WHICH CONDITIONS MATTER? TASK-CONDITIONED CRITICALITY FOR TASK PLAN VERIFICATION AND REPAIR

Anonymous authors
Paper under double-blind review

ABSTRACT

Embodied plan verification must determine not only whether a condition is violated, but also whether that violation matters for the current task. We introduce TCCV, a framework for task-conditioned plan verification and repair. TCCV learns condition criticality from matched simulator interventions and combines it with current environmental evidence to decide whether to execute, observe, repair, or reject a plan. In controlled AI2-THOR experiments, we use structured simulator state to isolate criticality estimation from perception. On 282 score-independent ALFRED `valid_unseen` contrast pairs, TCCV achieves 99.3% decision accuracy with zero false acceptances, compared with 90.2% for adapted VeriGraph and 73.4% for adapted VerifyLLM. A matched-plan diagnostic shows that performance depends on joint task–condition semantics: removing either input reduces ranking to chance. TCCV-guided repair completes 98% of tasks and fixes all 50 injected failures while preserving 96% of initially successful plans. These results support task-conditioned criticality as an auditable interface for embodied plan verification and repair under structured evidence.

1 INTRODUCTION

Embodied agents increasingly use large language models (LLMs), vision–language models (VLMs), and vision–language–action policies to generate and execute plans for household tasks (Ichter et al., 2023; Huang et al., 2023; Singh et al., 2023; Zitkovich et al., 2023; Kim et al., 2025). These systems can produce plausible action sequences, but plausibility does not guarantee physical executability or semantic completion. A plan may refer to a missing or wrong object, assume an unreachable target, invoke an unsupported affordance, or omit an action required for the goal. Such errors can compound over long horizons or lead to unsafe behavior. Pre-execution verification is therefore important before embodied control.

A central challenge is that not every violated condition matters equally. Consider *pick up any cup* and *pick up the red cup*: a color mismatch is irrelevant to the first task but decisive for the second. Similarly, a microwave may start closed if the plan opens it before use. Condition importance therefore depends on the instruction, object role, temporal scope, scene, and consequence of violation. We call this quantity *task-conditioned criticality*. For a temporally grounded obligation, criticality measures how likely its violation is to change semantic task success under audited matched interventions. The model learns this relation across related contexts rather than treating a predicate as globally important.

This view separates three quantities. *Satisfaction* describes whether an obligation currently holds. *Criticality* describes whether its violation changes task success. *Evidence uncertainty* describes whether its current state is known reliably. These distinctions enable different responses to different failures: an irrelevant mismatch can be ignored, an unresolved critical condition can trigger observation, and a confirmed critical violation can trigger repair or rejection. We further distinguish semantic completion from controller success to avoid interpreting executor limitations as task-level necessity.

We introduce task-conditioned criticality for task plan verification and repair (TCCV). TCCV constructs temporally grounded obligations from the instruction, candidate plan, action interface, and goal predicates. It learns their success-flip effects from matched simulator interventions around audited successful anchors. At verification time, environmental evidence is combined with learned

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

criticality to localize execution risk. TCCV can then execute the plan, request additional evidence, or target the earliest high-risk obligation for bounded repair. Safety invariants remain hard constraints.

Our contributions are:

- We identify and address the problem of task-dependent condition relevance in embodied plan execution. Instead of treating environmental conditions as universally critical or non-critical, we formulate condition criticality as a context-dependent counterfactual effect on task success, enabling the model to understand which conditions truly determine execution outcomes under different tasks.
- We propose TCCV, a task-conditioned criticality verification framework that learns execution-grounded condition criticality through counterfactual interventions and uses it to selectively verify task-relevant obligations. Unlike existing approaches based on fixed predicate sets or predefined hard/soft constraints, TCCV assigns context-dependent criticality scores to candidate obligations instantiated from the current task and environment.
- We develop a closed-loop verification and repair mechanism that leverages learned condition criticality to diagnose failure causes, acquire necessary environmental evidence, and guide bounded LLM-based plan revision. This enables embodied agents to detect unreliable plans and repair predicted plan defects before execution through targeted intervention.

The remainder of the paper is organized as follows. Section 2 presents the motivation for this work. Section 3 reviews related work. Section 4 develops the formal formulation of TCCV. Section 5 presents the experimental evaluation. Section 6 discusses the limitations and concludes the paper.

2 Motivation and Design Requirements

Why explicit matching is insufficient. A verifier must distinguish a harmful violation from a mismatch that is irrelevant or repaired by the plan itself. For example, a closed microwave need not invalidate a plan that opens it before placement; mug existence is critical when the instruction names that mug but may be irrelevant when any suitable container satisfies the goal. Current satisfaction, observation uncertainty, and task-conditioned consequence are therefore different quantities. TCCV separates them so that a consequential unknown triggers observation, a consequential violation triggers repair, and an irrelevant mismatch does not block execution.

Motivating diagnostic. We tested ten conditions in matched true/false states in AI2-THOR FloorPlan1, yielding 20 controller rollouts. Table 1 shows representative cases. Plan/PDDL records whether the generated plan explicitly mentioned the condition; Score is Qwen's self-reported importance on a 0–100 scale; Exec. is the paired execution-derived criticality. Across the diagnostic, plan/PDDL mentions recovered only 16.1% of execution-critical conditions, and the importance scores had a mean absolute error of 48 points. The error reached 93 points for initially unsatisfied but plan-recoverable conditions. These results are a small motivation study rather than a benchmark, but they expose the failure mode addressed by matched, execution-grounded supervision.

Table 1: Representative cases from the 10-condition motivation diagnostic. LLM scores are self-reported; execution criticality is derived from matched rollouts.

Condition context	Plan/ PDDL	Score	Exec.
Mug exists, specific target	Omitted	100	100
Mug exists, any-object task	Omitted	0	0
Microwave initially open	Mentioned	100	0

3 Related Work

Embodied planning and recovery. SayCan, Inner Monologue, ProgPrompt, and SayPlan ground language plans in affordances, feedback, programs, or scene graphs (Ichter et al., 2023; Huang et al.,

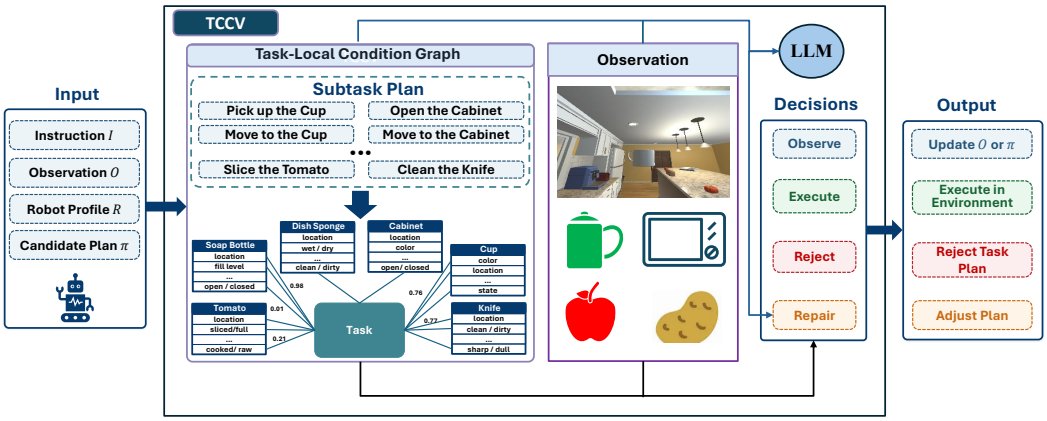


Figure 1: Overview of the proposed TCCV framework.

2023; Singh et al., 2023; Rana et al., 2023); RT-2 and OpenVLA connect vision and language to control (Zitkovich et al., 2023; Kim et al., 2025). Recent systems add semantic histories, safety prediction, belief-space replanning, reflective correction, calibrated failure prediction, uncertainty decomposition, or multi-agent coordination (Jeong et al., 2025; Li et al., 2025; Tang et al., 2025; Feng et al., 2025; Römer et al., 2025; Yin et al., 2025; Kang et al., 2025). These methods improve plan generation and execution; TCCV instead attributes the risk of a produced plan to individual task-dependent conditions.

Verification and structured reasoning. Classical validation assumes known predicates and action schemas (Fox & Long, 2003; Baier & Katoen, 2008). VerifyLLM, HVR, Code-as-Symbolic-Planner, iterative LLM judges, and VeriGraph add temporal, knowledge-graph, code, critique, or scene-graph reasoning (Grigorev et al., 2025; Petruzzellis et al., 2025; Chen et al., 2025; Hariharan et al., 2025; Ekpo et al., 2026). Lifted HTN verification, counter-example-based domain correction, PlanOwl, and MRBTP similarly check or repair explicit planning structures (Lauer et al., 2025; Lin et al., 2025; Adamik & Forte, 2025; Cai et al., 2025). TCCV is complementary: it learns empirical success sensitivity rather than treating every extracted predicate as equally binding.

Failure recovery and safety. REFLECT and proactive scene-graph replanning use experience or predicted downstream failures to correct behavior (Liu et al., 2023; Yu et al., 2025); safety-aware planners, calibrated failure predictors, and VestaBench study whether actions should execute under risk (Li et al., 2025; Römer et al., 2025; Sadhu et al., 2025). TCCV separates uncertainty (whether a condition holds) from criticality (whether its violation changes task success), while keeping declared safety invariants as hard constraints.

4 Task Plan Formulation and TCCV Method

Given instruction I, evidence O, robot profile R, and plan $\pi = (a_1, \ldots, a_T)$, TCCV represents temporally grounded obligations, learns their task-dependent consequence, and combines it with current evidence.

Figure 1 shows the resulting separation. The obligation extractor proposes task- and step-specific questions; matched interventions supervise the consequence of changing each question; and an evidence provider evaluates its current truth value. The decision layer never treats criticality alone as proof of failure: a highly critical condition creates risk only when it is violated or unresolved.

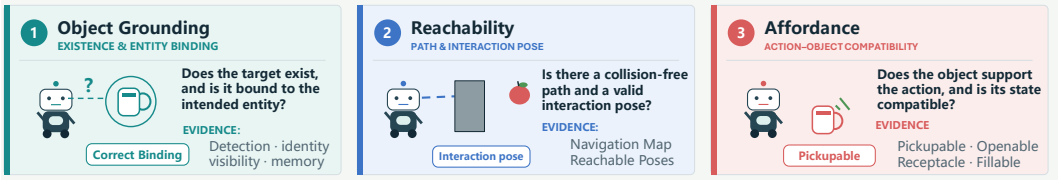


Figure 2: Three families of grounded obligations represented by the environment adapter. Object grounding checks existence and entity binding; reachability checks navigation and interaction poses; affordance checks action–object and state compatibility. The taxonomy specifies *what* evidence to request, while TCCV separately learns whether each grounded obligation matters for the current task.

4.1 Outcomes and Obligations

Let A_t indicate controller success at step t, G semantic goal completion, and S safety satisfaction. We record controller and semantic outcomes separately:

$$Y_\pi^{\text{ctrl}} = \bigwedge_{t=1}^{T} A_t, \qquad Y_\pi^{\text{sem}} = G \wedge S, \tag{1}$$

Semantic success supplies the primary criticality target; controller success is an auxiliary diagnostic, preventing executor weakness from being interpreted as universal task relevance.

This distinction is essential when the reference executor lacks a capability. For example, failure to discriminate object color may cause repeated controller failures even though color is irrelevant to a generic pickup instruction. Such failures belong to the executor head unless the final semantic goal itself requires that color. Conversely, a sequence may execute without API errors but terminate in the wrong semantic state. Reporting both outcomes prevents either case from contaminating the criticality interpretation.

Each step induces obligations $\mathcal{O}_t = \{c_{t,1}, \ldots, c_{t,m_t}\}$, where $c_{t,j} = (p, \theta, t, \kappa)$ specifies a predicate, grounded arguments, step, and initial/action-time/invariant/terminal scope. The task-local graph $\mathcal{G}_C = (V, A)$ links prerequisite and state-flow dependencies. An environment-specific extractor instantiates nodes from the instruction, plan, action schemas, and goal predicates; omitted obligations cannot be recovered by the criticality head during that episode.

Edges encode only auditable dependencies: a prerequisite edge states that one obligation must hold before another can be tested, and a state-flow edge propagates the effect of an action to later steps. Temporal scope avoids a common category error. A receptacle may be allowed to start closed if the plan opens it before placement, whereas the same receptacle must be open at the placement step. Safety invariants are represented in the graph but bypass learned relaxation. The graph therefore organizes candidate checks; it is not assumed to be a complete symbolic world model.

Figure 2 illustrates the environment-facing taxonomy used in our AI2-THOR instantiation. Other platforms may expose different predicates, but they retain the same separation between candidate-obligation generation, current evidence, and learned task-conditioned consequence.

4.2 Success-Flip Criticality

For obligation $c_{t,j}$, let $X_{t,j} = (I, O, R, \pi, c_{t,j}, \mathcal{G}_C)$ and $Y_{t,j}^k(v)$ be the semantic or controller outcome after a matched intervention sets the obligation to $v \in \{0, 1\}$. A pair supplies binary outcomes, not a repeated-trial probability for one scene. We therefore learn across related contexts using audited successful anchors $H_{t,j} = 1$.

For every source trajectory, we first restore a successful anchor and verify that controller execution, semantic completion, safety, and non-target state are consistent. We then alter exactly one supported condition, reset the simulator, and replay the same plan. Pairs are rejected when the edit changes additional task-relevant facts, produces an invalid state, or cannot be verified at the intervention time. This matched design is what gives a flip label its local interpretation; merely contrasting unrelated successful and failed episodes would confound condition relevance with plan and scene differences.

The semantic flip label and criticality estimand are

$$F_{t,j}^{\text{sem}} = \mathbf{1}\left[Y_{t,j}^{\text{sem}}(1) = 1 \ \wedge \ Y_{t,j}^{\text{sem}}(0) = 0\right], \tag{2}$$

and

$$\alpha_{t,j}(x) = P\big(F_{t,j}^{\text{sem}} = 1 \mid X_{t,j} = x, H_{t,j} = 1\big). \tag{3}$$

This population-relative estimand assumes consistency, audited isolation, and reset-based non-interference; it is not an unconditional causal effect of a naturally occurring condition. Only $(1, 0)$ flips and $(1, 1)$ non-flips enter ordinary training. We quarantine $(0, 0)$ as unidentified and $(0, 1)$ as reversal/audit rather than labeling either negative. With $\widehat{\alpha}_{t,j} = \sigma(f_{\text{crit}}(f_{\text{enc}}(X_{t,j})))$, mask M for eligible audit-passing pairs, label $y = 1 - Y^{\text{sem}}(0)$, and balancing weight w, we optimize

$$\mathcal{L}_{\text{crit}} = \frac{1}{\sum_{t,j} M_{t,j} w_{t,j}} \sum_{t=1}^{T} \sum_{j=1}^{m_t} M_{t,j} w_{t,j} \, \text{BCE}\big(\widehat{\alpha}_{t,j}, y_{t,j}\big). \tag{4}$$

An auxiliary controller-flip head diagnoses executor sensitivity but does not replace semantic criticality. Full intervention assumptions, masking, and four-pattern handling appear in Appendix D.

The encoder represents the instruction, candidate obligation, its temporal scope, plan prefix, scene summary, and robot profile. The criticality head receives both semantic embeddings and explicit relation features $[p, c, |p - c|, p \odot c]$. In addition to the masked binary loss, matched ranking margins require a task-relevant condition to score above a neutral or mismatched condition drawn from the same audited context. This discourages memorizing that a predicate such as `isHot` is globally important and instead rewards alignment between the instruction and the candidate condition.

4.3 Evidence, Risk, and Decision

An external provider returns $s_{t,j} \in \{1, 0, -1\}$ (satisfied, violated, unknown) and, when available, violation probability $q_{t,j}$. Unknown critical evidence triggers Observe; it is not imputed as success. The extractor determines *what to check*, whereas the provider reports only its current state. Our experiments instantiate this interface with structured AI2-THOR metadata; the resulting claims therefore evaluate the criticality layer under structured evidence rather than an end-to-end perception system. The provider never receives intervention identities, outcomes, or future states.

In the deterministic metadata setting, $q = 0$ for a satisfied condition, $q = 1$ for a verified violation, and the state is unknown when the required field is absent or the target is not observable under the interface. A learned provider may instead return calibrated uncertainty. TCCV consequently does not require visual recognition to be part of its contribution, but it does require the provider's error modes to be measured: confidently wrong evidence can defeat any downstream verifier, whereas missing evidence can be handled through abstention and observation.

TCCV combines consequence and evidence as

$$r_{t,j} = \widehat{\alpha}_{t,j} q_{t,j}. \tag{5}$$

Thus irrelevant violations and satisfied critical obligations contribute no risk. TCCV aggregates by bottleneck:

$$B_t = \max_{c_{t,j} \in \mathcal{O}_t} r_{t,j}, \qquad B_\pi = \max_{1 \le t \le T} B_t. \tag{6}$$

These are diagnostic risks, not calibrated joint failure probabilities.

The maximum aggregation implements a bottleneck view of sequential execution and preserves localization. We avoid multiplying condition probabilities because the obligations are dependent and $\widehat{\alpha}$ is not a calibrated hazard rate. Other monotone aggregators are possible, but they must retain the distinction between consequence and current evidence.

Safety invariants remain hard constraints. With validation-fixed thresholds, TCCV observes unresolved critical evidence, executes when $B_\pi \le \tau_{\text{exec}}$, and otherwise selects

$$t^* = \min\{t : B_t > \tau_{\text{repair}}\}. \tag{7}$$

or rejects if no admissible repair exists.

4.4 EXPERIENCE-GUIDED REPAIR

Figure 1 summarizes the loop. TCCV packages the earliest risky obligation, its criticality and evidence, and aligned successful plans from training-only memory. An LLM proposes a bounded edit; a deterministic validator rejects malformed, ungrounded, or goal-breaking revisions. After any accepted edit, TCCV rebuilds downstream obligations and risks before another decision, avoiding stale repairs. Operational repair can apply whenever the interface supplies an obligation and evidence, but causal interpretation of $\widehat{\alpha}$ remains limited to the audited anchor population and intervention support; success beyond that support demonstrates predictive control, not a newly identified causal effect. Appendix D gives the full policy and prompts.

5 EXPERIMENTS

We evaluate TCCV on unseen AI2-THOR task contexts to test both its learned criticality estimates and their effect on verification and repair. The experiments focus on pre-execution decisions under validation-selected thresholds, followed by execution-based evaluation of repaired plans.

5.1 EXPERIMENTAL SETUP

We use ALFRED (Shridhar et al., 2020) in AI2-THOR (Kolve et al., 2017). From 1,000 successful source trajectories, we generate 8,000 matched step–obligation interventions. Each successful arm is reconstructed and replayed before intervention, and must pass controller, semantic-goal, safety, and non-target-change audits. Semantic and controller outcomes are stored separately. Source trajectories are assigned to disjoint train, validation, and test partitions before deriving intervention pairs, preventing trajectory-level leakage. Exact split counts are provided in Appendix D. Structured AI2-THOR metadata supplies only current evidence, never obligations, labels, outcomes, or future states. We therefore evaluate TCCV as a criticality layer under structured evidence.

The interventions cover object existence and identity, receptacle availability, held-object state, clean/dirty, heated/cooled, sliced/unsliced, and temporal ordering. Invalid anchors, simulator failures, and interventions with unintended state changes are excluded from criticality supervision. The compact Transformer predicts semantic flip, controller flip, and step scope using masked BCE and matched ranking losses. The text encoder is frozen, while the task–condition interaction module and prediction heads are trained on the intervention pairs. All hyperparameters, thresholds, and budgets are selected on validation data. Results are averaged over seeds 7, 17, and 27. Appendix D provides the full collection audit, architecture, optimization settings, multi-defect protocol, prompts, and simulator operations.

5.2 EVALUATION PROTOCOL

E2 uses 2,916 train, 80 distribution-matched validation, and all 282 `valid_unseen` contrast pairs from disjoint trajectories; every evaluated task-type–condition cell has at least five training pairs. The threshold maximizes validation pass rate subject to FAR ≤ 0.10. We compare a **uniform hard validator (UHV)**, **static criticality (SCV)**, a **direct Qwen-Plus judge (DLJ)**, and AI2-THOR adaptations of **VerifyLLM** (Grigorev et al., 2025) and **VeriGraph** (Ekpo et al., 2026). All receive aligned task, plan, capability, and current-evidence inputs where their interfaces permit, but no labels, intervention identities, future states, or test statistics. **Same-encoder direct success** is used only as an objective control in E1. Adaptation and prompt details are in Appendix E.1.

UHV blocks any explicit violation, while **SCV** assigns a task-independent score to each condition type. **DLJ** directly predicts plan success from the complete textual input. **VerifyLLM** converts the task and plan into constraints; **VeriGraph** compares the desired and current structured scene. These baselines distinguish three alternatives to TCCV: universal condition matching, global outcome prediction, and structured symbolic/graph checking. For fairness, operating points are frozen before test scoring and every method is evaluated on the same successful and failed member of each pair.

We report accuracy, successful-plan pass rate, failure blocking, FAR, and FRR; repair additionally uses completion, repair success, success preservation, and attempts. Appendix B reports held-out composition, 0/1/2/3-defect stress, executor stratification, and the structured-evidence corruption

Table 2: Comprehensive learned-estimator ablation on 30 held-out matched-intervention triplets (90 contexts; five triplets per condition family). Retrained criticality variants change only the named component. Bold marks the best value in a column, with ties. Here, p and c denote the semantic embeddings of the task instruction and the candidate condition, respectively

Variant	Bal. Acc. ↑	AUROC ↑	Matched rank ↑	Mismatch negative ↑
Objective and full model				
Same-input/MLP direct success	0.767	0.870	0.767	0.967
Full TCCV	0.792	0.886	0.800	**1.000**
Supervision and ranking				
− neutral-pair margin	0.775	0.886	0.800	**1.000**
− mismatch-pair margin	0.783	0.881	0.800	0.967
− both margins (criticality BCE)	0.783	0.856	0.733	**1.000**
− class balancing	0.792	**0.894**	0.800	**1.000**
Task–condition representation				
Concatenation only $[p, c]$	0.700	0.812	0.633	0.967
$-\lvert p - c\rvert$	0.758	0.871	0.767	**1.000**
$-p \odot c$	0.758	0.838	0.733	0.967
Interactions only $[\lvert p - c\rvert, p \odot c]$	**0.808**	0.878	**0.833**	0.967
Instruction only	0.642	0.670	0.500	0.600
Candidate condition only	0.500	0.500	0.500	0.833
Shuffled candidate condition	0.500	0.499	0.500	0.367
Inference robustness				
Single seed (17)	0.783	0.872	0.800	**1.000**
Fixed threshold 0.5	0.792	0.886	0.800	**1.000**

audit. The corruption audit shows graceful abstention under missing evidence but rising FAR under severe confidently wrong evidence (Appendix B.2).

Decision accuracy alone can conceal an unusably conservative verifier, so pass and block rates are always reported together. FAR is the fraction of failed plans incorrectly executed; FRR is the fraction of successful plans unnecessarily blocked. Within-pair ranking tests whether the violated member receives higher risk than its matched successful counterpart without committing to one threshold. Repair completion is measured by semantic goal evaluation after executing the revised plan, not by an LLM judgment.

5.3 Main Experiments

E1: What does each learned-estimator component contribute? We hold scene, plan, evidence, and candidate condition fixed while contrasting neutral, required, and mismatched instructions in 30 held-out triplets (90 contexts). Table 2 shows that aligned task and condition inputs are essential: either single-input variant and a shuffled condition reduce ranking to chance. Raw concatenation loses 9.2 balanced-accuracy and 16.7 ranking points, while removing both matched margins loses 6.7 ranking points. Full TCCV exceeds the same-input direct-success control by 2.5 balanced-accuracy and 3.3 ranking points, but bootstrap intervals include zero; we therefore claim mechanism-level diagnostic value, not universal superiority. Protocols, intervals, and family slices are in Appendix B.1.

All retrained variants use the same 580 training triplets, 24 validation triplets, 30 test triplets, frozen encoder, optimization budget, and checkpoint rule. The direct-success control changes only the target and loss; it is therefore a test of whether the condition-level objective adds information beyond the same representation. A 10,000-replicate cluster bootstrap gives full-minus-instruction-only intervals of $[7.5, 23.3]$ balanced-accuracy points and $[13.3, 46.7]$ ranking points. Full TCCV's smaller advantage over direct success is not statistically decisive, but the collapse under removing or shuffling either semantic input is strong evidence that the model is using their relation rather than a condition-frequency shortcut.

Table 3: Support-constrained E2 comparison with external verifiers on all 282 score-independent `valid_unseen` pairs (564 contexts). The TCCV threshold is fixed on a distribution-matched, trajectory-disjoint 80-pair ALFRED-train validation set under FAR ≤ 0.10.

Method	Acc. ↑	Pass ↑	Block ↑	FAR ↓	FRR ↓
UHV	0.500	0.000	**1.000**	**0.000**	1.000
SCV	0.500	0.103	0.897	0.103	0.897
DLJ (Qwen-Plus)	0.541	0.082	**1.000**	**0.000**	0.918
VerifyLLM (adapt.)	0.734	0.486	0.982	0.018	0.514
VeriGraph (adapt.)	0.902	0.855	0.950	0.050	0.145
TCCV	**0.993**	**0.986**	**1.000**	**0.000**	**0.014**

Table 4: Repair comparison on 100 AI2-THOR task contexts derived from official ALFRED high-level plans. Comp., Succ., Pres., and Att. denote completion, repair success, success preservation, and average attempts, respectively.

Policy	Final Comp. ↑	Repair Succ. ↑	Success Pres. ↑	Avg. Att. ↓
No Repair	0.500	0.000	**1.000**	**0.000**
LLM Repair	0.790	0.700	0.880	1.000
VeriGraph	**1.000**	**1.000**	**1.000**	0.500
TCCV	0.980	**1.000**	0.960	0.500

E2: Does task-conditioned criticality improve pre-execution verification? Table 3 compares validation-selected execute/block decisions. TCCV reaches 99.3% accuracy, 98.6% pass, zero FAR, and perfect within-pair ranking; its four errors are conservative rejections. VeriGraph reaches 90.2%, VerifyLLM 73.4%, and the direct Qwen-Plus judge 54.1%. VeriGraph's main failure is treating an action-time held-object requirement as a terminal graph match. E2 therefore establishes learning and held-out transfer for supported explicit contrasts, not open-ended generalization.

The gap between E1 and E2 is intentional and informative. E2 contains explicit, support-constrained contrast pairs for which current evidence and task–condition alignment are highly diagnostic; it therefore tests the complete decision pipeline. E1 removes those easy differences and asks whether the learned objective distinguishes task relevance under matched inputs. We consequently do not attribute E2's 99.3% solely to the criticality learner or claim that it dominates direct success on arbitrary plans. Instead, E2 shows that the decomposition can turn reliable evidence and supported condition knowledge into selective decisions, while E1 identifies the narrower learned mechanism.

E3: Can TCCV-Guided Repair Recover Failed Plans? On 50 successful plans and 50 disjoint plans with injected subtle failures, Table 4 compares no repair, plain Qwen, VeriGraph, and TCCV-guided Qwen. TCCV completes 98%, fixes all injected failures, preserves 96% of valid plans, and halves attempts relative to plain repair. VeriGraph reaches 100%, showing that global scene-graph context is beneficial in this controlled injected-failure setting.

The failing plans are produced by deleting, substituting, or reordering one high-level action while preserving syntactic validity. Plain repair receives only the instruction and candidate plan; TCCV additionally supplies the earliest high-risk obligation and its current evidence. VeriGraph supplies its global graph diagnosis. All repairs pass the same deterministic grounding and goal-preservation checks before execution, so the comparison isolates the information used to target a shared LLM backend.

Plain Qwen fixes 35/50 failures but invalidates 6/50 valid plans; TCCV fixes 50/50 and invalidates 2/50. Because TCCV and plain repair share the same LLM backend, the difference isolates the value of deciding *whether* to repair and supplying a localized condition target rather than using a stronger generator. VeriGraph's perfect result further suggests that global scene structure and learned condition criticality are complementary. Complete repair traces are reported in Appendix E.2.

Table 5: E4 generalization evaluation on 25 complete pairs (50 contexts) from ALFRED `valid_unseen`. Selection is independent of model scores, the source trajectories are absent from training and validation, and all operating points are frozen before evaluation.

Method	Acc. ↑	Pass ↑	Block ↑	FAR ↓	FRR ↓
UHV	0.500	0.000	1.000	0.000	1.000
SCV	0.500	0.040	0.960	0.040	0.960
DLJ (Qwen-Plus)	0.560	0.120	**1.000**	**0.000**	0.880
TCCV	**1.000**	**1.000**	**1.000**	**0.000**	**0.000**

E4: Does TCCV generalize beyond the training distribution? E4 evaluates transfer from ALFRED training trajectories to independently selected `valid_unseen` trajectories. The set contains 25 complete pairs sampled with seed 20260927 from 25 support-constrained E2 test trajectories, is source-disjoint from training and validation, and is selected without access to any model score. It covers unseen source trajectories and scenes together with task–condition combinations that differ in composition from the training split. All thresholds, model weights, obligation interfaces, and structured-evidence rules are frozen before evaluation.

Table 5 shows that TCCV classifies all 50 contexts correctly, whereas the direct Qwen-Plus judge reaches 0.560 accuracy and rejects 88% of successful plans. Because the subset contains unseen ALFRED trajectories and scenes and is selected independently of all model outputs, the result provides evidence that the learned task–condition relation transfers beyond the training trajectories to additional `valid_unseen` data. The experiment remains within the ALFRED/AI2-THOR domain and is support-constrained, so it demonstrates in-domain distributional generalization rather than cross-platform generalization.

Additional robustness evidence. We keep secondary stress tests out of the main tables to preserve space, but summarize their implications. On 43 anchors with 0/1/2/3 simultaneously valid defects, TCCV detects the presence of any defect reliably, while recall of every individual defect drops to 51.2% and 48.8% for two and three defects. This exposes a limitation of single-intervention supervision under interacting failures. Under 10% synthetic structured-evidence corruption, TCCV reaches 0.756 accuracy; at 20–30%, FAR rises to 0.310–0.327. Missing evidence can trigger abstention, whereas confidently incorrect evidence remains a major deployment risk. Full tables and protocols are in Appendix B.

6 Limitations and Conclusion

Limitations. Our current evaluation focuses on ALFRED tasks in AI2-THOR with structured simulator evidence, providing a controlled setting for studying task-conditioned criticality. TCCV is not tied to a specific simulator. Its obligation interface and evidence provider can be adapted to new environments, while the criticality estimator can be retrained for the new setting. Extending TCCV to other simulators, perception-based evidence, and real robotic systems is an important direction for future work. The current experiments also use templated contrast instructions and injected repair failures. Broader linguistic variation and naturally occurring failures remain to be studied. Finally, criticality is learned from the observed anchor and intervention distributions, so substantial changes in the task domain or obligation set may require additional training data.

Conclusion. We introduced TCCV, a framework for embodied plan verification and repair based on task-conditioned criticality. Matched interventions estimate whether violating an obligation changes semantic task success. Current environmental evidence then determines whether that obligation creates execution risk. Experiments show that TCCV learns supported task–condition relations, transfers to held-out trajectories, and relies on aligned task and condition semantics. The framework also enables targeted repair through auditable condition-level diagnoses. Future work should evaluate the interface with additional evidence providers, expand intervention coverage, and test broader tasks and real robotic systems.

REPRODUCIBILITY STATEMENT

We provide an anonymized supplementary package containing the core TCCV implementation, evaluation scripts, configuration files, and compact result artifacts used to audit the reported experiments. The package supports inspection and partial reproduction of the main experimental pipeline. Full-scale reproduction additionally requires the publicly available ALFRED and AI2-THOR assets, as well as generated datasets and model checkpoints that are not included in the anonymous submission. We plan to release the complete codebase and the remaining artifacts after the double-blind review period.

AI USE STATEMENT

In this work, we used generative AI tools for language editing, drafting and debugging experiment code, constructing evaluation prompts, and organizing diagnostic analyses. Generative models used as experimental components for plan judgment, evidence extraction, or repair are identified in the corresponding protocols. We defined the research questions and evaluation rules, inspected the code, executed the experiments, checked every reported value against saved artifacts, and verified the cited works and the claims attributed to them. LLM-generated code was verified and tested for correctness by 3 authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Mark Adamik and Paolo Forte. PlanOwl: Automated PDDL files generation from OWL ontologies and visual language models. In *Proceedings of the AAAI Symposium Series*, volume 7, pp. 634–643, 2025. doi: 10.1609/aaaiss.v7i1.36944. URL https://ojs.aaai.org/index.php/AAAI-SS/article/view/36944.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

Yishuai Cai, Xinglin Chen, Zhongxuan Cai, Yunxin Mao, Minglong Li, Wenjing Yang, and Ji Wang. MRBTP: Efficient multi-robot behavior tree planning and collaboration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 14548–14557, 2025. doi: 10.1609/aaai.v39i14.33594. URL https://ojs.aaai.org/index.php/AAAI/article/view/33594.

Yongchao Chen, Yilun Hao, Yang Zhang, and Chuchu Fan. Code-as-symbolic-planner: Foundation model-based robot planning via symbolic code generation. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 19248–19254. IEEE, 2025. doi: 10.1109/IROS60139.2025.11247174. URL https://arxiv.org/abs/2503.01700.

Daniel Ekpo, Mara Levy, Saksham Suri, Chuong Huynh, Archana Swaminathan, and Abhinav Shrivastava. VeriGraph: Scene graphs for execution verifiable robot planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2026. URL https://arxiv.org/abs/2411.10446.

Yunhai Feng, Jiaming Han, Zhuoran Yang, Xiangyu Yue, Sergey Levine, and Jianlan Luo. Reflective planning: Vision-language models for multi-stage long-horizon robotic manipulation. In *Proceedings of the Conference on Robot Learning*, volume 305, pp. 2038–2062, 2025. URL https://proceedings.mlr.press/v305/feng25b.html.

Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003. doi: 10.1613/jair.1129.

Danil S. Grigorev, Alexey K. Kovalev, and Aleksandr I. Panov. VerifyLLM: LLM-based pre-execution task plan verification for robots. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 18489–18496. IEEE, 2025. doi: 10.1109/IROS60139.2025.11246320. URL https://arxiv.org/abs/2507.05118.

Ananth Hariharan, Vardhan Dongre, Dilek Hakkani-Tür, and Gokhan Tur. Plan verification for LLM-based embodied task completion agents. *arXiv preprint arXiv:2509.02761*, 2025. URL https://arxiv.org/abs/2509.02761.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tobin Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T. Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 2023. URL https://proceedings.mlr.press/v205/ichter23a.html.

Soojin Jeong, Seongwan Byeon, Sangwoo Kim, HyeokJun Kwon, and Yoonseon Oh. GNN-Transformer task planning enhanced with semantic-driven data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 14585–14593, 2025. doi: 10.1609/aaai.v39i14.33598. URL https://ojs.aaai.org/index.php/AAAI/article/view/33598.

Li Kang, Xiufeng Song, Heng Zhou, Yiran Qin, Jie Yang, Xiaohong Liu, Philip H. S. Torr, Lei Bai, and Zhenfei Yin. VIKI-R: Coordinating embodied multi-agent cooperation via reinforcement learning. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/ab16cc8f472cf94b282dbff7b99f5f8e-Abstract-Datasets_and_Benchmarks_Track.html.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An interactive 3d environment for visual ai. *CoRR*, abs/1712.05474, 2017. URL https://arxiv.org/abs/1712.05474.

Pascal Lauer, Songtuan Lin, and Pascal Bercher. Tight bounds for lifted HTN plan verification and bounded plan existence. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 35, pp. 64–73, 2025. doi: 10.1609/icaps.v35i1.36102. URL https://ojs.aaai.org/index.php/ICAPS/article/view/36102.

Siyuan Li, Feifan Liu, Lingfei Cui, Jiani Lu, Qinqin Xiao, Xirui Yang, Peng Liu, Kewu Sun, Zhe Ma, and Xun Wang. Safe planner: Empowering safety awareness in large pre-trained models for robot task planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 14619–14627, 2025. doi: 10.1609/aaai.v39i14.33602. URL https://ojs.aaai.org/index.php/AAAI/article/view/33602.

Songtuan Lin, Alban Grastien, Rahul Shome, and Pascal Bercher. Told you that will not work: Optimal corrections to planning domains using counter-example plans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26596–26604, 2025. doi: 10.1609/aaai.v39i25.34861. URL https://ojs.aaai.org/index.php/AAAI/article/view/34861.

Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing robot experiences for failure explanation and correction. In *Proceedings of the 7th Conference on Robot Learning*, volume

229 of *Proceedings of Machine Learning Research*, pp. 3468–3484. PMLR, 2023. URL https://proceedings.mlr.press/v229/liu23g.html.

Flavio Petruzzellis, Cristina Cornelio, and Pietro Liò. Hierarchical planning for complex tasks with knowledge graph-RAG and symbolic verification. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 49105–49127. PMLR, 2025. URL https://proceedings.mlr.press/v267/petruzzellis25a.html.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3D scene graphs for scalable robot task planning. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 23–72. PMLR, 2023. URL https://proceedings.mlr.press/v229/rana23a.html.

Ralf Römer, Adrian Kobras, Luca Worbis, and Angela P. Schoellig. Failure prediction at runtime for generative robot policies. In *Advances in Neural Information Processing Systems*, volume 38, pp. 8509–8548, 2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/0b7cb3b8cc44e652761245537027db44-Abstract-Conference.html.

Tanmana Sadhu, Yanan Chen, and Ali Pesaranghader. VestaBench: An embodied benchmark for safe long-horizon planning under multi-constraint and adversarial settings. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 2122–2145, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-industry.149. URL https://aclanthology.org/2025.emnlp-industry.149/.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10740–10749, 2020.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 11523–11530, 2023. doi: 10.1109/ICRA48891.2023.10161317.

Wenjing Tang, Xinyu He, Yongxi Huang, Yunxiao Xiao, Cewu Lu, and Panpan Cai. TRU-POMDP: Task planning under uncertainty via tree of hypotheses and open-ended POMDPs. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4fb6b4482e4e24a33a3de5b01a011eb0-Paper-Conference.pdf.

Shiyuan Yin, Chenjia Bai, Zihao Zhang, Junwei Jin, Xinxin Zhang, Chi Zhang, and Xuelong Li. Towards reliable LLM-based robot planning via combined uncertainty estimation. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/6fcbfb3721c1781728b10c6685cc2f6c-Abstract-Conference.html.

Che Rin Yu, Daewon Chae, Dabin Seo, Sangwon Lee, Hyeongwoo Im, and Jinkyu Kim. Scene graph-guided proactive replanning for failure-resilient embodied agent. *arXiv preprint arXiv:2508.11286*, 2025. URL https://arxiv.org/abs/2508.11286.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski,

Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

A Supplementary Material

This appendix provides the full supplementary record for the experiments in the main paper. Section C gives the motivating LLM self-assessment diagnostic. Section D describes the shared data construction protocol and TCCV benchmark. Section E reports the E2–E4 evaluation datasets and released artifacts. Section B includes the mechanism-isolation and structured-evidence corruption audits. The remaining sections document experiment-specific data, training parameters, and baseline details.

B Diagnostic Evaluations

We report these diagnostics separately because their evaluation units differ from the E2 comparison. All splits are fixed before training. Values after $\pm$ are sample standard deviations over seeds.

The held-out-composition split contains 4,532 training, 592 validation, and 844 test records. Test anchors never occur in training or validation; the six held-out task–condition pairs are absent from both, while each constituent task and condition remains observed separately. TCCV obtains goal AUROC 0.99982 ± 0.00023 and AUPRC 0.99984 ± 0.00019 over three seeds. These scores are split-valid, but deterministic contrast/template language remains a plausible shortcut and prevents a claim of broad semantic generalization.

B.1 Matched-intervention condition–language diagnostic

This diagnostic isolates the case in which satisfaction matching alone is insufficient. For each group, the scene, source trajectory, manipulated object, candidate plan signature, pre-step state evidence, condition predicate, and intervention signature are identical. Only the task instruction and its semantic outcome change. Each group contains (i) a neutral instruction that does not require the violated candidate condition, (ii) an implicit instruction that requires it, and (iii) a mismatch instruction that requires a different condition. The mismatch negative prevents the model from treating any constraint-like sentence as evidence that the candidate condition is critical.

We exclude `target_receptacle` because direct receptacle-string equality solves that category. The six retained categories are sliced/unsliced, clean/dirty, heated/unheated, cooled/uncooled, held-object, and temporal-order. Validation and test paraphrase templates are absent from training. Every final test triplet passes an exact equality audit over all non-text and non-label fields. We remove `context_role`, `object_role`, propagated constraint relations and intervals, inherited constraints, and semantic outcome annotations from the predictor input. The final test contains five groups per category, for 30 triplets and 90 contexts. The text encoder is frozen `all-MiniLM-L6-v2`; a fixed pairwise head receives the prompt embedding, candidate-condition embedding, their absolute difference, and elementwise product. Three seeds are averaged, and the operating threshold is selected on the trajectory-disjoint validation set.

The single complete estimator ablation and all matched-diagnostic outcomes are reported in Table 2 of the main paper. We do not duplicate a partial result table here. Its labels are deterministic structured semantic outcomes attached to matched plan/state contexts, not new physical Controller rollouts; the experiment therefore isolates task–condition alignment but does not establish open-ended language understanding or real-world perception.

Nineteen candidate anchors are excluded because they cannot support three distinct valid interventions. A strict executor and a navigation-and-retry executor are each run on all 172 variants (344 rollouts total) in AI2-THOR 5.0. Their traces differ on 33 paired variants, but their final controller and semantic outcomes are identical. The result therefore provides executor stratification, but not evidence of invariance across VLA policies.

Table 6: Multi-defect stress test on 43 anchors (per executor).

Defects	N	Decision Acc.	Any recall	All recall
0	43	1.000	–	–
1	43	0.860	0.860	0.860
2	43	1.000	1.000	0.512
3	43	1.000	1.000	0.488

Table 7: Combined structured-evidence corruption on the separate fixed 200-context diagnostic set. Arrows show the desirable direction.

Noise	Method	Accuracy ↑	Success Pass ↑	FAR ↓	Abstention ↓
0%	TCCV	0.875	0.863	0.112	0.000
	Hard validator	0.490	0.000	0.000	0.000
	Direct success	0.780	0.598	0.031	0.000
10%	TCCV	0.756	0.810	0.204	0.101
	Hard validator	0.444	0.002	0.000	0.101
	Direct success	0.712	0.559	0.033	0.101
20%	TCCV	0.631	0.749	0.310	0.178
	Hard validator	0.402	0.002	0.000	0.178
	Direct success	0.643	0.533	0.061	0.178
30%	TCCV	0.551	0.690	0.327	0.267
	Hard validator	0.360	0.002	0.000	0.267
	Direct success	0.570	0.465	0.053	0.267

B.2 Structured-evidence corruption audit

Question and scope. This experiment asks whether TCCV degrades gracefully when its upstream structured evidence is imperfect. It is deliberately an interface-level robustness test rather than an end-to-end sensing claim. AI2-THOR metadata is structured simulator state; a deployment would need an upstream provider for the same evidence contract. In the experiments above, metadata is used only to instantiate current object/state evidence after obligations have been extracted from the instruction, plan, action schemas, and goal. It is not presented as learned perception and never exposes the intervention identity, counterfactual label, terminal outcome, future state, or a test statistic during inference.

Frozen protocol. We freeze a separate 200-context structured-evidence diagnostic set, all model weights, and all decision thresholds; no method is retrained on corrupted inputs. For each $p \in \{0, 0.1, 0.2, 0.3\}$ and five seeds, we independently (i) delete object records with probability p, (ii) replace available attributes with an explicit `unknown` value with probability p, and (iii) inject false-positive (false→true) or false-negative (true→false) errors into visibility, reachability, and affordance fields. The combined condition applies object deletion and attribute unknown at rate p and each eligible FP/FN direction at rate $p/2$. Corruption touches only pre-execution evidence. If it removes evidence that was previously available for the candidate obligation, every method abstains; there is no imputation, clean-record lookup, or metadata fallback. Pre-existing unavailable fields retain the diagnostic protocol's decision so that the $p = 0$ row exactly reproduces its clean operating point.

We compare frozen TCCV, a uniform hard validator, and the same-encoder direct success baseline. The hard validator blocks an observed violated candidate, executes only if corrupted evidence masks that violation, and abstains when its relevant evidence is removed. Direct-success probabilities are averaged over the three trained seeds and thresholded at 0.5; TCCV uses the main table's corruption-audit threshold 0.99999. The direct model's clean accuracy here (0.780) belongs to this separate, fixed 200-context corruption set and should not be compared across evaluation units with the matched-intervention ablation. Accuracy counts abstentions as incorrect, Success Pass and FAR retain the main paper's denominators, and Abstention is the fraction of all 200 contexts without an execute/block decision. Table values are means over five corruption seeds; raw per-seed results and standard deviations are included with the released artifacts.

Table 8: TCCV under missing structured evidence, separated by corruption type. Means over five seeds.

Type	Noise	Accuracy	Success Pass	FAR	Abstention
Object deletion	10%	0.834	0.831	0.118	0.043
	20%	0.752	0.775	0.147	0.125
	30%	0.694	0.725	0.161	0.182
Attribute unknown	10%	0.796	0.843	0.182	0.060
	20%	0.763	0.861	0.249	0.088
	30%	0.686	0.786	0.273	0.146

Table 9: TCCV sensitivity to 30% channel-specific Boolean corruption. These flips do not create unknown values, so abstention is zero.

Channel	Error	Accuracy	Success Pass	FAR
Visibility	FP	0.876	0.869	0.116
	FN	0.860	0.896	0.178
Reachability	FP	0.875	0.863	0.112
	FN	0.847	0.894	0.202
Affordance	FP	0.838	0.824	0.147
	FN	0.868	0.882	0.147

Result. The audit gives a qualified rather than uniformly positive answer. With 10% combined corruption, TCCV retains the highest accuracy (0.756 versus 0.712 for direct success and 0.444 for the hard validator) and a substantially higher Success Pass rate. At 20–30% combined corruption, however, direct success slightly exceeds TCCV in overall accuracy, and TCCV's FAR rises from 0.112 to 0.327. Missing evidence is handled conservatively through abstention, but an incorrectly observed Boolean remains more dangerous because it is treated as known. Reachability false negatives produce the largest channel-specific FAR (0.202 at 30%). Thus the current system degrades gracefully under moderate deletion through an explicit unknown path, but not under severe, confidently wrong evidence. This result motivates calibrated evidence confidence and redundant sensing rather than assuming that clean structured-evidence performance transfers unchanged to other evidence providers.

C Detailed LLM Self-Assessment Diagnostic

This standalone supplement documents the protocol, all condition-level outcomes, and the scope of interpretation for the motivating Qwen/AI2-THOR diagnostic. The decisive three-condition comparison is repeated within the full results below, so the supplement does not depend on labels or definitions from the main manuscript.

C.1 Question and Comparison

The diagnostic asks whether an LLM importance judgment can replace execution-derived supervision for condition criticality. Both Qwen variants receive the same natural-language instruction, pre-execution AI2-THOR scene summary, robot profile, and planning context.

- **Ordinary plan/PDDL generation** returns an action plan and action-level preconditions/effects but is not asked to score condition importance.
- **Explicit Qwen self-check** receives the generated plan and a candidate condition list, marks whether each condition should be checked before execution, and assigns an integer importance score from 0 to 100.

Because the explicit variant receives the candidate list, this diagnostic does not treat its condition coverage as evidence of autonomous condition discovery. The substantive comparison is whether its numerical self-assessment agrees with the causal effect measured by execution.

We use AI2-THOR `FloorPlan1` and Qwen with deterministic decoding. The scene contains a mug, microwave, countertop, and alternative pickupable objects. The probe covers three in-

Table 10: Complete diagnostic results. "Std." indicates whether the ordinary plan/PDDL output explicitly mentions the corresponding condition. Y^0 is task success when the condition is false. Absolute error compares the Qwen self-score with paired execution criticality C^*. Each row is a deterministic paired probe with one rollout per condition state.

Task	Condition	Std.	LLM	Y^0	C^*	$\lvert\text{LLM} - C^*\rvert$	Type
Specific pickup	Mug exists	No	100	0	100	0	Critical
Any-object pickup	Mug exists	No	0	1	0	0	Irrelevant
Specific pickup	Mug initially within reach	No	95	1	0	95	Recoverable
Specific pickup	Mug initially visible	No	90	1	0	90	Recoverable
Specific pickup	Robot hand is empty	No	85	0	100	15	Critical
Specific pickup	Another pickupable exists	No	0	1	0	0	Irrelevant
Microwave placement	Microwave exists	No	100	0	100	0	Critical
Microwave placement	Microwave initially within reach	No	95	1	0	95	Recoverable
Microwave placement	Microwave initially visible	No	85	1	0	85	Recoverable
Microwave placement	Microwave initially open	Yes	100	1	0	100	Recoverable

structions: pick up a specified mug, pick up any pickupable object, and place the mug inside the microwave. Qwen generates the high-level plans. In particular, the microwave-placement plan contains `NavigateTo(Mug)`, `PickupObject(Mug)`, `NavigateTo(Microwave)`, `OpenObject(Microwave)`, and `PutObject(Mug,Microwave)`.

For each condition, the Controller is reset and the same high-level plan is executed once with the condition true and once with it false. Navigation is implemented by querying validated interactable poses and teleporting to the selected pose; teleportation is therefore a deterministic low-level navigation primitive rather than a success label. Pickup, opening, and placement use actual `Controller.step` outcomes. Success is determined from the final inventory or parent-receptacle relation, not from an LLM judgment. The condition-level target is

$$C_i^* = 100\left[Y_i^{\text{true}} - Y_i^{\text{false}}\right]_+, \tag{8}$$

where Y is semantic task success. Thus, $C_i^* = 100$ means that falsifying the condition changes success to failure in the matched pair, whereas $C_i^* = 0$ means that success is preserved.

Across the ten conditions, ordinary plan/PDDL generation explicitly covers 16.1% of the broader reference critical-condition set used in the motivating study and mentions only one of the ten conditions in Table 10. The Qwen self-score has a mean absolute error of 48 points against execution criticality, and five of ten conditions have an absolute error of at least 50 points. Error is concentrated on the five recoverable initial conditions, where the mean absolute error is 93 points. In contrast, the mean error is 5 points on the three truly critical conditions and zero on the two irrelevant conditions.

C.2 Why the Microwave Case Is Decisive

In the initial scene, the microwave is closed. Qwen correctly recognizes that `MicrowaveOpen` is required at the moment of placement and generates an opening action. Its self-check nevertheless labels "the microwave is open" as a mandatory pre-execution condition with importance 100. The false-state rollout starts with the microwave closed, executes the generated opening action, and successfully places the mug. The true-state rollout also succeeds. The initial state therefore has $C^* = 0$, even though the corresponding action-time predicate is necessary for `PutObject`. This case isolates the distinction between an intermediate precondition and a plan-level causal requirement.

The paired mug-existence rows provide a complementary control. Removing the mug causes a specified-mug task to fail, producing $C^* = 100$, but the same intervention preserves success for an any-object instruction because another pickupable object is a valid substitute. Qwen assigns 100 and 0, respectively, showing that LLM self-assessment can capture simple semantic relevance. The failure is therefore not that LLM judgments are uniformly poor; it is that they are not reliably grounded in plan dynamics and recoverability.

Table 11: AI2-THOR data used in the experiments.

Quantity	Count
TCCV source trajectories	1,000
Base step-obligation records	8,000
Text-contrast records	2,000
Combined TCCV benchmark records	10,000
Support-constrained E2 train/validation/test contexts	6,556 (5,832/160/564)
E3 repair-evaluation tasks	100
E4 valid-unseen transfer contexts	50

D Dataset Preparation

D.1 Dataset Scale and Splits

Table 11 reports the data used by the main experiments. All records derived from the same source trajectory remain in the same split. Training, validation, and test records are isolated by source trajectory, contrast group, and source path. The TCCV training data uses official ALFRED source trajectories and text-conditioned counterfactual records. E2 uses a separate 564-context score-independent comparison pool, and the transfer and repair studies use disjoint official ALFRED tasks.

The benchmark focuses on ALFRED conditions that can be verified from structured AI2-THOR metadata. The evaluation therefore isolates the criticality layer under structured evidence and does not make an end-to-end deployment claim. The benchmark includes object identity and type, target receptacle, parent-receptacle relations, sliced, dirty/clean, heated, cooled, held-object, and temporal-order conditions. The new text-contrast records keep the same scene, source trajectory, candidate plan, step, object, pre-step state, and condition fixed while changing the instruction-derived goal constraint and the goal-criticality label.

D.2 TCCV training data construction

The TCCV model used in the main experiments is trained on the ALFRED Text-Conditioned Step-TCCV benchmark. The benchmark contains 10,000 step-level planning records derived from 1,000 ALFRED source trajectories and 1,000 source `traj_data.json` paths. The source trajectories are taken from the official ALFRED `json_2.1.0` data and are processed into step-condition records with the same input contract used by TCCV: task text, high-level action, candidate plan signature, step index, pre-step evidence, object and receptacle fields, condition predicate, temporal scope, and constraint propagation fields.

The benchmark has two components. First, it contains 8,000 execution-grounded step-obligation records in the Step-TCCV subset. These records are obtained by replaying factual prefixes and applying atomic counterfactual interventions to one environmental condition at a time. The labels compare the factual execution with the counterfactual outcome and record whether the condition is locally critical for the current step and/or goal-critical for the task. Second, it contains 2,000 text-contrast records organized into 1,000 complete contrast groups. In each group, the scene, source trajectory, candidate plan, current step, high-level action, object binding, pre-step evidence, and condition predicate are held fixed. Only the instruction and the instruction-derived semantic condition are changed. This creates paired examples in which the same plan can be valid for one instruction but invalid for another. Table 12 reports the split sizes and isolation units.

Splits are isolated by source trajectory, contrast group, and source path. The overlap in each of these keys is zero for train–validation, train–test, and validation–test. Thus, no paired instruction contrast group or source trajectory can appear in more than one split. Scene identifiers are reported as part of the distribution audit; multiple official ALFRED trajectories can share the same FloorPlan, so the primary leakage barrier is the trajectory/group/path unit. Scene-transfer behavior is separately evaluated on `valid_unseen` in E4. Table 13 summarizes scene coverage across room types.

The task distribution combines Step-TCCV task families and text-contrast task families. Across all 10,000 records, the main Step-TCCV families are `look_at_obj_in_light` (5,656 records),

Table 12: TCCV benchmark split sizes and isolation units.

Split	Records	Base	Text	Traj.	Groups
Train	7,870	6,106	1,764	761	882
Validation	1,102	958	144	121	72
Test	1,028	936	92	118	46
Total	10,000	8,000	2,000	1,000	1,000

Table 13: Scene and task coverage in the TCCV benchmark.

Split	Scenes	Kitchen	Living	Bedroom	Bathroom
Train	86	20	26	25	15
Validation	39	3	17	14	5
Test	37	7	13	14	3
Total	89	22	27	25	15

`pick_and_place_simple` (1,490), and `pick_and_place_with_movable_recep` (854). The text-contrast families are `sliced_unsliced` (360 records), `target_receptacle` (360), `clean_dirty` (320), `heated_unheated` (320), `cooled_uncooled` (280), `held_object` (180), and `temporal_order` (180). The total goal-criticality labels are balanced: 5,004 non-critical and 4,996 critical records. Local criticality has 5,905 positive and 4,095 negative records. Table 14 reports the high-level plan-length distribution.

The most frequent evaluated high-level actions are `PickupObject` (8,357 records), `ToggleObject` (827), `GotoLocation` (480), `PutObject` (310), and `SliceObject` (26). The released training data are stored in

```
additional materials/
TCCV_v2_training_data_alfred
_text_tccv_v2/
```

The folder contains the full benchmark JSONL files, including `train.jsonl`, `validation.jsonl`, `test.jsonl`, `combined_10000.jsonl`, `text_contrast_2000.jsonl`, and `contrast_groups_1000.jsonl`, together with the leakage and deduplication audit files.

E Experiment-Specific Evaluation Data

E.1 E2 support-constrained, score-independent valid-unseen comparison

The E2 test pool contains all 564 eligible records (282 complete contrast pairs) from official ALFRED `valid_unseen` trajectories. The evaluation protocol was written before training or scoring so that held-out trajectory transfer is not confounded with absent task–condition support. Using seed 20260929, we generate structured transfer pairs from official ALFRED train trajectories. The support key is (`source_task_type`, `condition_family`). Every validation or test cell must retain at least five complete pairs in training. All 19 test cells satisfy this criterion, so the complete 282-pair test pool is retained and the unsupported/OOD file is empty. After source-group isolation, training contains 2,916 pairs from 2,285 trajectories. Validation contains 80 pairs from 80 different ALFRED-train trajectories and matches the 19-cell test distribution by fixed proportional allocation. Test contains 282 pairs from 198 `valid_unseen` trajectories. Train–validation, train–test, and validation–test source overlaps are all zero.

For TCCV, we search thresholds only on the validation set, maximizing successful-plan pass rate subject to FAR ≤ 0.10, with ties resolved by accuracy and then lower FAR. The frozen risk threshold is 0.0036562 (execute below threshold), at which TCCV obtains perfect validation decisions.

DLJ is a real Qwen-Plus call at temperature zero, not a keyword heuristic. It receives the instruction, complete candidate high-level plan, current step, robot capabilities, and structured scene evidence, and

Table 14: High-level plan-length distribution in the TCCV benchmark.

Length	0	3	4	6	7
Records	1,000	196	7,769	41	678
Length	8	10	11	12	13
Records	12	52	30	9	213

Table 15: Condition-level error audit for the two E2 external-baseline adaptations.

Condition	VerifyLLM		VeriGraph	
	Acc.	FAR	Acc.	FAR
clean/dirty	0.769	0.000	1.000	0.000
cooled/uncooled	0.784	0.000	0.986	0.000
heated/unheated	0.833	0.000	1.000	0.000
held object	0.409	0.455	0.455	1.000
sliced/unsliced	0.632	0.000	0.921	0.000
target receptacle	0.736	0.000	0.882	0.019
temporal order	0.750	0.000	0.944	0.000

returns a JSON EXECUTE/BLOCK decision. Its input explicitly excludes labels, condition-family and condition-predicate fields, TCCV scores, and contrast IDs. All 564 test responses parse successfully. The prompt, per-record predictions, token counts, and cache are released.

VerifyLLM and VeriGraph adaptations. Before scoring, we freeze a common-input protocol for both external baselines. The VerifyLLM adaptation uses Qwen-Plus at temperature zero to produce a grounded LTL-style formula, atomic propositions, and five-action-window findings for the complete candidate plan. We convert its decision and confidence to a risk score and choose the operating threshold only on the same 80-pair validation set under FAR ≤ 0.10. The selected threshold is 0.970; validation accuracy is 0.738 and validation FAR is 0.013. On E2, VerifyLLM obtains 0.734 accuracy, 0.486 successful-plan pass rate, 0.982 failure-blocking rate, 0.018 FAR, and 0.514 FRR. Its risk ranking accuracy is 0.496, showing that the improved context accuracy primarily comes from conservative blocking rather than reliable pairwise separation.

The VeriGraph adaptation asks Qwen-Plus only to construct a goal graph from the raw instruction and plan object vocabulary. A deterministic checker then replays the complete high-level plan and tests spatial, attribute, temporal, and held-at-action constraints. It obtains 0.902 accuracy, 0.855 pass rate, 0.950 failure-blocking rate, 0.050 FAR, 0.145 FRR, and 0.809 complete-pair accuracy. All 724 validation/test outputs per baseline parse successfully, and the goal-graph parser uses no fallback. The prompts exclude labels, condition-family and condition-predicate fields, extracted text constraints, contrast IDs, and TCCV scores.

The held-object row isolates the main graph limitation: the parsed terminal goal graph fails to preserve "hold object x at ToggleObject" reliably, so VeriGraph passes all eleven failed held-object plans. This is a limitation of the adapted goal parser/terminal graph representation, not evidence that the source VeriGraph method cannot be extended with explicit action-time state. The supplementary files are stored under

```
additional materials/
iclr_revision_20260925/
coverage_corrected_e2/
```

The main files are:

- `tccv/benchmark/train.jsonl`, `validation.jsonl`, and `test.jsonl`: the frozen trajectory-isolated split.
- `audit/split_manifest_pre_training.json`: support counts, cell-matching targets, and disjointness checks written before training.

Table 16: E3 official ALFRED repair-evaluation length distribution.

Length bucket	Total	Success	Failure
Short	34	17	17
Medium	34	17	17
Long	32	16	16

Table 17: Injected failures in the E3 repair-evaluation set.

Mutation	Count
Remove final `PutObject` fallback	20
Late `CleanObject`	9
Late `CoolObject`	7
Late `HeatObject`	3
Late `SliceObject`	11

- `audit/calibrated_thresholds.json`: selection objective, thresholds, and calibration metrics.
- `results/e2_coverage_corrected.csv` and `dlj/e2_dlj_result.csv`: the main numeric results.
- `dlj/dlj_predictions.jsonl` and `dlj/dlj_summary.json`: direct-Qwen predictions, parse rate, and token usage.
- `external_baselines/`: frozen prompts, cached Qwen outputs, per-record VerifyLLM/VeriGraph predictions, condition breakdowns, and the pre-scoring protocol manifest.

E.2 E3 Repair Evaluation Dataset and Prompts

For E3, we construct a separate repair evaluation set from official ALFRED `json_2.1.0` trajectories. The set contains 100 complete high-level task plans. We keep 50 official plans unchanged and label them as successful. We randomly sample another 50 disjoint official plans and inject subtle failures. The failure edits preserve the instruction and most of the plan, but break one task-critical condition. This setting tests whether repair benefits from knowing which condition is relevant to the current instruction. Tables 16 and 17 report the length buckets and injected failure types.

The released files are stored in

```
additional materials/
e3_official_alfred_qwen_repair
_taskcondition_ablation/
```

The main dataset file is `data/official_alfred_taskcondition_100.jsonl`. The Qwen responses are in `traces/qwen_plain_vs_tccv_guided_traces.jsonl`. The result tables are in `results/e3_taskcondition_qwen_results.csv` and `results/e3_taskcondition_qwen_results.tex`. After repair, each accepted plan is executed with the AI2-THOR Controller from the restored initial state. Final task completion is determined by the semantic goal evaluator, while repair success and success preservation are computed separately over the 50 injected-failure and 50 initially successful tasks. This execution protocol yields the E3 results reported in Table 4 of the main paper.

Prompt interfaces. The plain LLM repair baseline receives only the instruction, the candidate plan, and the legal action set. For example, for the mutated task *Put a heated piece of apple in the sink*, the plain prompt is:

```
{
  "instruction":
    "Put a heated piece of apple in the sink.",
  "candidate_plan": [
    {"action": "SliceObject",
     "objectId": "Apple|..."},
    {"action": "PutObject",
     "objectId": "AppleSliced|...",
     "receptacleObjectId": "Sink|..."},
    {"action": "HeatObject",
     "objectId": "Microwave|..."}
  ],
  "legal_actions": [
    "GotoLocation", "PickupObject",
    "PutObject", "SliceObject",
    "HeatObject", "CoolObject",
    "CleanObject", "..."
  ],
  "output_schema": {
    "repair_needed": "boolean",
    "repaired_plan": "list of actions",
    "reason": "brief"
  }
}
```

The VeriGraph row does not use an LLM prompt in this experiment. It receives a symbolic graph-repair input. The input contains the candidate plan, the instruction-derived goal graph, and the graph violation:

```
{
  "initial_graph": {
    "objects": [
      "Apple", "Sink", "Microwave"
    ],
    "...": "..."
  },
  "goal_graph": {
    "subject": "Apple",
    "required_state": {"heated": true},
    "target_receptacle": "Sink"
  },
  "candidate_plan": [
    "SliceObject", "PutObject", "HeatObject"
  ],
  "violation":
    "HeatObject occurs after final PutObject"
}
```

The TCCV-guided Qwen repair prompt uses the same LLM backend as the plain LLM baseline, but adds the task-conditioned violation inferred by TCCV:

```
{
  "instruction":
    "Put a heated piece of apple in the sink.",
  "candidate_plan": [
    "SliceObject", "PutObject", "HeatObject"
  ],
  "goal_constraints": {
    "required_actions": [
      "HeatObject", "PutObject"
    ],
    "target_receptacle_type": "Sink",
    "subject_type": "Apple"
  },
  "task_conditioned_violation_hint": {
    "missing_or_misplaced_action":
      "HeatObject",
    "expected_relation": "before_final_put",
    "why_task_critical":
      "heated state is required at final goal"
  }
}
```

This information asymmetry is deliberate. The plain LLM must infer the hidden condition–goal relation from the plan text. TCCV supplies the relation explicitly. Under this protocol, plain Qwen repair fixes 70.0% of injected failures and preserves 88.0% of already successful plans. TCCV-guided Qwen fixes all injected failures and preserves 96.0% of already successful plans.

Table 18: Composition of the E4 `valid_unseen` generalization set.

Quantity	Count
Planning contexts	50
Complete text-contrast pairs	25
Successful contexts	25
Failed contexts	25
Source trajectories	25
Candidate plan signatures	21

Qualitative AI2-THOR Repair Videos We also provide five qualitative before/after repair videos. Unlike the repair evaluation above, these videos are rendered from real AI2-THOR 2.1 Controller replay of official ALFRED low-level actions. Each video shows a side-by-side comparison: the left panel removes the task-critical action before repair, and the right panel restores it after TCCV-guided repair. The videos use a fixed-seed random sample from the longest available official ALFRED task bucket. All five have 20 high-level actions. Their low-level action counts are 234, 214, 197, 182, and 128. In these examples, the removed and restored task-critical action is `SliceObject`. The files are stored in

```
additional materials/
tccv_repair_ai2thor_videos/
```

The folder contains the five MP4 files, preview frames, a manifest, and a short summary file. These videos are intended as qualitative supplementary material, not as additional quantitative Controller-rollout results.

E.3 E4 GENERALIZATION TO VALID-UNSEEN TRAJECTORIES

E4 asks whether TCCV generalizes beyond the ALFRED training distribution to additional `valid_unseen` trajectories and scenes. We use a second score-independent sample from the frozen E2 test split. After removing the 25 E1 source trajectories, the remaining source list is shuffled with seed 20260927 and one complete pair is selected from each of the first 25 sources. Thus, E4 is source-disjoint from training and validation, and its membership is unaffected by any method's prediction. The evaluation remains within the ALFRED/AI2-THOR domain and within task–condition families supported by training, so it tests in-domain distributional generalization rather than cross-platform transfer. These files are stored in

```
additional materials/
iclr_revision_20260925/
coverage_corrected_e2/
```

The set contains 25 complete text-contrast pairs, yielding 50 planning contexts: 25 successful contexts and 25 failed contexts. Each pair keeps the same official ALFRED source trajectory, scene, candidate high-level plan, current step, object binding, and pre-step evidence. The success record uses an instruction satisfied by the candidate plan. The paired failure record changes only the task instruction and its derived condition, such as requiring a cooled object, a clean object, a held object at the relevant step, or a different target receptacle. The candidate plan is therefore held fixed while the semantic goal changes. This construction tests whether the task–condition relation learned from training trajectories transfers to unseen sources, scenes, and a different mixture of supported conditions while recognizing that the same plan may be safe for one instruction but invalid for another. Tables 18 and 19 report the subset composition and plan-length distribution.

The condition distribution is 34 `target_receptacle`, 6 `sliced_unsliced`, 4 `cooled_uncooled`, 4 `clean_dirty`, and 2 `held_object` contexts. The released files are:

- `data/e4_independent_rows.jsonl`: the 50 frozen contexts.
- `results/e4_results.csv` and `dlj/e4_dlj_result.csv`: the E4 comparison results.

Table 19: High-level plan-length distribution for the E4 `valid_unseen` generalization set. Length is the number of ALFRED `plan.high_pddl` actions in the official source trajectory.

Plan length	Contexts
4	2
5	6
6	6
7	24
8	6
9	2
12	2
13	2

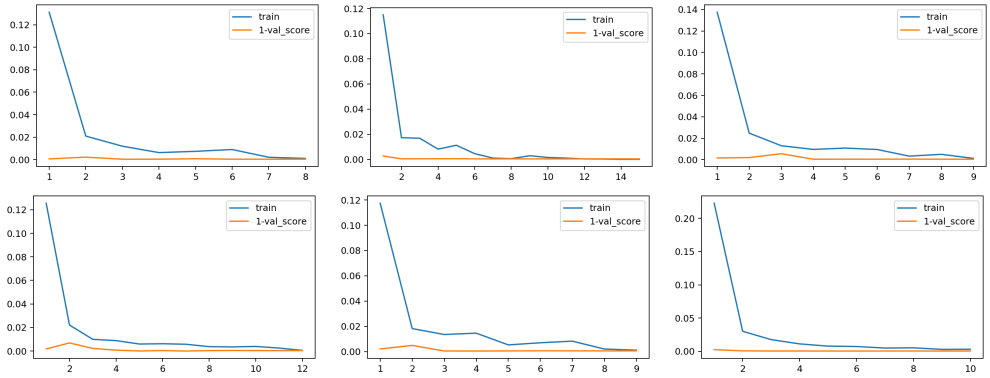


Figure 3: Training loss curves for the six TCCV hyperparameter configurations. Each panel corresponds to one configuration, C01–C06, from the same training split and seed used in the hyperparameter sweep.

- `audit/split_manifest_pre_training.json`: source counts, random seed, score-independent protocol, and split disjointness audit.

F Additional Experiments

F.1 TCCV Convergence Analysis

Figure 3 shows the training loss curves for the six hyperparameter configurations used in the TCCV search. The curves are generated from the saved training logs of the reported TCCV run, using the same benchmark described above. Across configurations, the losses decrease smoothly within the early-stopping budget, and the selected configuration is not the result of an obviously unstable training trajectory.

F.2 Training Parameters and Metric Details

Table 20 lists the six hyperparameter configurations used in the TCCV search. All runs use AdamW, weight decay 0.01, gradient clipping at 1.0, batch size 32, no DataLoader workers, and early stopping with patience 5 after at least 8 epochs. The instruction encoder is frozen for the first five epochs. The validation score is the weighted model selection objective defined over text-contrast pair accuracy, goal AUPRC, semantic-failure recall, and local AUPRC.

Table 21 reports the seed-7 search summary. C04 has the highest validation score and is therefore the strongest single-seed configuration in this search. C01 and C04 were subsequently trained with additional seeds.

The training curves in Figure 3 show rapid convergence. Most configurations reduce training loss sharply within the first two epochs. The validation objective is already near saturation after the early

Table 20: TCCV hyperparameter search configurations.

Config	Hidden	Layers	Heads	Dropout	Head LR	Text LR	Pair weight
C01	128	2	4	0.10	3×10^{-4}	1×10^{-5}	0.5
C02	192	2	4	0.10	3×10^{-4}	1×10^{-5}	0.5
C03	192	3	4	0.15	2×10^{-4}	1×10^{-5}	0.5
C04	256	2	8	0.15	2×10^{-4}	1×10^{-5}	0.5
C05	192	2	4	0.10	3×10^{-4}	5×10^{-6}	0.8
C06	192	2	4	0.20	1×10^{-4}	1×10^{-5}	0.2

Table 21: Seed-7 TCCV search summary. Higher validation scores are better.

Config	Best epoch	Train loss ↓	Validation score ↑	Text grad norm
C01	3	0.01195	0.99963	0.00
C02	10	0.00156	0.99959	156.33
C03	4	0.00956	0.99960	0.00
C04	7	0.00579	**0.99999**	567.39
C05	4	0.01458	0.99959	0.00
C06	5	0.00782	0.99959	0.00

epochs, which explains why early stopping selects short runs for several settings. C04 gives the best validation score while maintaining a small training loss. This supports the use of C04 as the main configuration. The complete training artifacts are stored in

`additional materials/`
`tccv_v2_training_loss_curves/`

Unified decision metrics. For binary pre-execution verification, rows correspond to the true execution outcome $Y \in \{1, 0\}$ and columns correspond to the final EXECUTE/NOTEXECUTE decision. We report

$$\mathrm{FAR} = \frac{FP}{FP + TN},$$
$$\mathrm{FRR} = \frac{FN}{TP + FN}.$$

Selective Risk is reported as "–" when no plan is executed. For repair experiments, Final Task Completion is the fraction of episodes that reach the semantic task goal after repair.

F.3 TCCV OPERATING THRESHOLD CALIBRATION

This section documents the operating-threshold calibration used for the support-constrained pre-execution comparison. No E2 test label is used to select a threshold.

Calibration split. We use an 80-pair (160-context) ALFRED-train validation set whose 19 task-type–condition cell counts proportionally match the complete `valid_unseen` E2 test distribution. Every `source_path` belongs to exactly one of train, validation, or test. The split seed is 20260929, and the manifest is persisted before model training. Validation contains 80 source trajectories; E2 test contains 198, with zero overlap. Each evaluated cell has at least five training pairs after validation sources are removed.

Score and decision rule. TCCV outputs a goal-criticality score p_{goal} by applying a sigmoid to the goal-criticality logit. Larger values indicate higher predicted risk. For a threshold τ, the deployment decision is

$$d = \begin{cases} \textsc{Execute}, & p_{\mathrm{goal}} < \tau, \\ \textsc{Block}, & p_{\mathrm{goal}} \geq \tau. \end{cases}$$

Thus a lower threshold gives a more conservative verifier.

Table 22: Operating points selected on the distribution-matched, trajectory-disjoint ALFRED-train validation set.

Split	Acc.	Succ. Pass	Fail Block	FAR	FRR	τ
TCCV	1.000	1.000	1.000	0.000	0.000	0.0037

Selection criterion. We choose τ on the validation set under a safety constraint. The constraint is FAR ≤ 0.10, where FAR is the fraction of failed plans that are incorrectly executed. Among thresholds satisfying this constraint, we select the one with the highest success-pass rate. Ties are broken by higher decision accuracy and lower FAR. Table 22 reports the selected calibration point. This rule selects

$$\tau^{\star}_{\mathrm{TCCV}} = 0.0036562.$$

Frozen-threshold test check. After selection, the threshold is frozen and evaluated once on the 564 E2 test contexts. TCCV obtains 0.993 accuracy, 0.986 success-pass rate, 1.000 failure-blocking rate, zero FAR, 0.014 FRR, 1.000 pair accuracy, and 1.000 AUROC. TCCV errors are four conservative false rejections; no test-set threshold adjustment is applied.

Released calibration artifacts. The calibration artifacts are included under `additional materials/iclr_revision_20260925/coverage_corrected_e2/`. The folder contains the pre-training split manifest, frozen benchmark rows, selected thresholds, three-seed checkpoints, scored records, and frozen test decisions.