 2018.

[32] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[33] Marta Skreta, Zihan Zhou, Jia Lin Yuan, Kourosh Darvish, Alán Aspuru-Guzik, and Animesh Garg. Replan: Robotic replanning with perception and language models. *arXiv preprint arXiv:2401.04157*, 2024.

[34] Xin Tan, Bangwei Liu, Yicheng Bao, Qijian Tian, Zhenkun Gao, Xiongbin Wu, Zhihao Luo, Sen Wang, Yuqi Zhang, Xuhong Wang, et al. Towards safe and trustworthy embodied ai: Foundations, status, and prospects. 2025.

[35] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[36] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[37] David D Woods. Four concepts for resilience and the implications for the future of resilience engineering. *Reliability engineering & system safety*, 141:5–9, 2015.

[38] David D Woods. The theory of graceful extensibility: basic rules that govern adaptive systems. *Environment Systems and Decisions*, 38(4):433–457, 2018.

[39] Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*, 2025.

[40] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

[41] Sheng Yin, Xianghe Pang, Yuanzhuo Ding, Menglan Chen, Yutong Bi, Yichen Xiong, Wenhao Huang, Zhen Xiang, Jing Shao, and Siheng Chen. Safeagentbench: A benchmark for safe task planning of embodied llm agents. *arXiv preprint arXiv:2412.13178*, 2024.

[42] Naoki Yokoyama, Sehoon Ha, and Dhruv Batra. Success weighted by completion time: A dynamics-aware evaluation criteria for embodied navigation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1562–1569. IEEE, 2021.

[43] Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, et al. R-judge: Benchmarking safety risk awareness for llm agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1467–1490, 2024.

[44] Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, et al. Agentevolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

[45] Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, et al. Vlabench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11142–11152, 2025.

[46] Tao Zhang, Kaixian Qu, Zhibin Li, Jiajun Wu, Marco Hutter, Manling Li, and Fan Shi. Using large language models for embodied planning introduces systematic safety risks. *arXiv preprint arXiv:2604.18463*, 2026.

[47] Zihao Zhu, Bingzhe Wu, Zhengyou Zhang, Lei Han, Qingshan Liu, and Baoyuan Wu. Earbench: Towards evaluating physical risk awareness for task planning of foundation model-based embodied ai agents. *arXiv preprint arXiv:2408.04449*, 2024.

Appendices

A Practical Mechanisms Details **15**

A.1 Rebound Metrics Detailed Calculation 15
A.2 Stability Metrics Detailed Calculation 17
A.3 Graceful Extensibility Metrics Detailed Calculation 19
A.4 Value Function Implementation 21

B Dataset Illustration **23**

B.1 Dataset Structure and Experiments Components 23
B.2 Datasets Generation 23
B.3 Stress Setting 26
B.4 Validity & Limitation 28

C Resilience Evaluation Implementation **28**

C.1 Asynchronous EAS and Resilience Evaluation Execution 28
C.2 Methods Comparison 29
C.3 EAS Methods Reproduction & Implementation 30
C.4 Context Evolution Protocol 32
C.5 Diagnosis-Guided Execution Changes 32

D Detailed Experiments Results **33**

D.1 Computation Resources 33
D.2 Resilience Aspect Analysis in Validity Experiments 33
D.3 Statistical Reliability and Sensitivity 34
D.4 LLM Judge Consistency and Simulator Ground-Truth Alignments 35
D.5 Embodied Execution Case Analysis 37

E LLM Configuration and Prompt **41**

E.1 LLM Configurations Overview 41
E.2 LLM-as-a-Judge Usage 41
E.3 LLM Plan Reasoning Backbone Usage 44
E.4 Datasets Generation LLM Usage 46
E.5 EAS Self-Evolve LLM Usage 47

NeurIPS Paper Checklist **50**

A Practical Mechanisms Details

A.1 Rebound Metrics Detailed Calculation

Rebound Window Definition.

$$[t_d, t_r] \tag{11}$$

Here t_d marks the first departure from nominal execution at anchor a_t, and t_r marks the first return to acceptable execution. At each planner step, the EAS evaluation layer checks execution events and task and proposition progress to identify these boundaries.

Recovery Cost Formalization. We characterize **recovery cost** through *cognitive effort*, *physical effort*, and *state debt*, using the local Stage Baseline $\mathcal{N}$ at a_t as the normalization reference:

$$\begin{aligned}
\mathbf{z}_t^{\mathrm{cog}} &= \left[\frac{(T_t^{\mathrm{per}} - \bar{T}^{\mathrm{per}}(a_t))_+}{\bar{T}^{\mathrm{per}}(a_t) + \epsilon}, \frac{(T_t^{\mathrm{plan}} - \bar{T}^{\mathrm{plan}}(a_t))_+}{\bar{T}^{\mathrm{plan}}(a_t) + \epsilon}, \frac{(T_t^{\mathrm{coord}} - \bar{T}^{\mathrm{coord}}(a_t))_+}{\bar{T}^{\mathrm{coord}}(a_t) + \epsilon} \right]^\top, \\
\mathbf{z}_t^{\mathrm{phy}} &= \left[\frac{(N_t^{\mathrm{nav}} - \bar{N}^{\mathrm{nav}}(a_t))_+}{\bar{N}^{\mathrm{nav}}(a_t) + \epsilon}, \frac{(N_t^{\mathrm{mpl}} - \bar{N}^{\mathrm{mpl}}(a_t))_+}{\bar{N}^{\mathrm{mpl}}(a_t) + \epsilon}, \frac{(N_t^{\mathrm{retry}} - \bar{N}^{\mathrm{retry}}(a_t))_+}{\bar{N}^{\mathrm{retry}}(a_t) + \epsilon} \right]^\top, \\
\mathbf{z}_t^{\mathrm{deb}} &= \left[\frac{(-\Delta p_t)_+}{\bar{\Delta p}(a_t) + \epsilon}, \frac{(-\Delta q_t)_+}{\bar{\Delta q}(a_t) + \epsilon}, \frac{(r_t - \bar{r}(a_t))_+}{\bar{r}(a_t) + \epsilon} \right]^\top.
\end{aligned} \tag{12}$$

We measure cognitive effort through perception feedback, replanning, and coordination events. For activity $c \in \{\mathrm{per}, \mathrm{plan}, \mathrm{coord}\}$, $\mathcal{A}_t^c$ is the set of active event identities. Newly activated events determine the activity count v_t^c and its excess over the stage reference:

Cognitive recovery cost. Following the information flow of ReAct execution, we organize cognitive recovery cost $\mathbf{z}_t^{\mathrm{cog}}$ into perception fusion (usually represents as world graph drifts), planning and decision-making (LLM planner agent context hallucination), and multi-agent collaboration (explicit information or implicit world graph fusion). For activity $c \in \{\mathrm{per}, \mathrm{plan}, \mathrm{coord}\}$, $\mathcal{A}_t^c$ is the set of active event identities. Newly activated events determine the activity count v_t^c and its excess over the stage reference:

$$v_t^c = \left| \mathcal{A}_t^c \setminus \mathcal{A}_{t-1}^c \right|, \qquad z_{t,c}^{\mathrm{cog}} = \frac{(v_t^c - \bar{v}^c(a_t))_+}{\max(|\bar{v}^c(a_t)|, 1)} (1 - \rho_t). \tag{13}$$

The weight ρ_t is the fraction of the next L available steps with task or proposition progress:

$$\mathcal{L}_t = \{t+1, \ldots, \min(t+L, T)\}, \quad \rho_t = \frac{\sum_{u \in \mathcal{L}_t} \mathbb{I}[\Delta p_u > \epsilon_p \vee \Delta q_u > \epsilon_p]}{\max(1, |\mathcal{L}_t|)}. \tag{14}$$

The recovering factor $(1 - \rho_t)$ assigns greater cost to excess activity followed by sparse progress.

Perception fusion. Agent i updates its world graph $\mathcal{G}_t^i$ from observations o_t^j and feedback y_t^j available from sources $j \in \mathcal{S}_t^i$:

$$\mathcal{O}_t^i = \{o_t^j\}_{j \in \mathcal{S}_t^i}, \qquad \mathcal{Y}_t^i = \{y_t^j\}_{j \in \mathcal{S}_t^i}, \qquad \mathcal{G}_t^i = \mathcal{F}_i\big(\mathcal{G}_{t-1}^i, \mathcal{O}_t^i, \mathcal{Y}_t^i\big), \tag{15}$$

where $\mathcal{F}_i$ updates entity states and relations. We record active macro-skill (as perception skill) tool response identities as $\mathcal{A}_t^{\mathrm{per}} = \mathrm{RespID}(\mathcal{Y}_t)$, where $\mathcal{Y}_t$ contains the recorded responses. New activations define v_t^{per} in Eq. 13 and serve as a proxy for feedback-processing effort.

Planning and decision-making. The updated world graph and execution feedback support revision of the agent's plan $\mathcal{P}_t^i$ and selection of next action $\hat{u}_t^i$:

$$\mathcal{P}_t^i = \mathcal{R}_i\big(\mathcal{P}_{t-1}^i, \mathcal{G}_t^i, \mathcal{Y}_t^i\big), \qquad \hat{u}_t^i = \mathrm{Next}(\mathcal{P}_t^i), \tag{16}$$

Here, $\mathcal{R}_i$ revises the plan using execution feedback. The set $\mathcal{A}_t^{\mathrm{plan}}$ contains active agent-level replanning identities, with planning-transition labels used as a fallback. Their new activations define v_t^{plan}, which is compared with the stage reference.

Multi-agent collaboration in EAS. Under partial observability (POMDP exhibits in each sub-trajectory), agent i forms local decisions from its observation-action history h_t^i and available shared information. Coordination uses explicit messages and shared task and world-state information.

Explicit messages convey intentions, task assignments, and feedback. Let $m_t^{j\to i}$ denote context sent by agent j or the team planner to agent i. The implicit context consists of the shared instruction ℓ and the local world graph $\mathcal{G}_t^i$, including observed outcomes of other agents' actions:

$$\mathcal{C}_t^{i,\mathrm{exp}} = \{m_t^{j\to i}\}_{j\in\mathcal{S}_t^i}, \qquad \mathcal{C}_t^{i,\mathrm{imp}} = (\ell, \mathcal{G}_t^i). \tag{17}$$

Here, the instruction supplies a common task objective, while cross-agent graph updates expose changes in object locations, holding states, and action outcomes. These changes inform task allocation and execution dependencies even when agents exchange no dedicated coordination message.

Within $[t_d, t_r]$, the joint policy uses the histories, capabilities, and coordination contexts collected in $\mathcal{I}_t^{\mathrm{team}}$:

$$\mathbf{u}_t = (u_t^1, \ldots, u_t^n) \sim \pi_{\mathrm{team}}(\cdot \mid \mathcal{I}_t^{\mathrm{team}}). \tag{18}$$

We record coordination through designated (collaboration) macro skills $\mathcal{U}^{\mathrm{coord}}$ and their active agent–skill identities:

$$\mathcal{A}_t^{\mathrm{coord}} = \left\{(i, u_t^i) \;\middle|\; i \in \mathcal{I},\ u_t^i \in \mathcal{U}^{\mathrm{coord}}\right\}. \tag{19}$$

Here, $\mathcal{I}$ is the agent set. Newly activated identities define v_t^{coord}, a proxy for coordination workload.

Physical recovery cost. We measure physical recovery through navigation, manipulation, and retry or backtracking activity within $[t_d, t_r]$. Let $\mathcal{E}_t^d$ contain the distinct agent–skill execution records newly logged between consecutive observations for component $d \in \{\mathrm{nav}, \mathrm{mpl}, \mathrm{retry}\}$. Each record identifies an execution attempt, so an ongoing attempt is counted once even when it appears in several observations. The component counts and their stage-normalized excesses are

$$b_t^d = \max(\bar{N}^d(a_t), 1), \qquad e_t^d = \frac{(N_t^d - b_t^d)_+}{b_t^d}, \qquad d \in \{\mathrm{plan}, \mathrm{sim}\}. \tag{20}$$

Planning steps count execution-loop iterations, and simulator steps count the simulated work between observations. We mark stagnation when all recorded progress signals are at or below their thresholds:

$$I_t^{\mathrm{stall}} = \mathbb{I}[\Delta p_t \le \epsilon_p \wedge \Delta q_t \le \epsilon_p \wedge \Delta \varsigma_t \le \epsilon_p], \tag{21}$$

where ς_t is the recorded task-success indicator.

Navigation recovery cost. Navigation macro skills $\mathcal{U}^{\mathrm{nav}}$ move agents toward task objects, receptacles, or regions. Their presence in the joint execution is indicated by

$$I_t^{\mathrm{nav}} = \mathbb{I}\left[\exists i \in \mathcal{I} : u_t^i \in \mathcal{U}^{\mathrm{nav}}\right]. \tag{22}$$

Let $D_{t,w}^{\mathrm{nav}}$ indicate spatial oscillation or deviation detected at step t or within recovery window $[t_d, t_r]$, and the corresponding execution burden is

$$\gamma_t^{\mathrm{nav}} = I_t^{\mathrm{nav}} I_t^{\mathrm{stall}} D_{t,w}^{\mathrm{nav}} \frac{N_t^{\mathrm{sim}}}{b_t^{\mathrm{sim}}}. \tag{23}$$

This term measures stage-normalized simulator work when navigation, stagnation, and spatial-recovery evidence coincide.

Manipulation execution recovery cost. Manipulation macro skills $\mathcal{U}^{\mathrm{mpl}}$ change object placement, possession, or attributes. Their execution indicator is

$$I_t^{\mathrm{mpl}} = \mathbb{I}\left[\exists i \in \mathcal{I} : u_t^i \in \mathcal{U}^{\mathrm{mpl}}\right]. \tag{24}$$

Let $D_{t,w}^{\mathrm{mpl}}$ mark a placement-target mismatch, a required state change left incomplete, or a pick–place cycle with stalled progress. The manipulation stagnation load is

$$\gamma_t^{\mathrm{mpl}} = \alpha_{\mathrm{mpl}} I_t^{\mathrm{mpl}} I_t^{\mathrm{stall}} D_{t,w}^{\mathrm{mpl}} \frac{N_t^{\mathrm{sim}}}{b_t^{\mathrm{sim}}}, \quad \alpha_{\mathrm{mpl}} = \frac{1}{2}. \tag{25}$$

The coefficient α_{mpl} is predefined manipulation-gap weight used in our implementation (which is easier for calculation).

Let $\mathcal{E}_t^{\mathrm{retry}}$ contain retry and backtrack events at step t. We mark these intervals with $R_t = \mathbb{I}[\mathcal{E}_t^{\mathrm{retry}} \neq \varnothing]$ and retain their workload in the execution counters.

Retrying and backtracking. During the recovery windows, repeated execution can be executed for instructed task purpose. Let $\mathcal{E}_t^{\text{retry}}$ contain the recorded retry and backtrack events at step t, we mark these intervals with $R_t = \mathbb{I}[\mathcal{E}_t^{\text{retry}} \neq \varnothing]$ and retain their workload in the execution counters. Notice, this retry activities are often taken place in embodied execution, but for rebound recovery, we mainly take the recovering progress (usually in recovery windows) into account.

We also account for idle execution when an agent waits without forward progress, I_t^{wait} for the presence of a waiting skill, and the total stagnation load is

$$\gamma_t^{\text{gap}} = \gamma_t^{\text{nav}} + \gamma_t^{\text{mpl}} + I_t^{\text{wait}} I_t^{\text{stall}} \max\left(\frac{N_t^{\text{plan}}}{b_t^{\text{plan}}}, \frac{N_t^{\text{sim}}}{b_t^{\text{sim}}}\right). \tag{26}$$

Embodied execution state recovery debt. During the recovery time, usually EAS went into the state where some errors had been made. Since physical execution actions can not be recalled, we define the state recovery debt to describe how we quantify this effort. Here state debt measures lost task progress, lost proposition satisfaction, and excess risk during recovery.

Task-progress regression. Progress loss is $d_t^p = (-\Delta p_t)_+$, where $\Delta p_t = p_t - p_{t-1}$.

Proposition-satisfaction regression. Proposition loss is $d_t^q = (-\Delta q_t)_+$, where $\Delta q_t = q_t - q_{t-1}$ and q_t is the fraction of satisfied propositions.

Excess execution risk. For nonnegative risk r_t and its stage reference $\bar{r}(a_t)$, the excess-risk component is

$$d_t^r = \begin{cases} \dfrac{(r_t - \bar{r}(a_t))_+}{\bar{r}(a_t)}, & \bar{r}(a_t) > 0, \\ r_t, & \bar{r}(a_t) = 0. \end{cases} \tag{27}$$

Together, these components yield

$$\mathbf{z}_t^{\text{deb}} = \left[\frac{d_t^p}{\max\big(|\bar{\Delta}p(a_t)|, \epsilon_b\big)}, \frac{d_t^q}{\max\big(|\bar{\Delta}q(a_t)|, \epsilon_b\big)}, d_t^r\right]^\top, \tag{28}$$

where $\epsilon_b = 10^{-12}$ (empirical values) provides positive normalization scales.

Rebound Statistics Aggregation and Signature Capture. We sum window costs within each episode, then report the family mean and upper quantile. Let e, w, and k index episodes, recovery windows, and EAS versions $\pi^{(k)}$, respectively:

$$\begin{aligned} C_{\text{rec},e}^{(k)} &= \sum_{w \in \mathcal{W}_e^{(k)}} C_{\text{rec},ew}^{(k)}, \\ \bar{C}_{\text{rec},f}^{(k)} &= \frac{1}{N_f^{(k)}} \sum_{e \in \mathcal{E}_f^{(k)}} C_{\text{rec},e}^{(k)}, \\ Q_{1-\alpha,f}^{(k)} &= \text{Quantile}_{1-\alpha}\left(\left\{C_{\text{rec},e}^{(k)}\right\}_{e \in \mathcal{E}_f^{(k)}}\right), \end{aligned} \tag{29}$$

where $C_{\text{rec},ew}^{(k)}$ denotes window-level recovery cost and $0 < \alpha < 1/2$ is fixed across comparisons, then comes to episode level and task-family level. The family signature and its change between versions are

$$\mathbf{R}_{\text{reb},f}^{(k)} = \left[\bar{C}_{\text{rec},f}^{(k)}, Q_{1-\alpha,f}^{(k)}\right]^\top, \qquad \Delta\mathbf{R}_{\text{reb},f}^{(k)} = \mathbf{R}_{\text{reb},f}^{(k)} - \mathbf{R}_{\text{reb},f}^{(k-1)}. \tag{30}$$

With a common task distribution and protocol, negative changes indicate lower mean or upper-tail recovery cost. We report valid episode counts and censored fractions with each signature.

A.2 Stability Metrics Detailed Calculation

Stability describes how consistently a system progresses and plans during execution, different from (ML) robustness, which describes how well a specified property is preserved under perturbations [37]. Our metric focuses on the EAS execution dynamics, clean–stress comparisons test their

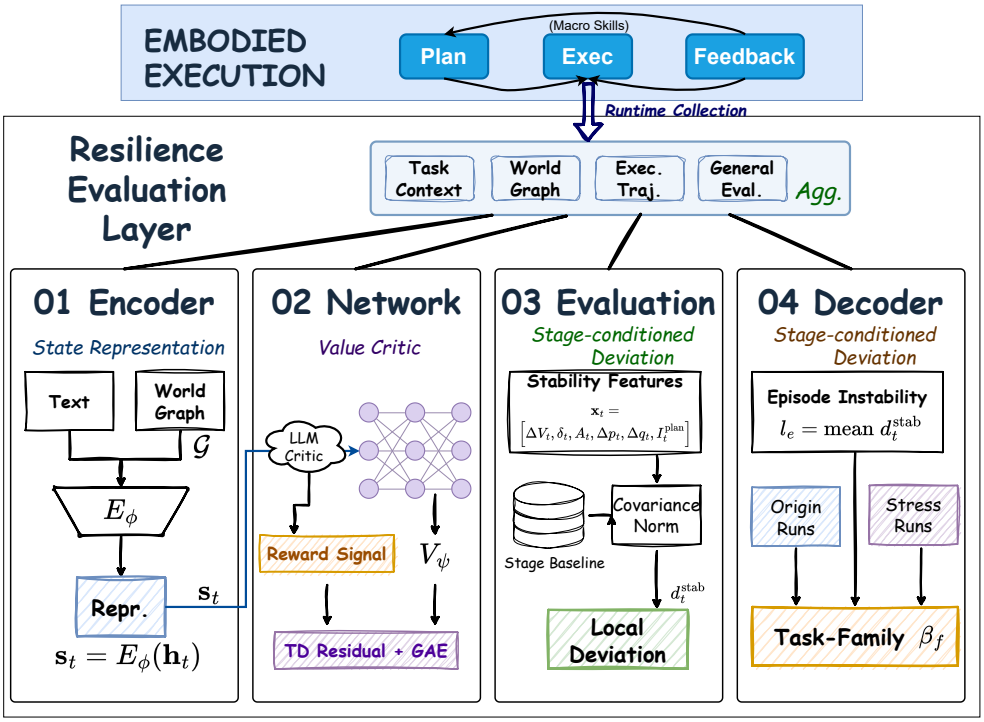


Figure 7: **Stability computation.** Execution signals are compared with Stage Baselines to obtain episode losses, whose clean–stress differences measure disruption sensitivity.

consistency, complementing outcome-based robustness evaluation. Figure 7 shows how execution evidence becomes an episode loss and then a sensitivity estimate from clean–stress comparisons.

State Encoding and Critic Signals. The StateEncoder maps task context, world-graph information, recent actions and feedback, and task progress (from resilience evaluation aggregation layer) to an execution state:

$$\mathbf{h}_t = \left[\mathbf{h}_t^{\text{task}}, \mathbf{h}_t^{\text{world}}, \mathbf{h}_t^{\text{event}}, \mathbf{h}_t^{\text{prog}}\right], \quad s_t = E_\phi(\mathbf{h}_t). \tag{31}$$

The value critic $V_\psi(s_t)$ estimates the discounted remaining execution utility using reward $\tilde{r}_t$, including LLM-judged feedback when reward shaping is enabled. It provides obtain value changes, temporal-difference (TD) residuals, and generalized advantage estimates (GAE):

$$\begin{aligned} \Delta V_t &= V_\psi(s_t) - V_\psi(s_{t-1}), \\ \delta_t &= \tilde{r}_t + \gamma(1 - d_t^{\text{term}})V_\psi(s_{t+1}) - V_\psi(s_t), \\ A_t &= \delta_t + \gamma\lambda_{\text{GAE}}(1 - d_t^{\text{term}})A_{t+1}, \quad A_{T+1} = 0. \end{aligned} \tag{32}$$

Here, d_t^{term} indicates episode termination, γ is the discount factor, and λ_{GAE} controls residual accumulation. Appendix A.4 details the encoder and critic.

Stage-Conditioned Instability and Episode Loss. We combine critic signals with task progress increments Δp_t, proposition-completion increments Δq_t, and a planning or replanning indicator I_t^{plan}:

$$\mathbf{x}_t = \left[\Delta V_t,\, \delta_t,\, A_t,\, \Delta p_t,\, \Delta q_t,\, I_t^{\text{plan}}\right]^\top. \tag{33}$$

Clean trajectories (stress-free) provide the mean $\mu(a_t)$ and covariance $\Sigma(a_t)$ at anchor a_t. The local deviation is

$$\widetilde{\mathbf{x}}_t = \mathbf{x}_t - \mu(a_t), \qquad d_t^{\text{stab}} = \sqrt{\widetilde{\mathbf{x}}_t^\top \left(\Sigma(a_t) + \eta I\right)^{-1} \widetilde{\mathbf{x}}_t}, \tag{34}$$

where $\eta > 0$ regularizes the covariance and I is the identity matrix. The distance accounts for stage-specific variability and feature correlations.

Algorithm 2 Stability computation from paired execution trajectories

Require: Clean–stress pairs $\mathcal{D}_f$; fixed E_ϕ, V_ψ, and Stage Baseline $\mathcal{N}$.

Ensure: Episode losses ℓ_i, oscillation diagnostic β_{step}, and family sensitivity $\widehat{\beta}_f$.

for all distinct trajectories z_i in the paired set **do**
  Order records by step; $s_t \leftarrow E_\phi(\mathbf{h}_t)$ (Eq. 31).
  Cache $\{V_\psi(s_t)\}_t$ for the full trajectory.
  Compute ΔV_t and δ_t (Eq. 32).
  $A_{T+1} \leftarrow 0$; compute A_t backward (Eq. 32).
  for all eligible transitions t **do**
    Resolve a_t; retrieve $\mu(a_t)$ and $\Sigma(a_t)$ from $\mathcal{N}$.
    Assemble $\mathbf{x}_t$ and compute d_t^{stab} (Eqs. 33–34).
  end for
  $\ell_i \leftarrow \text{mean}_t\, d_t^{\text{stab}}$; store loss and coverage (Eq. 35).
end for
for all pairs $j \in \mathcal{D}_f$ **do**
  $b_j \leftarrow$ disruption-normalized absolute difference of stored losses (Eq. 36).
end for
$\beta_{\text{step}} \leftarrow \beta_{\text{vv}}$: aggregate normalized clean–stress critic-oscillation differences.
$\widehat{\beta}_f \leftarrow \max_{j \in \mathcal{D}_f} b_j$.

For episode z_i under policy π, we average over T_i eligible transitions:

$$\ell(\pi, z_i) = \frac{1}{T_i} \sum_{t=1}^{T_i} d_t^{\text{stab}}. \tag{35}$$

Eligible transitions have complete features and an available Stage Baseline. Larger losses indicate greater average deviation from that baseline, we record transition coverage with each loss.

Task-Family Disruption Sensitivity. For statistical analysis, we measure task-family sensitivity through the change in episode loss per unit disruption,

$$\beta_f = \sup_{z \in \mathcal{Z}_f,\, \rho \in \mathcal{P}_f} \frac{|\ell(\pi, \rho(z)) - \ell(\pi, z)|}{\|\rho\| + \epsilon}. \tag{36}$$

Here $\mathcal{Z}_f$ is the episode set and $\mathcal{P}_f$ is the set of disruptions that preserve the task objective. The perturbed instance is $\rho(z)$, $\|\rho\|$ indicates disruption magnitude, and the constant $\epsilon > 0$ stabilizes the denominator. Appendix B.3 details the perturbation protocols and severity calibration. We estimate $\widehat{\beta}_f$ as the largest normalized loss difference.

A smaller $\widehat{\beta}_f$ indicates greater consistency across the tested disruptions. Episode loss measures instability within a run, and $\widehat{\beta}_f$ captures its sensitivity across runs. Algorithm 2 gives the computation order.

A.3 Graceful Extensibility Metrics Detailed Calculation

Graceful Extensibility (GE) concerns how an EAS extends its capacity to adapt when demands challenge its operational limits [38]. For each task family f, we map stress severity λ to an operational margin $M_f(\lambda)$ using execution trajectory and evaluation thresholds:

$$m_i(\lambda) \;\rightarrow\; \mathbf{r}_f(\lambda) \;\rightarrow\; M_f(\lambda) \;\xrightarrow{\text{stress sweep}}\; \lambda_f^*.$$

Here, $\mathbf{r}_f(\lambda)$ summarizes family-level evaluation, and $M_f(\lambda)$ characterizes the degradation pattern, while the stress capacity λ_f^* gives the highest tested severity with $M_f(\lambda) \geq 0$.

Beyond stress-response, graceful extensibility focuses on whether EAS adaptive planning and coordination expand the conditions under which execution remains acceptable. Margin gains provide more room to absorb disturbances, while boundary extension shows that EAS resilience enables acceptable execution under stronger stress.

Stress Specification and Evaluation Conditions. For each task family f, we evaluate a fixed EAS version under a chosen stress operator $\mathcal{T}_\lambda$ stt. Severity indexes six prescribed intervention

levels (easy for calculation):

$$\lambda_j = j/5, \quad j = 0, \ldots, 5, \quad \Lambda = \{0, 0.2, 0.4, 0.6, 0.8, 1.0\}. \tag{37}$$

Level 0 indicates the clean reference. Under distractor-count stress, level j adds specific irrelevant nodes to the planner's scene representation, planning context or collaboration. Appendix B.3 gives stress schedules and the interface on the applied perturbations, inner stresses are not included because of uncertainty (we analyze here through a sufficient number of repeated runs).

Execution Trajectory and EAS Aggregation. During resilience EAS execution for each episode at stress λ, each episode supplies five measures from task monitors and execution logs $\big(p_{\mathcal{T},i}(\lambda),\, q_{\mathcal{T},i}(\lambda),\, C_{\mathrm{rec},i}(\lambda),\, \beta_i(\lambda),\, U_i(\lambda)\big)$. This measure task completion, proposition satisfaction, recovery cost, stability load, and constraint-violation load, recovery costs (Eq. 29) and stability load (Eq. 36), evaluated at the current λ.

Within each task family f and stress level λ, we group valid episodes from the same EAS version and stress protocol. For $0 < \alpha < 1/2$, we summarize completion and satisfaction by their lower α-quantiles, and recovery, stability, and safety loads by their upper $(1-\alpha)$-quantiles.

$$\begin{aligned} p_f(\lambda) &= Q^f_\alpha[p_T](\lambda), & q_f(\lambda) &= Q^f_\alpha[q_T](\lambda), \\ c_f(\lambda) &= Q^f_{1-\alpha}[C_{\mathrm{rec}}](\lambda), & v_f(\lambda) &= Q^f_{1-\alpha}[S_{\mathrm{stab}}](\lambda), \\ u_f(\lambda) &= Q^f_{1-\alpha}[U](\lambda). \end{aligned} \tag{38}$$

Here, $Q^f_\gamma[X](\lambda)$ is the empirical γ-quantile of episode measure X within the group. The resulting family statistics are $\big(p_f(\lambda),\, q_f(\lambda),\, c_f(\lambda),\, v_f(\lambda),\, u_f(\lambda)\big)$.

Resilience Extensibility and Stress-Response Mapping. Figure 8 illustrates distractor-count stress: added world-graph stress alter the planner's input while the placement goal remains fixed. We normalize repeated statistics against the evaluation thresholds and compute the operational margin using Eqs. 8 and 9. Evaluating all levels in Λ yields the sampled stress-response mapping $\lambda \mapsto M_f(\lambda)$.

Nonnegative margin indicates that the weakest K normalized dimensions meet their targets on average. The resulting profile provides the basis for identifying stress capacity and characterizing degradation in the next paragraph.

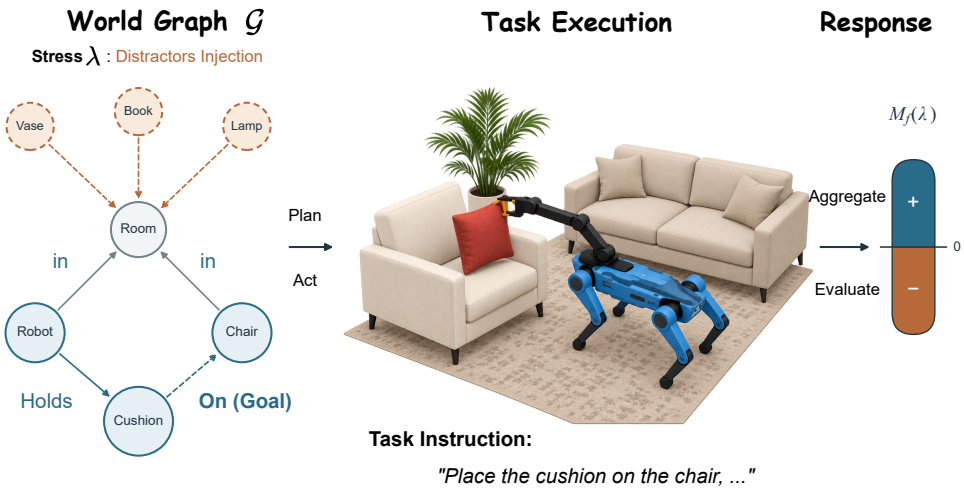


Figure 8: **Stress-response mechanism.** World graph nodes mark added distractors, while relations describe the current holding state and placement goal. Evidence from repeated executions yields the task-family operational margin $M_f(\lambda)$.

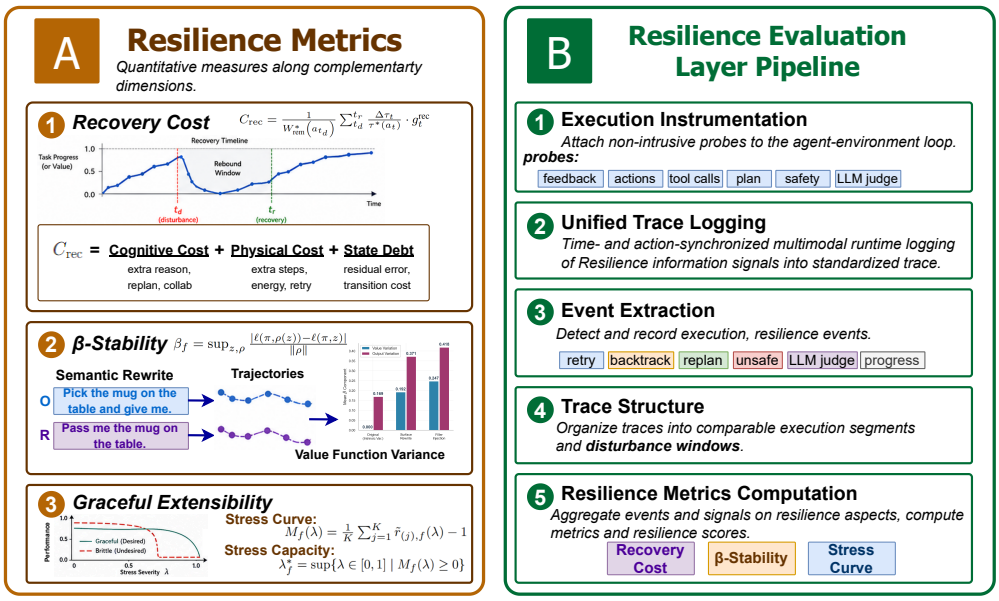


Figure 9: **Overview of our EAS Resilience Evaluation framework**. (A) illustrates resilience metrics concept and their computation, and (B) illustrates the resilience evaluation layer pipeline.

Stress Capacity and EAS Graceful Extensibility Profile. Following Eq. 10, we estimate stress capacity λ_f^* on the sampled grid Λ as the highest tested severity with $M_f(\lambda) \geq 0$. This gives the observed capacity for task family f under the chosen stress protocol; if every sampled margin is negative, the capacity is undefined.

To assess graceful extensibility, we compare this capacity and the associated response profile with adaptive planning and coordination enabled and disabled in the same EAS, holding tasks, stress levels, execution budgets, and evaluation thresholds fixed. Stress levels where $M_{f,\mathrm{on}}(\lambda) \geq 0 > M_{f,\mathrm{off}}(\lambda)$ identify newly acceptable conditions, and an increase from $\lambda^*_{f,\mathrm{off}}$ to $\lambda^*_{f,\mathrm{on}}$ indicates extension of the tested capacity. Higher margins where both configurations remain acceptable capture additional operating headroom, giving the two adaptive gains we have shown in Fig. 2.

The full response profile also captures performance throughout the stress sweep. For ordered stress levels $\lambda_1 < \cdots < \lambda_m$, we compute the area under the shifted, nonnegative margin curve as

$$\mathrm{AUDC}_f = \sum_{j=2}^{m} \frac{y_f(\lambda_j) + y_f(\lambda_{j-1})}{2} (\lambda_j - \lambda_{j-1}), \quad y_f(\lambda) = \max\left(M_f(\lambda) + 1, 0\right). \tag{39}$$

Here, AUDC_f summarizes overall performance over a common stress range, while local margin drops identify abrupt degradation, complementing the capacity and margin comparisons in the GE profile.

In a word, we express our EAS Resilience Evaluation framework as Fig 9.

A.4 Value Function Implementation

Providing critic-side statistical traces for Stability and GE diagnostics, our evaluation layer uses an auxiliary critic to estimate the discounted reward remaining in an execution. It encodes task context and execution feedback into a latent state, then produces a scalar value and intermediate features whose temporal variation supports Stability analysis and the stability component of GE evaluation.

State Encoding and Network Architecture. The critic observation $o_t^c = (\mathcal{W}_t, \mathcal{I}_t)$ combines a structured world-state dictionary, including task context, with auxiliary execution information. The StateEncoder represents this observation through complementary text and numerical branches, then

fuses them into a 128-dimensional state:

$$s_t = E_\phi(o_t^c) = F_\phi\big([m_{\text{text}}\mathbf{e}_t^{\text{text}};\, m_{\text{num}}\mathbf{e}_t^{\text{num}}]\big) \in \mathbb{R}^{128}. \tag{40}$$

The binary masks $(m_{\text{text}}, m_{\text{num}})$ select full, text-only, or numerical-only input. Table 4 summarizes the feature construction and network dimensions. Text segments are concatenated into a description, truncated after 512 characters with an ellipsis when needed, and encoded by a frozen sentence transformer; a deterministic hash encoder is available as a fallback. Numerical features use fixed normalization constants and are padded or truncated to 32 dimensions.

Table 4: Critic inputs and architecture. Each hidden block applies a linear map, a rectified linear unit (ReLU), layer normalization, and dropout in that order.

Component	Dimensions	Content / operation	Output
Text branch	384	Task and scene context; execution status and feedback	$\mathbf{e}_t^{\text{text}}$
Numerical branch	32	Geometry and progress; counts and interaction state	$\mathbf{e}_t^{\text{num}}$
Fusion network	$416 \to 256 \to 128$	One hidden block; linear map + ReLU	s_t
Value features	$128 \to 256 \to 128 \to 64$	Three hidden blocks	$\mathbf{g}_t$
Value head	$64 \to 1$	Linear projection	$\widehat{V}_\psi(s_t)$

Value Estimation and Training. The value network maps s_t to the expected discounted sum of critic rewards,

$$\widehat{V}_\psi(s_t) \approx \mathbb{E}\left[\sum_{u=t}^{T} \gamma^{u-t}\tilde{r}_u \,\middle|\, s_t\right], \tag{41}$$

where $\tilde{r}_u$ includes the LLM reward contribution when shaping is enabled. Its scalar output provides the temporal difference (TD) residual and generalized advantage estimation (GAE) target. For an episode of T transitions, with terminal indicator d_t and $A_{T+1} = 0$, these are computed as

$$\delta_t = \tilde{r}_t + \gamma(1-d_t)\widehat{V}_\psi(s_{t+1}) - \widehat{V}_\psi(s_t), \tag{42}$$

$$A_t = \delta_t + \gamma\lambda_{\text{GAE}}(1-d_t)A_{t+1}, \tag{43}$$

$$R_t = A_t + \widehat{V}_\psi(s_t). \tag{44}$$

GAE is computed over the recorded episode, and the resulting targets are held fixed during each value update:

$$\mathcal{L}_V = \frac{1}{T}\sum_{t=1}^{T}\left(\widehat{V}_\psi(s_t) - R_t\right)^2. \tag{45}$$

Table 5 gives the evaluation runner's configurable training and measurement settings. During evaluation, the encoder and value network use predefined parameters and inference (we usually turn off dropout). These example settings for Stage Baseline initialization should be adapted to the target EAS and evaluation tasks, see our reproduction section for more details.

Table 5: Critic configuration template used by the evaluation runner.

Parameter	**Value**
Discount factor γ	0.80
GAE decay λ_{GAE}	0.65
Optimizer	Adam
Learning rate	10^{-3}
Training dropout	0.1
Gradient norm cap	1.0
Distance scale	10
Count scale	10
History cap	50

Resilience Evaluation Deployments. The critic exposes both $\widehat{V}_\psi(s_t)$ and the 64-dimensional feature vector $\mathbf{g}_t$ to characterize changes along the execution. The implementation combines value

change, TD residual, GAE magnitude, and feature change into an episode-level oscillation proxy:

$$\mathbf{o}_t = [|\Delta V_t|,\ |\delta_t|,\ |A_t|,\ \|\Delta \mathbf{g}_t\|_2]\,,$$
$$\bar{\ell}^{\mathrm{osc}} = \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{w}_{\mathrm{osc}} \odot \mathbf{o}_t\|_2, \qquad \mathbf{w}_{\mathrm{osc}} = [1, 1, 1, 0.25]. \tag{46}$$

Here ΔV_t and $\Delta \mathbf{g}_t$ denote changes from the preceding step, with zero change at the first step. This proxy supplies critic-side evidence for the comparison of clean and stress executions, and GE evaluation incorporates the resulting Stability measure alongside task outcomes and recovery costs in Sec. A.2 and Sec. A.3.

B Dataset Illustration

We evaluate how an embodied agent system (EAS) executes embodied ousehold tasks under controlled input changes, the structured episodes follow the PARTNR task representation [6], with shared initial scenes and executable goals providing a common reference for clean and stress runs. This section describes the dataset structure, generation protocol, stress setting, and resilience metrics evaluated comparisons. Dataset generation code and data are available at the GitHub Repo.

B.1 Dataset Structure and Experiments Components

Episode Components. Each structured episode combines a task instruction, an initial simulator state, evaluation rules, and metadata:

$$e_i = (\iota_i, x_i^0, \mathcal{E}_i, \mu_i), \qquad \mathcal{E}_i = (\mathcal{P}_i, \mathcal{D}_i, \mathcal{C}_i). \tag{47}$$

Here, ι_i is the natural language task instruction, x_i^0 specifies the scene and object states, and μ_i records episode identity and entity mappings. The evaluation rules combine goal propositions $\mathcal{P}_i$ with dependencies $\mathcal{D}_i$ that determine when they are evaluated and constraints $\mathcal{C}_i$ on their ordering, entity assignments, and terminal satisfaction (spatial, sequential, temporal and multi-robots constraints).

For example, placing two objects next to each other on a table requires both destination and relative-placement predicates, while terminal-satisfaction constraints specify which goals must hold at the end of execution. These rules remain fixed across conditions, giving clean and stressed runs a common definition of completion.

Dataset Tasks Composition. Our evaluation suite contains 400 tasks, comprising 369 structured validation episodes and 31 instruction tasks without constraints. The structured pool spans six scenes and encodes 1,683 machine-checkable goal propositions, whose coverage and distribution are shown in Fig. 10.

The evaluation structures further capture different aspects of task execution: 179 episodes require relative placement, 199 include proposition dependencies, 138 include active temporal constraints, and 14 episodes impose entity-consistency constraints that require propositions to share or distinguish their object assignments. Together, the proposition-count distribution and scene breakdown describe the size and environmental coverage of the task specifications.

Experiments Datasets. The completed study uses the full embodied task set for benchmarking and episode subset of the structured pool for controlled mechanism studies. Within these studies, metric validation and diagnosis-guided optimization each evaluate serveral episodes with 3 repetitions, as summarized in Table 6. In addition, Graceful Extensibility (GE) comparisons evaluate matched tasks at a clean reference and stress levels.

B.2 Datasets Generation

Following PARTNR episodes construction and effects in Habitat-Sim 3.0, our resilience evaluation datasets generation workflow uses LLMs to propose tasks and goal propositions, then assembles the episodes and checks their initialization in the simulator. Figure 11 traces this process from scene-grounded proposals to verified exports. We provide the datasets generation method, and engineers could test their embodied tasks on Habitat-Sim 3.0.

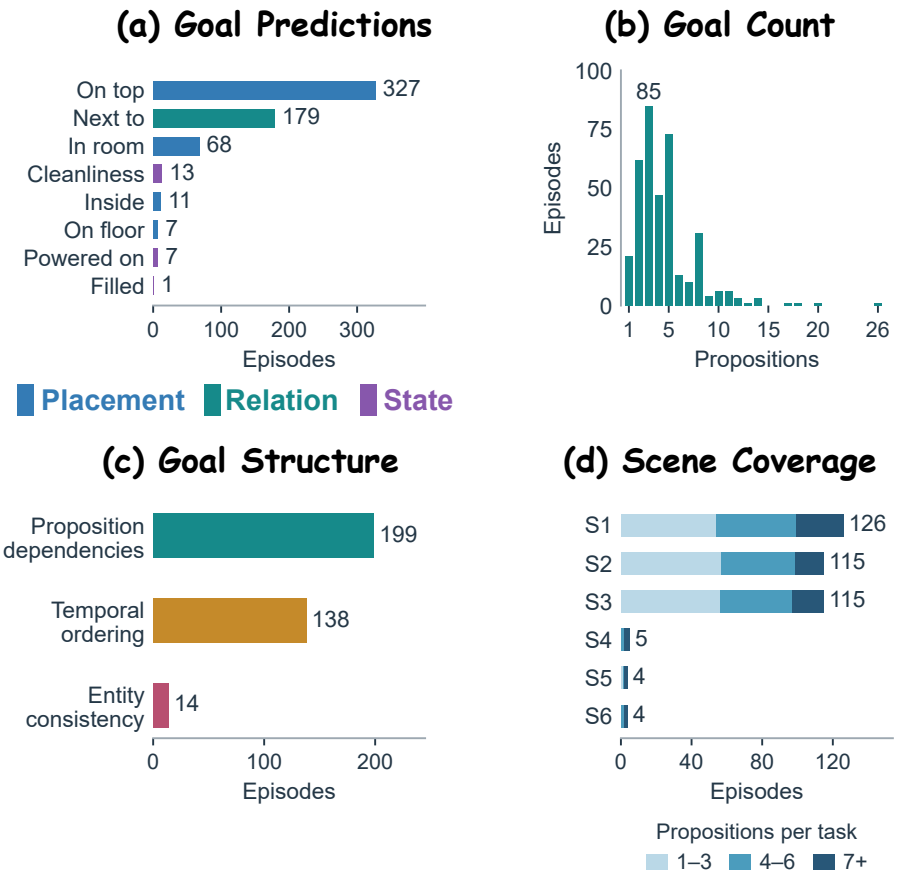


Figure 10: **Composition of the dataset episodes structured pool.** The panels summarize goal-predicate coverage (a), propositions per episode (b), evaluation structures (c), and scene coverage by proposition-count band (d).

Table 6: **Completed experimental allocation.** Episode counts refer to distinct tasks within each experiment, while repetitions, stress levels, and configuration comparisons contribute additional executions. Controlled studies draw on the 193-episode mechanism subset.

Experiment	Task allocation	Comparison purpose
Validation (Exp-Q1)	45 episodes; 3 repetitions; GE: 6 levels including clean	Distinguish recovery, stability, and stress response among executions with similar outcomes.
Benchmark (Exp-Q2)	400 tasks	Compare EAS resilience profiles under common task and measurement conditions.
Optimization (Exp-Q3)	45 episodes; 3 repetitions; original/optimized variants for 3 resilience targets	Evaluate diagnosis-guided optimization under matched tasks and interventions.

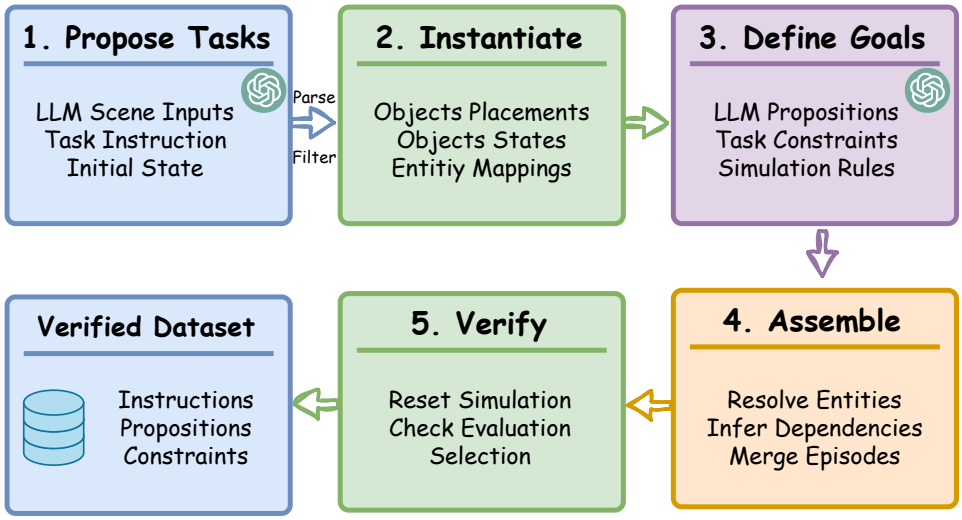


Figure 11: **Dataset generation and verification.** Scene information grounds task proposals, which are parsed and filtered before scene instantiation, evaluation-rule construction, dataset assembly, and initialization checks.

Stage I: LLM Instruction Proposal. The instruction prompt combines the scene's room–furniture inventory, available object categories, task examples, and a JSON output schema. In guided generation, a reference task supplies the spatial, temporal, or object-state requirements, and the LLM adapts its objects and furniture to the selected scene. Object-state prompt variants additionally request actions such as cleaning, filling, and switching power.

The LLM returns a JSON list whose entries pair an `instruction` with the `initial_state` specification containing object counts, categories, furniture, and allowed rooms. Parsing checks these fields against the scene metadata, and filtering removes unsupported or underspecified steps and duplicate instructions before scene instantiation.

Stage II: Simulation Scene Instantiation. The episode generator uses each retained initial-state specification to sample object assets and placements within the designated furniture and rooms, then applies the specified object states in the simulator. The resulting episode records store the realized scene configuration and entity mappings, providing the concrete initial state used to generate evaluation rules.

Stage III: LLM Evaluation Generation. The proposition prompt combines the generated instruction and instantiated scene description with a predefined predicate vocabulary and examples of instruction-to-predicate conversion. It asks the LLM to express completion conditions as a parsable list of predicate calls, such as `is_on_top`, `is_next_to`, and `is_clean`, using the scene's entity names as arguments. Temporal and entity-consistency prompt templates use the instruction and propositions to specify ordering and shared or distinct object assignments, while multi-robots heterogeneous constraints and propositions sequential constraints condition the EAS task collaboration. These outputs form the editable evaluation specifications passed to assembly.

Stage IV: Episode Assembly. Assembly resolves semantic names to simulator handles and combines the evaluation specifications with the episode's initial state and metadata. Declared dependency lists are retained, empty lists receive inferred dependencies, and terminal-satisfaction requirements are updated from the resulting dependencies. The assembled episodes are then merged into a dataset with unique identifiers while retaining their source identifiers and metadata.

Stage V: Tasks Initialization Verification. The verification routine resets each assembled episode, checks referenced entities and constraint arguments, and queries initial completion and success mea-

sures. Episodes that pass these checks enter the verified export, with initialization outcomes and exceptions recorded in the verification logs.

B.3 Stress Setting

Stress conditions specify an input change, its realized dose, and the controls used to compare executions. This protocol assigns an intervention, verifies the change presented to the planner, and measures the resulting execution, following the mechanism and two ordered stress schedules in Fig. 12.

B.3.1 Intervention Targets and Execution Mechanism

Intervention Boundary. The intervention operators modify the instruction or initial world graph supplied to the planner while holding the simulator state and task evaluator fixed at injection (Table 7). Because the instruction expresses the goal and the graph represents the entities and states used for planning, a text edit can alter goal interpretation or attention, whereas a graph mutation can create a mismatch between the planner's belief and the simulated scene. These input changes affect the physical execution through the planner's subsequent actions, as illustrated in Fig. 12a. Later perception updates may correct altered beliefs, and the associated actions and recovery events contribute to the measured response.

Table 7: **Intervention operators and acceptance evidence.** Each positive intervention requires an observable change in the input received by the planner. Room and possession changes use the corresponding graph-adapter operations when supported.

Operator	Changed planner input	Acceptance evidence
Rewrite, filler	Instruction wording or irrelevant text	Recorded edit and visible text change with the task goal retained.
State, room, or possession	Attribute of task-relevant World Graph nodes	Resolved target and verified mutation across the affected graph views.
Distractor addition	Irrelevant labels in the scene description	Observed label count and increased planner-visible description length.
Instruction/context sequence	Prepared rewrites and cumulative context	Selected variant, preserved task meaning, assigned level, and calibrated text dose.
Multi-robot coordination	Task allocation, per-robot world graphs, or inter-robot communication	Affected robot IDs and verified changes in assignments, graph views, or message content and delivery.

Stress Levels and Calibration. We compare clean and perturbed executions of the same task under matched conditions. Both ordered stress protocols use $\Lambda = \{\lambda_j = j/5 : j = 0, \ldots, 5\}$, with $\lambda_0 = 0$ as the clean reference. Distractor-count stress adds j task-irrelevant labels at level j, while instruction/context stress applies two successive goal-preserving rewrites followed by three cumulative context additions. When room, furniture, and object context are all available, they are added in that order; otherwise, the stored sequence specifies the available context units.

For each episode and perturbation seed, the same text sequence is reused across levels and compared configurations. Text sequences are checked for goal preservation and calibrated against the assigned grid before evaluation, while distractor interventions are verified through the added labels and growth in the planner-visible scene description.

Stress Operator Families and Comparison Experiments. We compare clean and perturbed executions of the same task under matched conditions. A Boolean state toggle provides a single perturbed condition with realized dose one, while distractor counts and instruction/context sequences allow stress to increase over several levels. Comparing these levels against the clean reference shows how execution changes along the prescribed stress path.

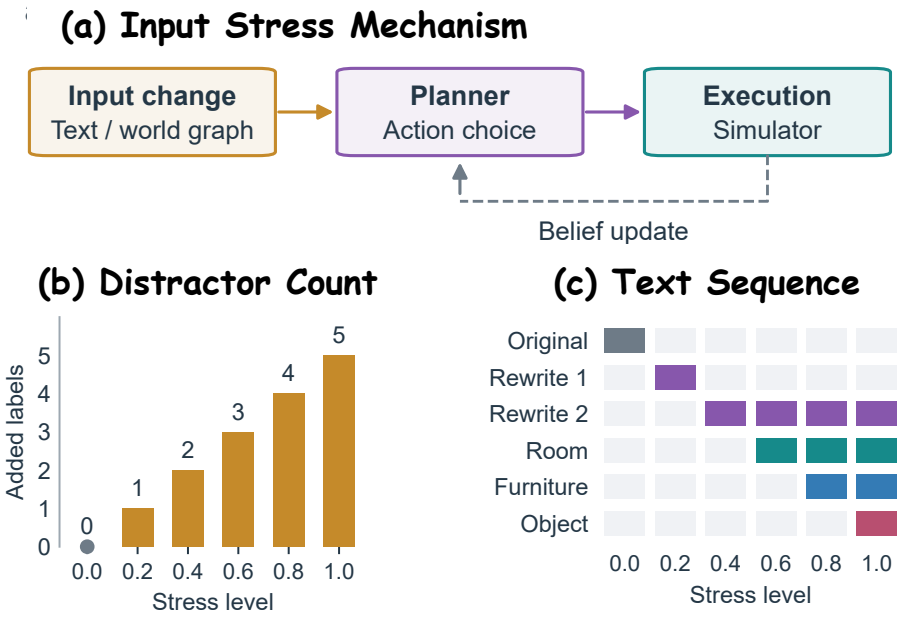


Figure 12: **Controlled input changes and stress schedules.** (a) Instruction and belief changes affect planning and execution while the task remains fixed; (b) Distractor-count levels; (c) Instruction/context sequence with room, furniture, and object context available.

B.3.2 Matched Execution and Metric Analysis

We organize the analysis at three levels: individual episodes, task families, and successive EAS versions. Comparisons share the task evaluator, execution budget, encoder, value critic, Stage Baseline, and evaluation thresholds.

Episode: Matched Execution and Evidence. Each episode run starts from a reset scene, and clean–stress pairs share the episode, repetition, and execution context initialization. Run records also identify the EAS version, intervention, perturbation seed, and selected text sequence where applicable. Execution traces provide task completion, proposition satisfaction, recovery cost, stage-conditioned loss, and constraint violations, each computed according to its definition in Appendix A. Recovery costs accumulate over detected recovery windows, with windows still open at termination recorded as censored. These episode records provide the evidence for comparison and aggregation within each task family.

Task Family: Aggregation and Stress Response. For each EAS version and stress protocol, we group eligible episodes by task family. Recovery costs are summarized by their mean and upper quantile, while absolute loss differences between matched clean–stress runs are normalized by the realized perturbation dose; their maximum gives the family sensitivity $\widehat{\beta}_f$. At each stress level, lower quantiles of completion and satisfaction and upper quantiles of recovery, Stability, and constraint loads determine the operational margin $M_f(\lambda)$ under the common thresholds. The resulting curve gives the family's stress response, with sampled capacity λ_f^* defined as the highest tested level with nonnegative margin.

Within each EAS version and repetition, text-stress aggregation uses complete six-level sweeps for each episode and perturbation seed, keeping the text sequence and variant-collection version fixed. Coverage records identify contributing episodes and exclusion reasons, with distinct counts for tasks, completed executions, and metric-valid records.

Evolve: Comparison Across EAS Versions. For each recorded EAS version $\pi^{(k)}$, we repeat the family-level analysis on matched tasks and stress levels. The context-evolution protocol uses earlier executions to produce episode-specific and batch-level guidance for subsequent attempts (Appendix C.4). Comparisons with the non-evolve context control track changes in recovery cost, disruption sensitivity, and the full margin profile, showing which task families improve or degrade. Newly acceptable stress levels and increased capacity indicate extension of the tested operating conditions, while margin changes where both versions remain acceptable measure gains or losses in operating headroom.

B.4 Validity & Limitation

Threats to Metrics Validity Rebound measures recovery effort, Stability measures stage-conditioned variation and clean–stress differences, and GE measures acceptability along a prescribed stress path (Appendix A).

LLM judgments introduce sensitivity to prompt wording and model behavior because semantic rewards can affect critic-derived Stability signals and the stability component of GE. Fixed resilience evaluation rubrics, low-temperature LLM inference, and structured JSON outputs standardize these judgments, while saved inputs and reports allow individual assessments to be inspected. Online reward shaping uses step-level evidence, and offline diagnosis additionally uses terminal outcomes; Appendix E.2 specifies their inputs and settings.

For EAS generalization claims, we require held-out tasks that are excluded from adaptation and model selection. We solve this by encoding stage baseline, which only funcs in our EAS resilience evaluation layer.

Limitations of Scope and Evaluation Cost. Our current datasets design and evaluation focus on simulation (Habitat-Sim 3.0), and experiments on physical robots remains future work and verification. Trace collection, critic inference, and LLM calls add runtime and compute costs. The implementation controls semantic-call frequency, caches repeated reward queries, and limits judge context (Appendix E.1).

C Resilience Evaluation Implementation

The implementation connects task-grounded planning, embodied skill execution, and the trajectory measurements defined in Appendix A. This appendix describes the shared runtime, the method adapters, and the context updates used in the experiments.

C.1 Asynchronous EAS and Resilience Evaluation Execution

Embodied Execution and Measurements Record Workflow. In our EAS resilience evaluation, each episode begins with a scene reset and task instruction. The planner reads the current world graph, assigns skills to the agents, and uses observations and skill responses from Habitat-Sim to prepare the next decision. The runner records actions, observations, task progress, and recovery events with their episode, simulator-step, and planning-step identifiers. These identifiers associate evaluation inputs and outputs with the corresponding actions. Method adapters enter the loop through planning, action validation, skill feedback, and episode-completion callbacks. Figure 13 connects the embodied execution views to the measurement inputs.

The encoder and value critic process recorded states with fixed evaluation parameters, using Stage Baseline $\mathcal{N}$ for normalization. Recovery windows support Rebound, paired stage-conditioned losses support Stability, and matched stress sweeps support graceful extensibility. At episode termination, the critic completes trajectory analysis and the runner assembles the metrics. When enabled, Attribution-Driven Credit Assignment (ADCA) uses the completed trajectory and outcome to assess individual actions. These diagnostic reports supply failure explanations and guidance for later attempts. The evaluation prompts detail in Appendix E.2.

Parallel Scheduling and Asynchronous LLM Critic Episodes run concurrently in isolated worker processes, each with its own (Habitat-Sim) simulator, planner state, and records. The scheduler collects completed episodes and dispatches the remaining work as workers become available.

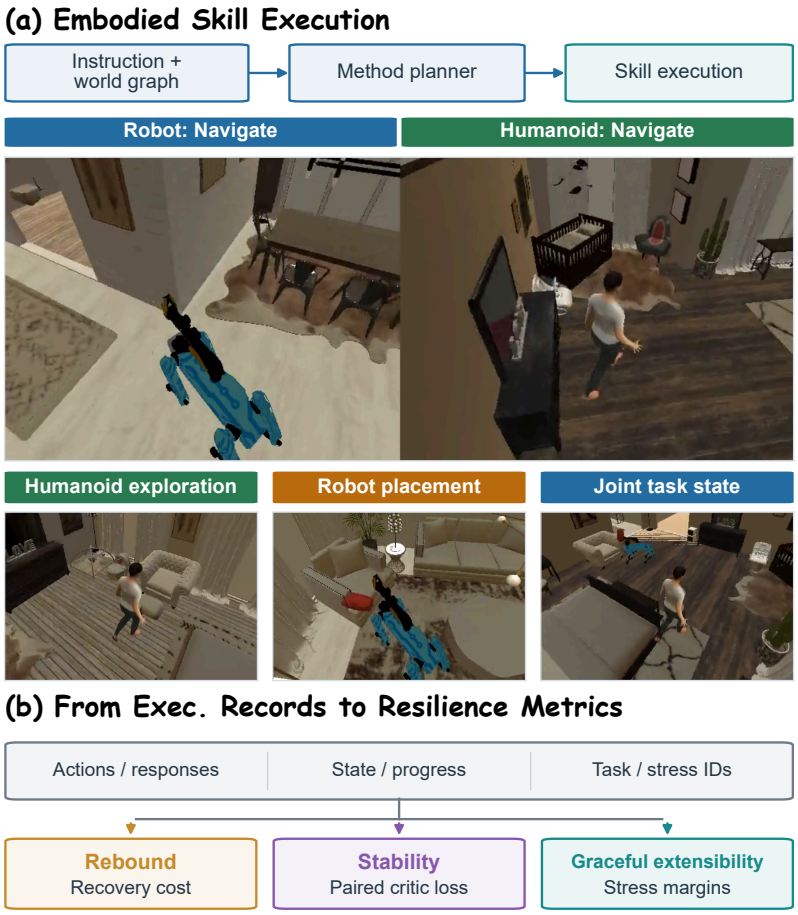


Figure 13: **Embodied execution and resilience measurement.** (a) Representative Habitat-Sim views show robot and humanoid skills in embodied environment. (b) Execution records supply recovery, Stability, and stress-response measurements.

Results are joined at repetition boundaries before aggregate analysis and context updates. For efficiency, LLM Critic Judge inference runs through API service, while Habitat-Sim advances inside the episode worker. After each environment step, local evaluation monitors update the execution and progress records, and semantic reward shaping requests a judgment on planning changes, recovery events, or configured post-replanning steps, and reuses cached responses. The returned assessment enters the critic's reward record before the next planner decision.

Embodied Execution Skills The planner selects an agent and a parameterized skill, using world-graph entity names to specify object, furniture, or room targets. Table 8 summarizes the skills under Habitat-Sim 3.0 executor, with availability determined by each agent's capability configuration. Perception tools supply additional planning context through queries about objects, receptacles, rooms, and other agents' actions.

C.2 Methods Comparison

EAS Evaluation Methods Comparison We analyse existing benchmarks and show the comparison in Table 9. Existing benchmarks increasingly move beyond final success, but they still pro-

Table 8: **Embodied execution skills.**

Skill family	Calls	Operation
Navigation	`Navigate`	Moves the agent to a named entity.
Exploration	`Explore`	Searches a room by visiting its furniture and receptacles.
Manipulation	`Pick`, `Place`	Grasps a target object or places the held object on or within a destination, optionally next to a reference object.
Rearrangement	`Rearrange`	Combines navigation, picking, transport, and placement.
Container access	`Open`, `Close`	Opens or closes an articulated container.
State changes	`Clean`, `Fill`, `Pour`, `PowerOn`, `PowerOff`	Cleans, fills, or switches an object on or off when interaction conditions are met. Pouring transfers liquid from a held container to a target.
Waiting	`Wait`	Keeps the agent idle for a fixed interval.

Table 9: **Positioning existing EAS evaluation against resilience metrics.**

Evaluation focus	Representative work	Primary signals	Resilience perspective
Outcome and capability	PARTNR, EmbodiedBench	Success, completion, and planning / action capability	Can not distinguish resilience exec. from costly success.
Safety and hazards	SafeAgentBench, HazardArena	Rejections, violations, and hazard progression	Mainly focus on responsive post disruption recovery and stability.
Perturbation robustness	IS-Bench; adversarial stress tests	Performance changes under input, scene, or policy perturbations	Degradation endpoint, no recovery trajectory analysis.
Resilience	**Our framework**	Recovery cost, β-stability, stress-response, and extensibility capacity	Joint assessment of rebound, stability, and graceful extensibility.

vide limited process-level measurements for recovery burden, decision stability, and stress-response degradation.

EAS Methods Overview We report comparative EAS baselien methods in Table 10, and further detail the reproduction on Habitat-Sim simulation forum (integrating with our resilience evaluation). Under asynchronous multi-robots collaboration design we have mentioned before, different EAS mechanism focus on motion planning and failure recovery, which keeps the semantic task, simulator, skill set, and execution budget comparable across methods.

C.3 EAS Methods Reproduction & Implementation

Each method uses the shared PARTNR skill executor and contributes its own planning or adaptation (including recovery and evolution) mechanism. Table 10 indexes the adapters and the state they update.

M1. ReAct [40]. At each planning turn, a centralized ReAct-style planner reads the task instruction, current world graph, and execution history. It generates reasoning together with agent-specific calls in the shared skill grammar. Habitat-Sim executes the selected skills and returns observations and completion messages for the next planning turn.

M2. SayCan [2]. SayCan generates candidate skill calls from the task and observed world. Conditional token likelihood supplies the language score, while graph-based checks estimate whether the target, object state, and agent state support execution. Their product ranks the candidates. Joint selection reserves manipulated objects across agents, and the selected calls pass to the shared skill executor.

Table 10: EAS resilience-relevant comparative methods and their target aspects based on resilience evaluation.

ID	Method	EAS Mechanism	Resilience Adaptation	Target Aspects
M1	ReAct	Reason–act–observe	Joint skill grammar	Action plan
M2	SayCan	Likelihood $\times$ feasibility	Graph affordance; object reservation	Joint action
M3	Inner Monologue	Execution feedback	Scene and agent-state messages	Prompt history
M4	CoPAL	Precondition repair	Navigation and grasp prerequisites	Pending skill
M5	Reflexion	Verbal reflection	Task-keyed trial memory	Reflection memory
M6	CycleVLA	Verify and backtrack	Semantic verification; history checkpoints	Planning context
M7	SMART-LLM	Team allocation	Tool capabilities; task dependencies	Task allocation
M8	AgentEvolver	Practice and reuse	Grounded practice; step attribution	Experience memory
M9	CLARE	Route and expand	Reconstruction routing; textual guidance	Router and skill library

M3. Inner Monologue [14]. Inner Monologue consumes each completed skill response once and combines it with the current scene and agent state. A feedback generator turns this information into a structured message retained in the planner's history. The next reasoning turn uses the message to assess the preceding action and choose subsequent skills.

M4. CoPAL [15]. CoPAL checks proposed manipulation skills against the observed world before execution. For an out-of-reach target or an object that must first be grasped, it inserts a navigation or pick action while retaining the intended manipulation as pending work. Completed failures add corrective context for the next decision. Bounded local repair and execution feedback then guide retry of the original action.

M5. Reflexion [32]. Reflexion stores verbal reflections keyed to the scene, episode, and task instruction. After a failed attempt, it summarizes the recorded trajectory into a corrected plan using the available skills. At the start of a later attempt on the same task, recent reflections enter the planning context. The trial runner retains attempt outcomes and skips tasks that have already succeeded.

M6. CycleVLA [23]. CycleVLA verifies subtask progress from semantic observations and can return to an earlier planning-history checkpoint. The restored context receives current observations and recovery guidance, preserving the consequences of executed actions. The planner samples discrete joint skill calls, identifies a dense neighborhood under structured action distance, and selects its medoid, the action most representative of that neighborhood. Subsequent observations support the next verification step.

M7. SMART-LLM [16]. SMART-LLM decomposes the instruction, forms teams from available tool capabilities, and allocates subtasks. Tasks with satisfied dependencies receive disjoint available teams, while running assignments are retained. Execution feedback and observed postconditions update task status; failures trigger bounded reallocation. The adapter translates the allocation into agent-specific skill calls and assigns waiting actions to agents without an active task.

M8. AgentEvolver [44]. AgentEvolver attributes completed trajectory steps and distills their outcomes into reusable experience. Successful pick or place evidence can supply a grounded practice goal, executed in a separate rollout before the environment is reset for the assigned task. At epoch boundaries, pending experience is published to memory. Later attempts can retrieve experiences with positive lexical overlap and add their guidance to the planning context.

M9. CLARE [30]. CLARE encodes the instruction and initial observed world using a fixed text encoder. Linear reconstruction models score this input, and the lowest-error model selects associated textual skill guidance for execution. Completed episodes contribute action and outcome evidence to task groups. At epoch boundaries, sufficiently observed groups enter an expansion check; groups requiring a new routing model are linked to new or reused textual guidance.

C.4 Context Evolution Protocol

Planner context. The shared context-evolution baseline combines batch-level guidance $G^{(k)}$ with guidance $E_i^{(k)}$ from earlier attempts at episode i. At version k, the resulting planner is

$$\pi_{\mathrm{ctx},i}^{(k)} = \pi^{(0)} \oplus G^{(k)} \oplus E_i^{(k)}, \tag{48}$$

where $\pi^{(0)}$ is the base planner and $\oplus$ denotes addition to its prompt context. The update changes the information supplied to the planner while retaining the base model parameters and measurement settings.

Update sequence. After each episode, the manager stores a summary of the outcome, progress, replanning activity, and diagnostic evidence. At the round boundary, it combines recent summaries into batch advice and publishes the context for subsequent attempts. History and character limits bound the retained material. The shared runner builds batch advice through rules over the summaries; an attached LLM client enables compression and refinement.

C.5 Diagnosis-Guided Execution Changes

The runtime provides recovery, phase-planning, and stall-control modules that turn execution feedback into a planning change. Table 11 maps their triggers to planner operations and evaluation records.

Table 11: **Execution modules and associated records.**

Module	Trigger	Planner operation	Recorded evidence
Recovery	Skill failure or missing state evidence	Refresh perception; summarize, reflect, or revisit a context checkpoint	Recovery windows; effort and state debt
Phase planning	Repeated actions or weak phase progress	Supply subtask status; advance the phase; add repetition feedback	Action history; stage-conditioned loss
Stall control	Persistent waiting or action cycles	Replan with corrective feedback; apply repetition and rollout limits	Stop reason; completion and execution costs

Feedback-guided recovery. The recovery manager uses skill feedback, world-graph evidence, and progress changes to guide subsequent planning turns. The planner can invoke context tools to refresh perception, summarize or reflect on recent history, and revisit an earlier planning context with the current observed world. Executed actions remain in the trajectory, and recovery cost accumulates effort and state debt within the detected recovery windows.

Consistency-guided phase planning. The phase helper supplies the current subtask and its tracked status. It uses execution-response cues or a configured phase limit to advance the phase, and adds feedback when recent action representations repeat. The resulting trajectories enter the stage-conditioned loss and paired disruption-sensitivity calculation in Appendix A.2.

Bounded execution under increasing stress. The planner tracks consecutive all-agent waits, repeated joint actions, and recurring wait, explore, or scan actions. Configured limits govern corrective feedback and termination, while the runner enforces the rollout budget. Stop reasons accompany completion, recovery, and loss records, which feed the family-level margin and capacity calculation in Appendix A.3.

Table 12: Rebound validity under ***matched successful outcome*** pairs.

Statistic	**Clean**	**Perturbed**
Mean C_{rec}	11.65	36.27
Median C_{rec}	9.81	29.90
Mann–Whitney U	414.0	
Two-sided p	7.40×10^{-8}	
One-sided p (Clean $<$ Perturbed)	3.70×10^{-8}	
AUC	0.15	
Rank-biserial correlation	-0.70	

D Detailed Experiments Results

D.1 Computation Resources

All experiments are implemented on our Resilience Evaluation Layer testbed, which couples embodied execution with rebound, and LLM-as-a-Judge mechanism. Experiments conducted on NVIDIA A800 GPUs, take the VLLM reasoning method [18], each method was evaluated for 3 epochs, with each epoch approximately consuming 44 GPU hours. We use the API-call LLMs for EAS resilience evaluation, we take the stronger 'GPT-5.1' model for analysis, which is able to find more detailments in EAS execution.

The simulator and planner-response budget determines how many executable trajectories are collected, while the LLM Judge budget determines how many diagnostic labels and reward-shaping signals are produced from those trajectories. Appendix D documents the experiment result surfaces and rollout-derived metrics, and Appendix E documents all LLM roles used to generate model decisions, judge scores, dataset instructions, and self-evolve procedures.

D.2 Resilience Aspect Analysis in Validity Experiments

We investigate our **Resilience Metrics** correlation, and reveal the variation in system resilience as resilience aspects: rebound, stability and graceful extensibility.

Rebound Validity under Comparable Completion. The successful-execution subset contains 24 clean and 115 perturbed observations, each with a valid recovery baseline and 100% task completion. Mean recovery cost C_{rec} is 11.65 for clean executions and 36.27 for perturbed executions, a ratio of 3.11 (Table 12). Equal completion therefore coexists with substantially different recovery burdens in this subset.

Figure 14 shows both the full cost distributions and their contributing runs. The median difference, 9.81 versus 29.90, places the contrast within the central part of the distributions as well as the high-cost tail.

Stability Validity under Semantic-Preserving Perturbations. Stability is reported through the neighborhood instability $\hat{\beta}$ and its decomposition into value variation and output-action variation. Surface rewrites and filler injections preserve the intended task while changing its linguistic presentation, a large increase in $\hat{\beta}$ indicates decision sensitivity and planning oscillation. Table 13 reports a neighborhood diagnostic that combines critic and output variation.

Clean instructions and surface rewrites both yield 79.2% mean completion, while recorded neighborhood instability rises from 0.084 to 0.264. Critic and output variation also increase. Filler injections have the highest completion and the largest values in all three diagnostic columns. A high final completion level can therefore coexist with substantial variation in the execution diagnostics.

Graceful Extensibility Validity under Stress Sweep. Stress-response profiles describe how task completion, recovery cost, and operational margin vary with stress severity λ. Following Appendix A.3, we assess graceful extensibility by comparing these profiles with adaptive planning

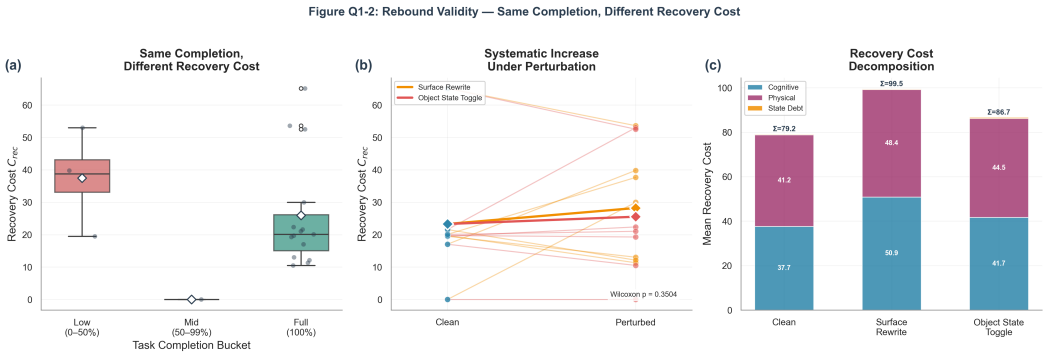


Figure 14: **Rebound validity under comparable completion.** (a) Episodes with similar task completion exhibit different recovery costs. (b) Paired clean/perturbed executions show systematic increase in C_{rec}. (c) Decomposition of C_{rec} identifies cognitive recovery, physical recovery, and state debt as distinct recovery burdens.

Table 13: **Stability validity under semantic-preserving perturbations.** The table reports the mean instability and its decomposition across perturbation groups. Higher $\hat{\beta}$ indicates less stable execution.

Condition	$\hat{\beta}$ **mean**	$\hat{\beta}$ **median**	$\hat{\beta}_{\text{vv}}$ **mean**	$\hat{\beta}_{\text{out}}$ **mean**
Clean	0.084	0.084	0.000	0.169
Surface Rewrite	0.264	0.225	0.192	0.371
Filler Injection	0.315	0.259	0.247	0.418

and coordination enabled and disabled under matched evaluation conditions. Newly acceptable conditions and increased stress capacity indicate boundary extension, while margin gains within the shared acceptable range quantify additional operating headroom.

The stress sweep reveals graceful extensibility changes in execution quality that task completion alone does not capture. At medium-to-high severity, recovery cost rises and operational margins $M_f(\lambda)$ become more negative, while boundary events indicate violations of the evaluation limits. These observations characterize degradation under stress and provide a basis for assessing graceful extensibility through matched comparisons with EAS adaptation process, examining acceptable conditions, changes in stress capacity, and margin gains within the acceptable range.

D.3 Statistical Reliability and Sensitivity

D.3.1 Resilience Metrics Statistical Reliability

To ensure metric statistical reliability, we measure the variance and stability of the proposed resilience metrics across multiple trials and varied physical/semantic noise conditions.

The low coefficient of variation across runs confirms that the proposed metrics do not fluctuate arbitrarily under environmental or agent noise, satisfying repeatability and reliability requirements expected in rigorous measurement instruments.

D.3.2 Sensitivity & Scalability Analysis

We examine backbone sensitivity under the same execution budgets, using Qwen3-8B-Instruct and Qwen3.5-9B for resilience Rebound and Stability analyses. Scalability comparisons include Qwen2.5-7B-Instruct and Llama-3.1-8B-Instruct to characterize task completion, replanning counts, and runtime.

Though stronger LLM backbones usually improve completion, reduce unnecessary replans, or shorten runtime, our resilience evaluation asks whether the same metric relations persist: outcome metrics can still hide rebound cost, unstable executions still appear as wider stability-load distribu-

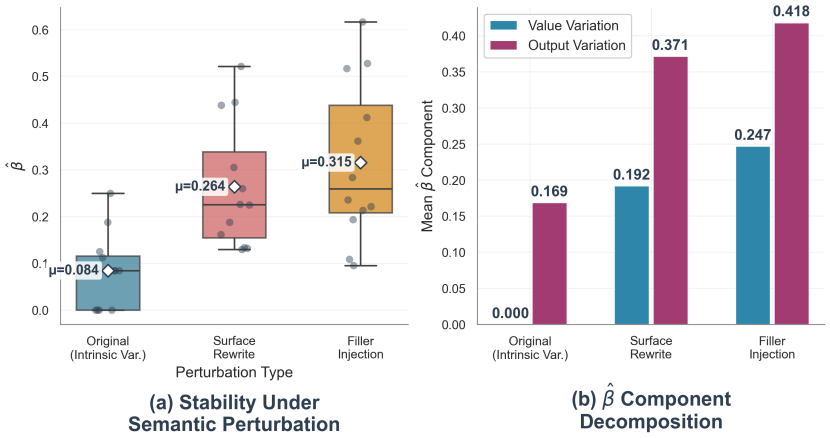


Figure 15: **Execution diagnostics under semantic perturbations.** (a) Neighborhood instability, shown as mean ± SD. (b) Mean critic and output variation.

Table 14: **Graceful Extensibility stress-response summary.** We aggregate the Validation Experiments stress sweep by stress severity. $M_f(\lambda)$ denotes the operational margin, boundary and hard-boundary rates report whether executions approach or hit the contract boundary.

λ	n_{pass}	**Completion**	C_{rec}	$M_f(\lambda)$	**Boundary Hit**	**Hard Boundary**
0.0	54	0.819	41.73	-0.633	0.222	0.778
0.2	39	0.903	38.68	-0.387	0.077	0.462
0.4	39	0.785	39.03	-0.383	0.077	0.462
0.6	33	0.795	50.61	-0.537	0.364	0.818
0.8	27	0.785	50.41	-0.600	0.111	0.667
1.0	27	0.843	41.61	-0.568	0.111	0.667

tions. Our experiments indicate that while larger models improve baseline success, the *Resilience Signatures* persist across scales.

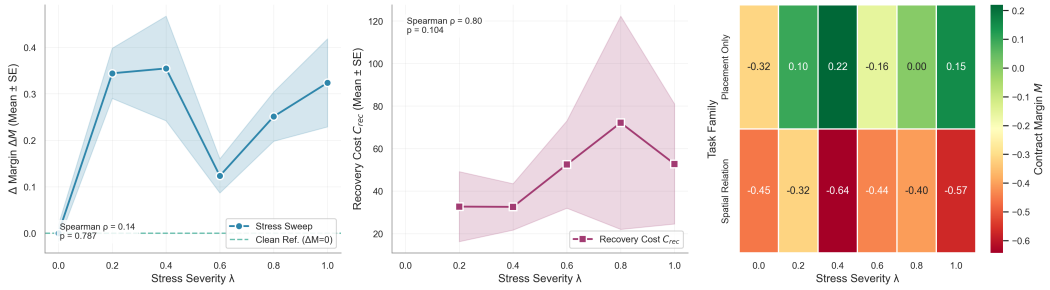
Figure 18 compares recovery burden and execution consistency across Qwen3 and Qwen3.5. The boxplot summarizes the spread of C_{rec}, while the scatter plot shows that executions can incur different recovery costs across different LLM backbones. The stability-load distributions complement this comparison by showing variation in stage-conditioned instability. These measurements distinguish execution processes that appear similar resilience signatures although different LLM backbones.

Figure 19 provides supporting evidence from Qwen3, Qwen2.5, and Llama3.1. Variation in replanning demand and runtime across these backbones highlights the execution costs that accompany task completion, motivating process-level diagnostics across model families. Combined with the primary metric comparisons, these experiments support our unified resilience evaluation framework to assess how different LLM-based systems achieve their outcomes.

In a word, the cross-backbone study supports the interpretation that resilience is a **system-level execution intrinsic property** of the EAS, rather than only a raw capability of one LLM component (which usually indicates one perspective of EAS).

D.4 LLM Judge Consistency and Simulator Ground-Truth Alignments

We assess the judge signals used in resilience evaluation from two perspectives: consistency across models and alignment with simulator ground truth. Appendix E describes the prompts and scoring procedures for reward shaping and trajectory diagnosis.



(a) Degradation from Clean Baseline

(b) Recovery Burden as Supporting Evidence

(c) Family-Specific Brittleness Heatmap

Figure 16: **Graceful Extensibility validity under stress sweep.** (a) GE margin can expose a localized degradation process around medium stress. (b) Boundary-hit and hard-boundary-hit events increase when the margin approaches zero. (c) AUDC_f summarizes whether degradation is smooth or cliff-like across task families.

Table 15: Statistical Reliability of Resilience Metrics (CV: Coefficient of Variation).

Metrics	**Condition**	**Mean ± Std**	**CV**
Recovery Cost C_{rec}	Clean Baseline	23.29 ± 1.989	0.085
Recovery Cost C_{rec}	Physical Perturb.	26.87 ± 1.776	0.066
Instability β	Clean Baseline	0.084 ± 0.0079	0.094
Instability β	Semantic Perturb.	0.289 ± 0.0153	0.053
Stress Curve Margin M_f	Clean Baseline	-0.351 ± 0.0063	0.018
Stress Curve Margin M_f	Stress Sweep ($\lambda > 0$)	-0.071 ± 0.0268	$0.376^{\dagger}$

† High CV in Stress Sweep is expected due to intentional continuous λ variation.

D.4.1 Consistency across LLM Judge Models.

We compare GPT-4o and GPT-5.1 LLM Critic on 210 identical prompt inputs (from validity experiments trajectories), fixing the task goal, execution context, and environment feedback. To examine consistency across EAS methods, we average rewards over each method's matched inputs and rank the resulting means, as shown in Table 16.

Table 16: **EAS reward rankings across judges.** Arrows show rank changes from GPT-4o to GPT-5.1; rank 1 indicates the highest mean r_{LLM}.

EAS method	**Mean r_{LLM}**		**Rank change**
	GPT-4o	**GPT-5.1**	
SMART-LLM	-0.146	0.245	$1 \rightarrow 2$
Reflexion	-0.304	0.265	$2 \rightarrow 1$
CoPAL	-0.323	0.237	$3 \rightarrow 3$
CLARE	-0.456	0.057	$4 \rightarrow 4$
InnerMono	-0.514	0.001	$5 \rightarrow 5$

The method-level correlation is $\rho = 0.90$ (95% CI $[0.40, 1.00]$): SMART-LLM and Reflexion exchange the first two positions, while the other three methods retain their ranks across LLM Judge backbones. The maximum displacement is one position, and both judges identify the same leading three methods. These results show greater agreement in aggregate reward rankings.

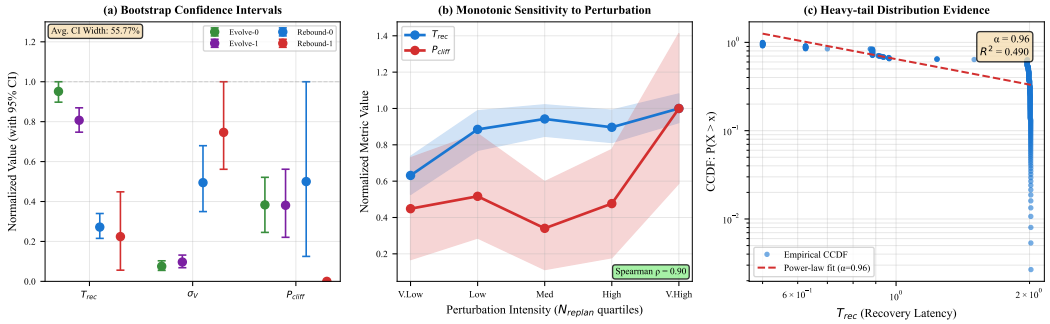
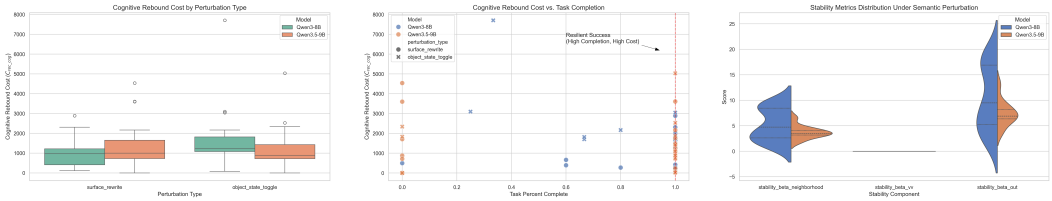


Figure 17: Reliability and sensitivity of resilience metrics. (a) Bootstrap 95% confidence intervals for c_{rec}, β_{val}, and safety signals across runs/batches. (b) Monotonic sensitivity under increasing perturbation intensity (proxied by β quartiles), with Spearman $\rho = 0.90$. (c) Heavy-tail evidence for recovery latency via CCDF with a power-law fit ($\alpha = 0.96$, $R^2 = 0.49$).



(a) Rebound cost (b) Completion vs. recovery (c) Stability load

Figure 18: **Resilience metrics Sensitivity on LLM backbones.** Across Qwen3 and Qwen3.5, C_{rec} distinguishes recovery cost among successful or high-completion runs, while stability distributions capture execution oscillation.

D.4.2 LLM Reward Alignment with Simulator Ground Truth.

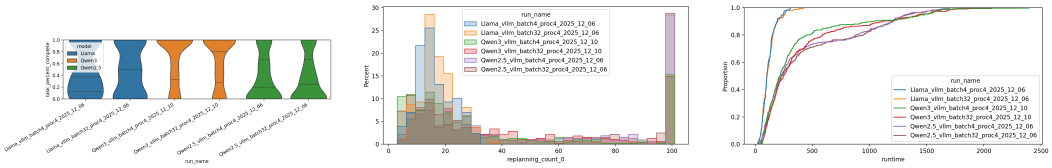
We analyze 2,230 matched pairs of Reward Shaper reports and episode outcomes from the benchmark dataset for LLM reward analysis. The coarse "Percent Complete" was calculated using Habitat-Sim and the task propositions from the public PARTNR benchmark (GT, while it is coarse reward), and the reward was derived from the LLM's episode-level report. Table 17 shows that the critic score increases along with ground-truth task completion, with an episode-level Spearman correlation of $\rho = 0.915$.

Table 17: **Judge scores with simulator completion.** Episode-level critic scores grouped by ground-truth completion $p_{\mathcal{T}}$.

Completion range	Episodes	Mean $p_{\mathcal{T}}$	Mean critic score
$p_{\mathcal{T}} = 0$	473	0.000	0.000
$0 < p_{\mathcal{T}} < 0.5$	316	0.286	0.107
$0.5 \leq p_{\mathcal{T}} < 1$	381	0.652	0.203
$p_{\mathcal{T}} = 1$	1,060	1.000	0.526

D.5 Embodied Execution Case Analysis

We select an illustrative case pair from the same domain where outcome alone is uninformative but resilience traces diverge sharply. Figures 20(a-b) contrast *Active Wait* vs. *Panic Loop* as rebound mechanisms. Figures 20(c-d) contrast *Constructive Evolution* vs. *Unanchored Drift* as evolve mechanisms.



(a) Completion (b) Replanning (c) Runtime

Figure 19: **EAS execution scalability analysis across LLM backbones.** (a) Task completion p_T, (b) replanning counts, and (c) runtime CDFs for Qwen3, Qwen2.5, and Llama3.1 provide outcome and cost scalability analysis for resilience evaluation.

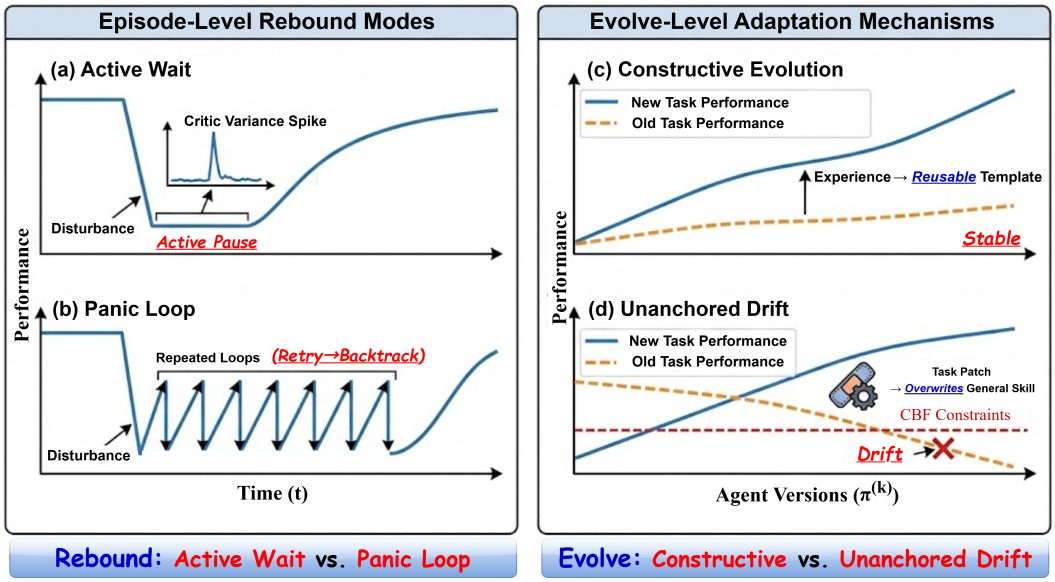


Figure 20: The Panic Loop and Active Wait, Constructive evolution and Overfitting drift.

D.5.1 Sampling Protocol and Labeling Schema

To avoid partial conclusions, we sample traces by stratifying over (i) distribution types, (ii) task families, and (iii) recovery-cost regimes. For each sampled episode, we assign a primary trigger source based on the earliest non-recoverable deviation in the trace: *Perception*, *Planning*, *Execution Tools*, *Memory/State*, and *Action Control*. When multiple sources co-occur, we record the first trigger and mark secondary cascades. This protocol keeps the analysis reproducible and aligns diagnosis with auditable traces.

Simulation trajectory. We report a representative Habitat-Sim execution in Fig. 21. The case is a two-object household transfer task sampled from the clean-reference setting. The rendered panels show the simulator-side motion realization, while the saved trace records the planner-level evidence as grounding, transport, recovery, and completion phases. Habitat-Sim expands these high-level decisions into navigation, grasping, transport, and placement primitives, where the visual panels expose the concrete embodied trajectory.

The auditable trajectory is summarized at the semantic phase level, and stored across the text trace, planner log, scalar statistics, and compact critic export. Table 18 shows how the simulator trajectory, planner rationale, and labeling schema are connected without relying on endpoint success alone.

Trace-level Finding: Auditable Simulation Evidence

This example illustrates why our sampling protocol stores trajectory evidence beyond endpoint success. The episode succeeds, but the trace still exposes primary *Tool* trigger, *Action*

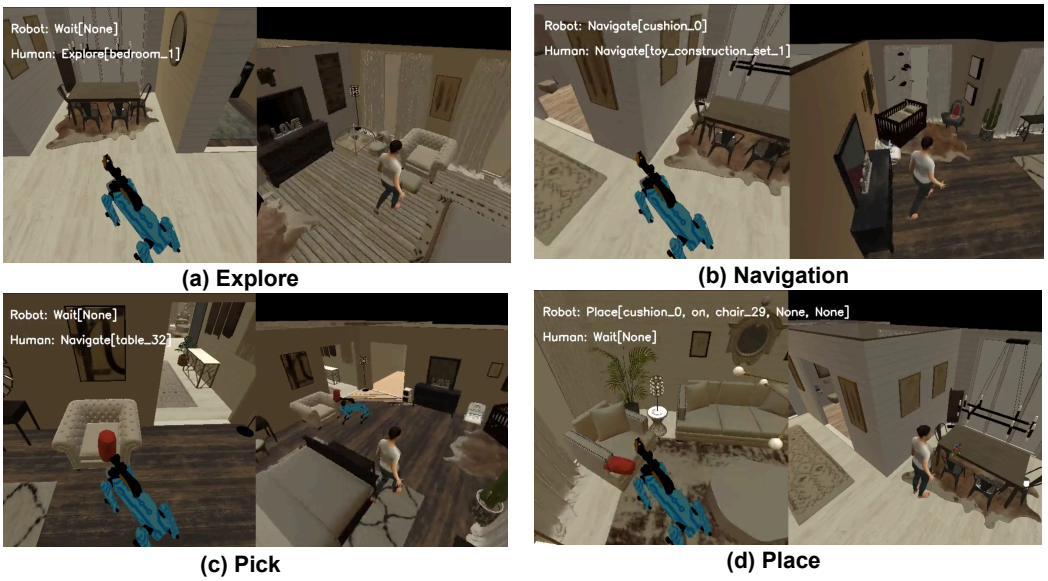


Figure 21: A complete two-object household transfer trajectory in Habitat-Sim. The trace first resolves object locations by exploration, then executes transport and placement actions for the cushion and toy construction set.

Table 18: **Execution and recovery sequence for Fig. 21.**

Stage	Planner action	Execution feedback	Trace label
Grounding	Plan two transfers and explore rooms.	Required objects and source receptacles identified.	Perception grounding.
Initial attempt	Submit an incomplete `Rearrange` call.	Call rejected for missing arguments.	Tool/API trigger.
Action repair	Supply the missing relation and constraint fields.	One transfer completes; the other remains unresolved.	Recovery window.
Control recovery	Wait, then retry manipulation after corrective feedback.	Waiting rejected; the remaining placement completes.	Action-control cascade.
Completion	Terminate after both target relations are satisfied.	Task succeeds after intermediate recovery.	Recovered success.

Control cascade, recovery window, and rebound cost. Thus, the Habitat-Sim case provides a complete audit chain from embodied world graph perception to motion, planner rationale, simulator response, label assignment, and resilience metrics.

D.5.2 Rebound Case Comparison Study

This case study focuses on two claims. First, success rate cannot identify whether a success is achieved by controlled recovery or by unstable brute force. Second, rebound and stability signals are complementary: high B_{epi} exposes the cost of recovery, while value variance and replanning count expose whether the agent changed hypotheses or merely repeated a failed plan.

***Active Wait* (efficient rebound, Case 139).** Under partial observability, the agent encounters grasp ambiguity and short tool-latency disruptions. The trace exhibits a brief *diagnostic pause* (stalled progress token with a transient β_{val} spike), followed by continued execution *without* retry/backtrack escalation. The resulting resilience signature is: **low C_{rec} + bounded replanning + localized variance spike**, indicating that uncertainty is handled by stabilization rather than by costly recovery actions.

***Panic Loop* (costly recovery by brute force, Case 141).** In a comparable episode (same success outcome), repeated placement failures trigger long retry/backtrack loops. The trace shows many repeated action templates with minor perturbations but *no* clear diagnostic transition. That is, the system does not change hypotheses, only re-executes. The signature becomes: **high C_{rec} + repeated replans + prolonged time-to-recover**, revealing a brute-force convergence behavior that success-rate metrics cannot penalize.

D.5.3 Evolve-Level Resilience: Adaptive Gains and Cross-Task Regression

Evolve-level evaluation on whether successive updates improve execution while retaining previously achieved capabilities, could be revealed by EAS resilience evaluation. Under the context-evolution protocol in Appendix C.4, each version $\pi^{(k)}$ incorporates guidance from earlier executions. We compare these versions with the fixed-context control on matched tasks and stress levels, keeping execution budgets and measurement settings fixed.

Constructive Evolution. Constructive updates produce reusable guidance that improves subsequent execution across tasks. We assess these benefits through lower recovery cost C_{rec} and reduced disruption sensitivity $\widehat{\beta}_f$, alongside retained task completion. For graceful extensibility, the comparison examines whether updated planning and coordination increase operational margins within the shared acceptable range and extend execution to newly acceptable stress conditions. These changes distinguish additional operating headroom from expansion of the stress capacity λ_f^*.

Context Drift and Cross-Task Regression. Updated guidance can favor recent tasks while degrading execution on previously evaluated task families. We identify this regression through increased recovery cost or disruption sensitivity, declining operational margins, and loss of previously acceptable stress conditions. Examining the corresponding guidance changes, planning decisions, and skill feedback helps trace these metric changes to specific execution failures, making the consequences of each update visible across task families.

D.5.4 Graceful Extensibility Case Study: Margin Proxy Response to Stress.

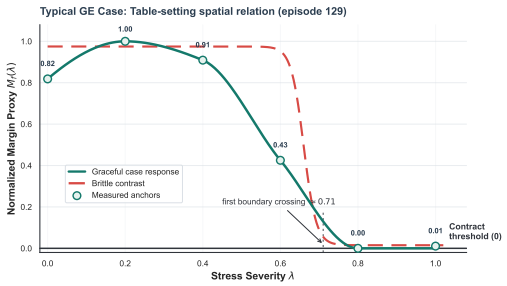


Figure 22: (Episode 129) Representative table-setting spatial-relation case for Graceful Extensibility response.

Case 129 involves arranging plates, glasses, and forks with specified spatial relations. Its stress-response mapping profile reveals increasing recovery demands before task completion declines (Fig. 22). At $\lambda = 0.6$, completion remains at 100%, while recovery cost reaches 130.22 compared with 78.62 under clean conditions($\lambda = 0$). Completion then falls to 50% and 60% at $\lambda = 0.8$ and 1.0. In this case, the adaptive gain could appear as reduced recovery burden while preserving completion, or as acceptable execution under previously unsuccessful conditions.

Referring to Appendix A.3, we assess such gains through matched comparisons with adaptation enabled and non-adaptation. Higher operational margins $M_f(\lambda)$ within the shared acceptable range indicate additional operating headroom, while newly acceptable stress conditions and increased stress capacity indicate EAS boundary extension on resilience.

D.5.5 Embodied Task Execution Outputs List

Each experiment run produces a structured evidence folder, the important artifacts are:

- **Episode summaries:** `episode_result_log.csv`, `run_result_log.csv`, and `end_result_log.csv`.
- **LLM inputs and outputs:** prompt files under `prompts/`, action traces under `traces/`, and planner-side JSON logs under `planner-log/`.

- **Simulator records:** per-episode statistics under `stats/` and optional detailed traces under `detailed_traces/`.
- **Metric analyses:** rebound, stability, degradation, LLM Critic, ADCA, and resilience JSON files under `analyses/`.
- **Critic audit records:** state encoder JSONL logs under `analyses/state_encoder/` and LLM reward shaper JSONL logs under `analyses/reward_shaper/`.
- **Evolution records:** batch-level advice, episode summaries, and evolution context files under `analyses/evolve/`.

E LLM Configuration and Prompt

E.1 LLM Configurations Overview

Generally speaking, we use LLMs in five roles: executable planning, dataset synthesis, step-level reward shaping, offline trajectory diagnosis, and context compression/evolution. These roles are intentionally separated. The planning LLM produces executable high-level tool calls; the judge LLMs produce structured diagnostic signals; and the context LLM compresses previous evidence into bounded prompt overlays. Table 19 summarizes the released usage pattern.

Table 19: **LLM roles and configurations.** T denotes temperature and L the output token limit. Planner temperatures follow the listed model order.

Role	Model	Interface	Output	Generation settings	Execution settings
Planning	Qwen3-8B-Instruct Qwen3.5-9B Qwen3.5-9B-YOYO-Instruct	vLLM [18]	Tool-call text	T: 0.3 / 0.7 / 0.7 L: 2048	World-graph-based grammar and stop tokens; execution traces saved.
Dataset generation	Meta-Llama-3.1-70B-Instruct	Hugging Face	Instructions and evaluation specifications	T: 0.5 L: 4096	Object-state templates grounded in embodied scenes.
Step reward shaping	GPT-5.1	API	Reward scores (JSON)	T: 0.1 L: 200	Calls aligned with planning; cache capacity of 1000 entries.
Offline diagnosis	GPT-5.1	API	Diagnosis report (JSON)	T: 0.2 L: 800	The last 20 formatted steps are retained for long trajectories.
ADCA attribution	GPT-5.1	API	`GOOD`/`BAD` labels	T: 0.0 L: 4096	Numbered trajectory and final outcome score.
Context compression / evolution	GPT-5.1	API	Summaries and advice (JSON)	T: 0.1–0.3 L: 150–2048	Entity names and recent turns preserved; evolution context limited to 2000 characters.

In our main Qwen3 configuration, the planner uses local vLLM inference, while critic reports, reward shaping, ADCA attribution, and context compression use GPT-5.1 through the API-used interface. In the Qwen3.5 configuration, the planner can point to a local vLLM endpoint with `Qwen3.5-9B-YOYO-Instruct`. All LLM-facing evidence is saved through prompt files, trace files, planner logs, reward-shaper JSONL logs, critic reports, ADCA files, or evolve-context artifacts.

E.2 LLM-as-a-Judge Usage

The judge LLMs are used for two structured signals: step-level reward shaping and offline critic reports. Their model and decoding settings are summarized in Table 19.

E.2.1 Critic Report Usage

The offline critic report prompt is:

Critic Report Prompt

```
You are an expert robot task execution analyst.

Task: {task_instruction}

Execution Trajectory:
{formatted_trajectory}

Final Outcome:
- Success: {success}
- Percent Complete: {percent_complete:.2f}
- Total Reward: {total_reward:.2f}
- Number of Simulation Steps: {num_steps}
- Number of Planning Steps: {planning_step_count}

Please analyze this trajectory and provide:
1. Key failure points (if any) - which steps went wrong?
2. Root causes - why did failures occur?
3. Improvement suggestions - what could be done better?
4. Overall assessment - summary of execution quality

Respond ONLY with a JSON object in this exact format:
{{
    "failure_points": [<list of step indices where failures occurred>],
    "root_causes": ["<cause 1>", "<cause 2>", ...],
    "improvement_suggestions": ["<suggestion 1>", "<suggestion 2>", ...],
    "overall_assessment": "<1-2 sentence summary>",
    "execution_quality_score": <float between 0.0 and 1.0>
}}
```

E.2.2 Step-Level LLM Reward Usage

The reward shaper uses a fixed JSON rubric and computes

$$r_{\text{LLM}} = 0.4r_{\text{progress}} + 0.4r_{\text{rationality}} + 0.2r_{\text{efficiency}}. \tag{49}$$

The prompt template is:

LLM Reward Prompt

```
You are an expert evaluator for robotic task execution.
You are assisting in training a robot by providing shaped reward signals.

Your Goal: Evaluate the quality of the current action in the context of the
     entire task history.

Task Goal: {task_instruction}
Current Step: {step_count}
Context:
- Task Percent Complete: {task_percent_complete:.2f}
- Perception Complete: {perception_complete}
Previous Actions (Last 5): {action_history}
Action Response: {action_response}
Previous State: {state_description}
Action Taken: {action_description}
Next State: {next_state_description}
Base Environment Reward: {base_reward}

Consider the following criteria:
```

```
1. **Goal Progress**: Did the action move closer to completing the task
    goal?
2. **Action Rationality**: Was the action reasonable given the current
    state and task?
   * **Exploration**: In early steps, exploration is valuable if the target
       is unknown.
   * **Logical Consistency**: Are actions logically sequenced? (e.g.,
       Navigate -> Pick -> Navigate -> Place).
   * **Preconditions**: Does the action respect preconditions (e.g.,
       holding an object before placing)?
3. **Efficiency**: Was this an efficient step, or could a better action
    have been taken?
4. **Perception & Completion**: Consider if the robot has successfully
    perceived the necessary objects and if the task is nearing completion.

Respond ONLY with a JSON object in this exact format:
{
    "goal_progress_score": <float between -1.0 and 1.0>,
    "rationality_score": <float between -1.0 and 1.0>,
    "efficiency_score": <float between -1.0 and 1.0>,
    "explanation": "<brief reasoning>"
}
Do NOT wrap the JSON in markdown or code fences.
```

E.2.3 ADCA Step Attribution Usage

The ADCA attribution prompt is:

Step-level LLM ADCA Judgment Prompt

```
You are an expert *process reward evaluator*, specializing in **
    attributional analysis** of multi-step solution trajectories.

**INPUT STRUCTURE:** The single message you receive always contains three
    labelled sections:
  1. **TASK DESCRIPTION** The user's original request.
  2. **SOLUTION TRAJECTORY** A strictly numbered list of assistant steps.
      Each step describes an 'ACTION' taken (and optionally an 'OBSERVATION
      ').
  3. **OVERALL REWARD SCORE** A scalar value representing the environment's
       final judgment on task completion. **>0** indicates the task was **
      successfully completed**. **0** indicates the task **failed or was
      incomplete**.

**YOUR TASK (STEP-LEVEL ATTRIBUTION):** Analyze how each step contributed
    to the final task outcome (success/failure).
Judge **each step independently** based only on the provided ACTION/
    OBSERVATION and how later steps use (or fail to use) its results. The
    OVERALL score provides **context**, but must **not** determine labels
    by itself.

**EVALUATION RULES:**

* **If OVERALL REWARD SCORE is POSITIVE (> {THRESHOLD:+.1f}) SUCCESSFUL
    COMPLETION:**
   * Mark a step as **GOOD** if it **directly advanced** the successful
       outcome or **enabled** later successful steps (e.g., establishing a
        needed state, retrieving information that is subsequently used,
       narrowing the solution path, or correctly finalizing).
```

```
    * Mark a step as **BAD** if it was **irrelevant, redundant without new
        effect, later undone without net benefit, or counterproductive**.

* **If OVERALL REWARD SCORE is NON-POSITIVE ( {THRESHOLD:+.1f}) TASK
    FAILURE:**
    * Mark a step as **GOOD** **only if** it **genuinely reduced the
        distance to a correct solution** for example, by establishing a
        necessary precondition later relied upon, correcting an earlier
        direction with observable improvement, validating or narrowing in a
         way later steps actually use, or preventing deterioration that
        would otherwise occur.
    * Mark a step as **BAD** if it **failed to advance** the solution (no
        new effect, outputs unused downstream, or repetition with no added
        value), **obscured or impeded** progress, or **finalized** an
        incorrect result.

**GUARDRAILS:**
* Do **not** label all steps the same unless every step independently meets
     the same criterion.
* **Unused or later-discarded outputs** generally indicate **BAD** unless
    the steps value is still demonstrable elsewhere.
* **Finalization** that submits an incorrect or incomplete outcome must be
    **BAD**.
* Keep analyses **concise** and **text-bound** (no external assumptions).

**FOCUS:** Judge based on **objective contribution to task completion**,
    not effort or intent.

**OUTPUT FORMAT:** Reply IN THE REQUIRED OUTPUT FORMAT and output nothing
    else.
```

E.3 LLM Plan Reasoning Backbone Usage

The planning backbone is used only as a high-level dialogue controller; the model and decoding settings are summarized in Table 19. The prompt template is shown below.

E.3.1 Dialogue-based LLM Prompts

The executable planner prompt is:

Multi-Agent Planning System Prompt

```
{system_tag}You are an expert multi-agent planning system. You coordinate two
     agents to complete household rearrangement tasks as efficiently as
    possible.

## Agent Roles
- Agent 0 (Robot): Object rearrangement only Navigate to objects, Pick them
    up, and Place them on receptacles.
- Agent 1 (Human): Object rearrangement plus additional capabilities wash,
    clean, fill, pour, and turn on/off devices.

## Planning Rules
1. Divide work between agents for maximum parallelism. When multiple objects
    need moving, assign one per agent and move simultaneously.
2. Minimize total actions. Always choose the most direct path to complete the
     task.
3. Explore rooms ONLY when task-relevant object locations are unknown. Never
    explore a room already explored by either agent. If an object is not
    found in one room, try the next most likely room.
```

```
4. If a navigation or rearrangement action is still in progress for an agent,
     reassign that exact same action until a successful execution is observed
     in the agent's observation.
5. Repeating the same failed action multiple times means something is wrong
    try a different action, target, or approach.
6. When one agent finishes its subtask while the other is still executing,
    assign Wait[] to the idle agent.
7. NEVER assign Wait[] to ALL agents at the same time. At least one agent
    must always perform a productive action (Navigate, Pick, Place, Open,
    Close, or other task-relevant action).
8. If you receive messages labeled "Context Update", "Stall Recovery", "
    Feedback", or similar, treat them as read-only observations. Do not
    respond to them directly incorporate the information and continue
    planning for the current task.
9. Do not repeat or quote any context update messages in your output. Output
    only the plan.

## Task Completion Protocol
CRITICAL: You MUST complete ALL of the following checks before declaring the
    task finished:
- Restate the original task description in full.
- Verify that EVERY object mentioned in the task has been successfully placed
     at its specified target location.
- Verify that NO agent is still holding any task-relevant object.
- If ANY condition above is not met, you MUST continue planning. Do NOT
    declare completion prematurely.

## Output Format
Each turn, respond in this exact format:

Thought: <analyze current progress, reason about what each agent should do
    next>
Agent_0_Action: <action for the robot agent>
Agent_1_Action: <action for the human agent>
Assigned!

When ALL task goals are verified complete, use this termination format:

Thought: <restate the task, list each goal and confirm it is satisfied>
Final Thought: Exit!{eot_tag}{user_tag}Task: {input}

{world_description}

Available actions per agent:
{agent_descriptions}

Assign the next action for each agent to make progress toward completing the
    task. Follow the output format specified in the system instructions.{
    eot_tag}{assistant_tag}Thought:
```

E.3.2 LLM Plan Execution and Action Parsing

The action parser connects LLM plan reasoning to embodied execution by converting each agent assignment into a tool name and its arguments. It reads directives in the form `Agent_<id>_Action: Tool[arguments]`, checks the agent identifier and command syntax, and passes valid calls to the agent's configured tool interface. Arguments specify world-graph entities and any required spatial relations or constraints. For example, `Rearrange[pillow_0, on, bed_0, None, None]` requests that the assigned agent move the named pillow onto the bed without an additional spatial constraint.

The selected motor skill resolves these arguments against the world graph and generates low-level actions from the agent's observations. A composite `Rearrange` call proceeds through navigation, picking, transport, and placement, while the individual skills follow the interfaces summarized in

Table 8. Execution status and failure messages return to the planner as agent observations, allowing subsequent decisions to continue an active skill or revise an unsuccessful action. Parsing errors enter the same feedback path, so the next planning request can correct the command before execution.

E.4 Datasets Generation LLM Usage

The prompts in this subsection support Stage I (LLM Instruction Proposal) and Stage III (LLM Evaluation Generation) of the five-stage pipeline in Appendix B.2. The templates illustrate tasks that combine object rearrangement with changes in cleanliness, fill status, or power state. Released model and sampling settings could be reached in Table 19.

Stage I: LLM Instruction and Initial State Proposal. The instruction prompt combines a scene's room–furniture inventory, available object categories, and a reference task. It asks the LLM to adapt the task to that scene and return a JSON list. Each entry pairs an `instruction` with an `initial_state` containing object counts, categories, furniture, and allowed rooms. After parsing and filtering, these specifications pass to Stage II (Simulation Scene Instantiation), which samples object assets and placements and records the realized scene. The following prompt excerpt shows the scene inputs and generation rules.

Instruction generation: template

```
You are a system that generates tasks for robots to perform with humans.
Do not be verbose. Answer the question with no added qualifications or
    caveats. Directly provide the answer in JSON.

You will be given a description of a house with objects and furniture. You
    will also be given a sample task. Your job is to modify this sample task
    so that it is applicable to this house.

Here is an example:
{example_task}

The house has the following rooms, each with the following furniture:
{house_furniture}
You can use the following objects:
{objects_list}
Here is the task:
Task: [
{task}
]

Modify this task to generate a JSON list of tasks, using the rooms and
    furniture from this house.
Just change the objects and furniture.
Make sure initial and final locations of objects are different.
Include actions such as turn on/off, fill and clean.
Make tasks multi-step, consisting of more than one object and action.

Your output should only be:
JSON_OUTPUT: result_list
where result_list should be a JSON list with the tasks.
Generate {k} tasks.
```

Stage III: LLM Evaluation Proposition Generation. Using the scene instantiated in Stage II, the proposition prompt receives the task instruction and the realized entity names, locations, objects and scenes states. A predefined predicate vocabulary covers spatial relations and state conditions, such as `is_on_top`, `is_filled`, and `is_powered_on`. The LLM expresses the task's completion conditions as predicate calls with scene entities as arguments. The excerpt below combines the template's output-format rules with one of its original few-shot examples.

Proposition generation: rules and example

```
Write each function on its own line.
It is essential to wrap each function in delimiters [FN] and [/FN].
End your functions with the token [END].

Instruction: "Move the laptop and mug from the office table to the living
    room cabinet. Turn on the laptop."

[FN] is_on_top("laptop_0", "cabinet in living_room") [/FN]
[FN] is_on_top("mug_0", "cabinet in living_room") [/FN]
[FN] is_powered_on("laptop_0") [/FN]
[END]
```

This generation example yields two placement conditions and one power-state condition, making both aspects of the instruction available to the task evaluator. Separate templates specify temporal ordering and shared or distinct entity assignments when these constraints are required. Appendix B.2 describes how these outputs are assembled with the episode and verified in the Habitat-Sim 3.0 simulator.

Stages IV–V: Episode Assembly and Initialization Verification. Evaluation specifications pass to Stage IV (Episode Assembly), which resolves semantic entity names to simulator handles and combines the propositions and constraints with the episode's initial state and metadata. Stage V (Tasks Initialization Verification) then resets each episode, checks entity references and constraint arguments, and queries the initial completion and success measures. Episodes that pass these checks enter the verified dataset.

E.5 EAS Self-Evolve LLM Usage

For a better analysis interface on resilience evaluation, we mainly take context evolution as the typical mechanism on adaptation. Context evolution turns evidence from completed executions into guidance for later EAS planning. Following Appendix C.4, each version combines the base planner with episode-specific guidance $E_i^{(k)}$ and batch guidance $G^{(k)}$, while retaining the base model parameters and measurement settings. LLM-assisted batch advice guides subsequent attempts; history compression manages dialogue during an ongoing execution. Table 20 distinguishes these uses.

Table 20: **Context used for evolution and ongoing execution.**

Context	Scope	Planner input
Episode guidance $E_i^{(k)}$	Later attempts at episode i	Task history and advice from recorded execution problems.
Batch guidance $G^{(k)}$	Subsequent attempts across tasks	Action recommendations from common failure patterns.
History summary	Current execution	Earlier dialogue and compact scene and agent descriptions.

E.5.1 Execution Evolve Guidance Updates

Episode execution guidance. After each episode, the manager stores the task outcome, completion, simulator steps, and replanning activity, together with available trajectory diagnoses, semantic reward summaries, step attribution, and recovery records. Episode guidance draws on the recorded root causes, improvement suggestions, low-value actions, repeated states, and recovery events. Rules organize this evidence into advice for the next attempt at the same task. For example, recorded placement failures can supply a reminder to check proximity and visibility before retrying.

Batch advice generation. At batch boundaries, the latest episode summaries provide guidance for subsequent tasks, and the rule-based path selects suggestions about recurring execution problems. When an LLM client is attached, the batch-advice prompt provides episode outcomes, completion, and root causes, together with the most frequent failure patterns. It asks for three concise recommendations expressed through available actions. The parser retains up to three items and limits each to 80 characters before forming the batch guidance.

Batch guidance for subsequent executions

```
You are an AI Agent Coach. Analyze the batch of episode summaries and
    generate 3 general advice items.

## Episode Summaries
{batch_summaries}

## Common Failure Patterns
{failure_patterns}

## Available Actions
Navigate[target], Pick[object], Place[object, on, furniture], Rearrange[
    object, on, furniture], Explore[room], Wait[], Done[]

## Rules
1. Each advice MUST reference a specific Action name
2. Each advice MUST be 20 words or less
3. Focus on preventing the most common failures
4. Output ONLY a JSON list: ["advice1", "advice2", "advice3"]

## Output:
```

E.5.2 Suggestions Retrieval and Planner Implementation

Batches updates and injection. At the end of each round, the manager makes the updated episode and batch guidance available for subsequent attempts. For each attempt, it combines the batch guidance with the episode history within the configured context limit, and the next planning request includes this content as feedback. Stored summaries, advice, and context versions record the guidance available to each execution attempt.

Retrieval for the current task. Before replanning, the system queries memory for lessons relevant to the current scene and task, covering ambiguous targets, world-graph inconsistencies, recovery decisions, and skill preferences. The retrieved lessons supply the planner with task guidance, potential risks, recovery suggestions, and preferred skill order. During recovery, the detected fault type provides additional retrieval keys.

History compression within an episode

```
You compress long household-robot planning context while preserving dialog
    form.

## Task Instruction
{task_instruction}

## Protected Entity Names
These object, furniture, and room names must remain verbatim if they are
    mentioned:
{protected_keywords}

## Old Dialog History To Compress
{chat_history}
```

```
## Trajectory Summary
{trajectory_summary}

## World Description
{world_description}

## Agent Description
{agent_description}

## Rules
1. Return ONLY valid JSON.
2. Use this schema:
{{
  "summary_turns": [
    {{"role": "user", "content": "..."}},
    {{"role": "assistant", "content": "..."}}
  ],
  "compressed_world_description": "...",
  "compressed_agent_description": "..."
}}
3. Keep the output compatible with the existing chat log format. Do not
    invent a new section structure.
4. Preserve object/furniture names exactly. Do not rename, alias, or
    abbreviate them.
5. Summarize only older context. Do not rewrite the task, system prompt,
    action schema, or recent turns.
6. Keep only facts useful for the next action: verified locations, completed
    subgoals, failed attempts, blockers, and active commitments.
7. ``summary_turns`` must contain 2 to 6 turns total and should read like a
    compact continuation of the existing dialog.
8. Keep ``compressed_world_description`` and ``compressed_agent_description``
     shorter than the originals while preserving task-relevant entities.

## Output JSON:
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our claims made in the abstract has been reflected the paper's contributions and scope, and proved in our article.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss our work strength and limitations in Appendix B.4, and we provide threats to validity analysis, concrete case study, and data leakage studt. Our work preliminaries are declared in the "Preliminary" subchapter, formalize the representation and the evaluation procedure.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: There's no theoretical results in this paper.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper fully disclose all the information needed to reproduce the main experimental results of the paper, and we release all our code to assist the reproducibility of our experimental results. Appendix B, C and E contain all necessary details for reproducing our results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: This paper provides open access to the data and code, and include instructions for running the code. We report the dataset generation pipeline B.2. We also open and refer our code access in abstract, showing Eas-Resilience-Evaluation.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix A, C, D and E contain our main experimental details and hyperparameters. We also provide the LLM prompt in Appendix E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We provide statistical significance analyses of the Mann–Whitney U test, R^2, and Coefficient of Variation in Sec 3.2.2 and Appendix D.3.1 .

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We include all experiments compute resources in the Appendix D, E, covering all resources used for generating model decision response, and LLM Judge.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We confirm the research conducted in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: We discuss future research work in Sec 5 and broader impact in Appendix **??**.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [Yes]

Justification: We discuss safeguards that have been put in place for responsible release of data or in Appendix B.4, B.3.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The creators or original owners of assets used in the paper are properly credited and are respected for the license and terms of use explicitly mentioned.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.

- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We detailed the documentation in our open code for all assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

    Justification: This work didn't take the experiments on human, our collaboration demonstration is based on the Habitat-Sim 3.0 collaboration interface which detailed in Appendix D.5.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: Our human collaboration decision-making demonstration is innocuous and thus does not require IRB approval.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We describe how LLMs are being used as part of the tools for plan decision making, LLM Judge, and Reward Shaper in Appendix E.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.