NEURO-SYMBOLIC ANCHORED GOVERNANCE FOR ALLEVIATING ACTION-LEVEL HALLUCINATIONS IN LARGE LANGUAGE MODEL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM) agents frequently suffer from action-level hallucinations. Unlike traditional hallucinations that appear as factual inconsistencies in text generation, action-level hallucinations involve proposing actions that are semantically plausible but violate environment-specific feasibility or procedural constraints. Probabilistic generation alone does not ensure action legality, and violation risks can compound over long action sequences. Explicit checks identify violations, but agents must also determine when to intervene and how to recover. To address this, we propose Neuro-Symbolic Anchored Governance (NSAG), a framework that externalizes rule enforcement from the LLM to a symbolic execution interface. NSAG selects task-relevant rules and learns to calibrate intervention using symbolic violation signals. To resume execution when an action is blocked, we introduce MinFix, a search-based repair operator that seeks a minimal executable sequence satisfying the action's missing preconditions while preserving high-level intent. We develop and release Town-Benchmark with 1,000 task instances across five domains and evaluate NSAG on Town-Benchmark and ScienceWorld. NSAG outperforms state-of-the-art baselines in hallucination mitigation and task performance, with consistent gains across LLM backbones and low inference overhead. The reductions are empirical, and the calibration is fitted on the same five domains it is evaluated on. The open environment and code are available at `https://anonymous.4open.science/r/NSAG/`.

1 INTRODUCTION

Large Language Model (LLM) agents act on executable environments, where every proposed step either changes state or is refused (Huang et al., 2024a; Li et al., 2024). Deployments now range from robot control (Kim et al., 2024; Kannan et al., 2024; Yang et al., 2024c) to open-world simulation and software engineering, where a plan has to survive contact with the system that runs it (Huang et al., 2024b; Wei et al., 2024; Yang et al., 2024a). A textual assistant can answer without committing to a state, and an agent cannot, since every proposed action must hold in the current state before the environment accepts it and lets the episode continue, one step at a time.

Even when the plan is coherent, an LLM agent can propose an action the current state does not permit. We define action-level hallucination as the generation of actions that are semantically plausible and align with the high-level goal, but violate the specific feasibility or procedural constraints of the current environment (Lin et al., 2025; Plaat et al., 2025; Sumers et al., 2023). For instance, in a household navigation task, an agent might propose to "walk through the wall" to reach a target room more quickly, or attempt to "pick up an apple" when its inventory is already full (Chakraborty et al., 2025; Guo et al., 2024; Yang et al., 2024b; Pan et al., 2025). The failure is not linguistic. The agent's internal world model and the external environment disagree about what the current state allows, and the execution failure that follows shapes every later step of the episode.

Current approaches include instruction tuning, in-context reasoning (e.g., Chain-of-Thought), and post-hoc filtering (Ray, 2025; Huang et al., 2024a; Zhou et al., 2024). However, next-token prediction alone does not ensure action legality in the current state (Yehudai et al., 2025). Even small per-step violation risks can compound over long action sequences, causing the probability of an

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.

entirely valid sequence to decay exponentially when the conditional violation probability remains bounded away from zero. This motivates explicit execution-time constraint enforcement for reliable LLM agents. Existing runtime-enforcement methods already provide external checks (Yang et al., 2024c; Wang et al., 2026a). Beyond detecting violations, agents must select task-relevant rules, decide when to intervene, and recover without abandoning useful plans.

To address these challenges, we propose Neuro-Symbolic Anchored Governance (NSAG), a framework that decouples semantic planning from rule enforcement. Instead of forcing the LLM to memorize constraints, NSAG externalizes governance to a symbolic execution interface. To connect static symbolic rules with dynamic, unstructured task contexts, we introduce a differentiable neuro-symbolic mechanism. This trainable module pairs a Rule Trigger Predictor that selects task-relevant registered rules with a Differentiable Symbolic Guard that calibrates how their violation signals trigger intervention (Bhuyan et al., 2024; Nawaz et al., 2025). Symbolic predicates identify violated conditions; learning turns these signals into task-dependent intervention decisions. Furthermore, a blocked action need not invalidate the plan: freeing inventory space may enable the intended pickup while retaining subsequent steps. We therefore introduce MinFix, a search-based repair operator. MinFix seeks a minimal executable sequence satisfying the blocked action's missing preconditions while retaining the agent's intended course of action. These preconditions explain the violation and define the repair target that the search then has to reach. A blocked step whose missing preconditions no legal sequence can supply inside the search budget ends the episode, and the experiments report how often the search runs out of budget before a bridge is found.

We evaluate NSAG on our newly released Town-Benchmark, a visual environment designed to stress-test agents across diverse scenarios involving navigation, resource management, and social interaction. Town-Benchmark enforces explicit physical and logical constraints across 1,000 task instances in five domains. The comparison in Table 1 reports task completion rates of 99% for NSAG and 72% for direct generation. We examine hallucinations, task completion and computational cost together to assess whether intervention helps agents complete their tasks.

Our contributions are summarized as follows:

- Conceptual Insight: We identify the limitations of probabilistic constraint enforcement in current agent paradigms and demonstrate that externalizing governance provides a more robust and predictable framework for reliability in deployed agent systems.
- Methodological Framework: NSAG integrates task-conditioned rule selection, learned intervention calibration, and MinFix to detect action-level hallucinations and repair missing preconditions while preserving the agent's high-level intent and course of action.
- Benchmark Release: We develop and release Town-Benchmark, with explicit physical and logical constraints and controllable task states for diagnosing action-level hallucinations.
- Empirical Performance: NSAG outperforms state-of-the-art baselines in hallucination mitigation and task performance on Town-Benchmark and ScienceWorld, with consistent gains across LLM backbones and low inference overhead; ablations isolate learning and repair.

2 Related Work

Action-level hallucination is addressed either inside the model or at the point of execution. NSAG belongs to the second route and changes what that route does with a step it blocks.

2.1 Action-Level Hallucination in LLM Agents

Hallucinations in autonomous systems also surface as execution failures that follow from a constraint violation, from an invalid parameter call to a breached long-horizon safety protocol (Yu et al., 2025; Zhao et al., 2024). Prevalent mitigation strategies work inside the model, either through inference-time reasoning, where chain-of-thought prompting (Wei et al., 2022) and self-refinement (Madaan et al., 2023; Shinn et al., 2023) induce iterative self-critique, or through external grounding, where the output is conditioned on retrieved documentation (Salama et al., 2025; Kagaya et al., 2024). These methods improve semantic coherence, and the guardrail stays inside the model's probabilistic substrate, so a dynamic feasibility constraint such as a resource limit is not enforced on the step that breaks it. NSAG adds a verification layer that separates rule enforcement from stochastic generation

and needs no fine-tuning of the LLM backbone. A blocked step is repaired by a symbolic search over the environment's own transition model, so the generator is never asked to correct itself at the same step, and an episode that hits a blocked step continues from the repaired state.

2.2 Runtime Enforcement for LLM Agents

A separate line of work enforces constraints on an agent while it runs. AgentSpec (Wang et al., 2026a) is a domain-specific language embedded in Python in which a rule is a trigger, a predicate and an enforcement mechanism, and it was evaluated on code execution, embodied agents and driving, where it is reported to prevent unsafe execution in over 90% of code-agent cases. Its enforcement acts on the current step, and it supports predefined corrective actions and LLM self-examination. VeriGuard (Miculicich et al., 2025) splits the problem differently, with an offline stage that synthesizes and verifies a behavioural policy and an online monitor that validates each proposed action against it, so its assurance is stated relative to that policy. ProbGuard (Wang et al., 2026b) learns a discrete-time Markov chain over abstracted traces and monitors the probability that the run stays safe, so it intervenes before a violation occurs, at the price of an abstraction whose accuracy is certified by a sample-complexity argument under its own sampling assumptions.

NSAG differs from all three in what it weighs and in what it does with a step it blocks. AgentSpec applies the enforcement a rule declares, VeriGuard rejects a step that the verified policy forbids, and ProbGuard intervenes on a predicted risk. The calibration of NSAG is conditioned on the task category, so it learns how much each registered rule matters for the category in hand and where the violation boundary sits. Its repair stage then searches the symbolic state graph for a legal sequence that satisfies the preconditions the blocked action is missing, so the response is a plan segment that lets the model's own proposal continue. The experiments implement AgentSpec's rules for the ten registered constraints and report the comparison, and Appendix E reports how much of the completion rate the repair stage recovers when one of the registered rules is wrong.

3 Preliminaries

Environment. We model the environment as a fully observable deterministic transition system $\mathcal{E} = (\mathcal{S}, \mathcal{U}, \mathcal{T}, \mathcal{R})$. A state $s = (\ell, \mathbf{o}, \mathbf{z}) \in \mathcal{S}$ holds a discrete location $\ell \in \mathcal{L}$, discrete object states $\mathbf{o} \in \mathcal{O}$ and the remaining task-relevant quantities $\mathbf{z} \in \mathbb{R}^d$, which carry the clock, resource levels and progress flags that a rule may read. An action $u \in \mathcal{U}$ is executable when its precondition holds, so $\mathcal{U}_{\text{valid}}(s) = \{u \in \mathcal{U} : \mathrm{Pre}(s, u) = 1\}$ collects the executable ones, and $\mathcal{T} : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$ returns the successor of such an action, so a refused action is never executed and spends one of the protocol's retries. The interface is stated over $\mathcal{S} \times \mathcal{U}$, so an external environment fits it once its rules are registered. Appendix B.5 states the definitions and the retry rule in full.

Rules. A rule is a predicate $r : \mathcal{S} \times \mathcal{U} \to \{0, 1\}$. Splitting the registry into hard constraints $\mathcal{R}_{\text{hard}}$ and soft constraints $\mathcal{R}_{\text{soft}}$, collective legality and compliance are

$$\mathrm{Legal}(s, u) \triangleq \prod_{r \in \mathcal{R}_{\text{hard}}} r(s, u), \qquad \mathrm{Comply}(s, u) \triangleq \prod_{r \in \mathcal{R}} r(s, u), \tag{1}$$

so each evaluates to 1 exactly when its constraints hold simultaneously. We register the hard rules so that $\mathrm{Legal}(s, u) = 1$ and $u \in \mathcal{U}_{\text{valid}}(s)$ agree on the actions the registry covers, and the repair search requires both, while a soft rule that fails leaves its action executable.

Goal and hallucination. A task is an initial state paired with a goal set $(s_0, \mathcal{G})$, and a trajectory $\mathbf{u}_{1:K'}$ succeeds when $\mathrm{Succ}(\mathbf{u}_{1:K'}) \triangleq \mathbb{I}(\exists t \in \{0, \dots, K'\} : s_t \in \mathcal{G})$. The horizon K' counts every executed action, including the ones a repair inserts. For each realised step the hallucination indicators are

$$\begin{aligned} h_t^{\text{hard}} &\triangleq \mathbb{I}\big(\mathrm{Legal}(s_{t-1}, u_t) = 0\big), \\ h_t^{\text{soft}} &\triangleq \mathbb{I}\big(\mathrm{Comply}(s_{t-1}, u_t) = 0 \wedge h_t^{\text{hard}} = 0\big), \\ h_t &\triangleq \mathbb{I}\big(\mathrm{Comply}(s_{t-1}, u_t) = 0\big) \equiv h_t^{\text{hard}} + h_t^{\text{soft}}, \end{aligned} \tag{2}$$

where the first captures illegal actions and the second preference misalignments among legal ones, and both are read against the state that precedes the action. A controller $\mathcal{C}(s_0, x, \mathbf{u}_{1:K})$ maps a proposal to a rectified sequence $\mathbf{u}^\star_{1:K'}$, where x is the textual task context of the methodology. Its

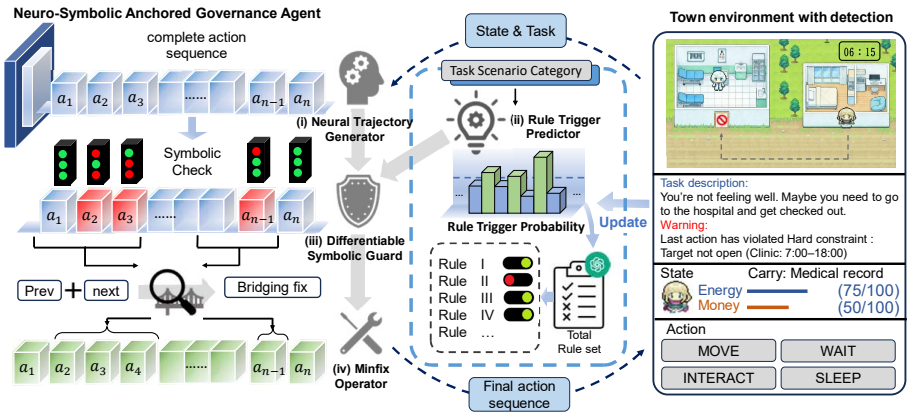


Figure 1: Overview of the neuro-symbolic governance framework. Its four stages are the neural trajectory generator, the rule trigger predictor, the differentiable symbolic guard and the MinFix operator, which bridges a blocked step. The figure writes a step a_i the text calls u_t.

design target is a reduction of the expected number of violations, $\mathbb{E}[\sum_t h_t]$ over the task distribution, at a completion probability of at least $\rho \in [0, 1]$, subject to the latency a deployed system can absorb. The controller is not built by solving that problem, and the experiments report the two quantities separately, since no single number summarises both of them.

4 Methodology

NSAG turns an untrusted proposal into a trajectory whose violations have been located and repaired. Given the textual task context x and the initial state s_0, a frozen backbone samples a candidate sequence $\mathbf{u}_{1:K} \sim P_\phi(\cdot \mid x)$, the guard converts the symbolic violation status of each proposed step into a calibrated intervention decision, and MinFix repairs the steps that decision blocks. Figure 1 shows the four stages, and Appendix B holds the derivations, the training objective, the pseudocode and the catalogue of registered constraints that the two stages share.

4.1 Neural Trajectory Generator

The generator supplies the semantic content that a fixed rule set cannot enumerate. A linearization function lin maps the structured state into text, and the context concatenates the system instruction, k demonstration trajectories, the linearized initial state and the goal, which keeps the interface independent of the environment's internal representation. Decoding is autoregressive over the frozen backbone that every configuration shares, and a parser segments the decoded tokens into actions. The backbone draws each token from its own distribution, which carries no instance of the transition function, so a proposal assigns mass to actions the environment refuses. That mass accumulates across the horizon because the per-step conditional probability of a refused action stays bounded away from zero, which is the gap the remaining stages close. Appendix B.1 gives the context construction and Appendix B.2 the distributional mismatch in full.

4.2 Differentiable Symbolic Guard

Checking every rule at every step costs $\mathcal{O}(K|\mathcal{R}|)$ predicate evaluations, and the weight a rule deserves varies with the category, so the guard learns which rules to weigh and how strongly. The Rule Trigger Predictor estimates, from a learned embedding of the category alone, the probability that a rule constrains it,

$$\mathbf{p}(c) = \sigma\big(\mathrm{MLP}_\psi(\mathbf{e}_c)\big) = [p_1, \ldots, p_M]^\top, \qquad \mathbf{p}(c) \in (0, 1)^M, \tag{3}$$

where $M = |\mathcal{R}|$ and σ applies elementwise, so the estimate is conditioned on the task category and reads no per-step context. A threshold turns the estimate into the active set $\mathcal{R}_{\mathrm{active}}(c) = \{r_j : p_j > \tau_{\mathrm{prune}}\}$, which shrinks the space the guard scans. Appendix B.7 gives the labels and the threshold. The guard grounds the proposal with a symbolic rollout from the state that precedes each action,

$s_t = \mathcal{T}(s_{t-1}, u_t)$, and records the exact status of every active rule,

$$v_{t,j} = \phi_{\text{check}}(r_j, s_{t-1}, u_t) = \mathbb{I}\big(r_j(s_{t-1}, u_t) = 0\big), \qquad r_j \in \mathcal{R}_{\text{active}}(c). \tag{4}$$

A rule that does not apply to the current action carries its applicability condition as an implication, so it returns 1 and stays out of the violation set. A binary status carries no severity, and rules differ in how much a task depends on them, so the guard scores each step with the violation energy

$$E_t \triangleq E(u_t, s_{t-1} \mid c, \Theta) = \sum_{j:\, r_j \in \mathcal{R}_{\text{active}}(c)} \underbrace{g(p_j; w_j)}_{\text{softplus-gated importance}} \cdot v_{t,j}, \tag{5}$$

which sums over the active set alone and keeps the weighting non-negative and monotone,

$$g(p_j; w_j) = \text{softplus}(w_j) \cdot p_j = \log(1 + e^{w_j}) \cdot p_j. \tag{6}$$

The prior p_j scales rule j's contribution by its estimated relevance, and the learnable sensitivity w_j raises the energy of rules that the policy keeps breaking, so the boundary adapts to a category without a per-rule threshold. Detection follows the same calibration,

$$q_t = \sigma\left(\frac{E_t - \gamma}{\tau_{\text{temp}}}\right), \qquad \tau_{\text{temp}} > 0, \tag{7}$$

where γ is a learnable tolerance and τ_{temp} a temperature that sharpens the boundary. With a positive temperature, q_t exceeds $1/2$ exactly when $E_t > \gamma$, so the temperature shapes only the training signal. The calibration learns $\Theta = \{w_1, \ldots, w_M, \gamma, \tau_{\text{temp}}, \psi, \mathbf{E}\}$. Inference intervenes on $b_t = \mathbb{I}(q_t > 1/2)$ and training samples $b_t \sim \text{Bernoulli}(q_t)$, so the loss differentiates the continuous weighting and probability computation that follow the rule checks. A step with $b_t = 1$ names (s_{t-1}, u_t) as the window MinFix repairs, and a step with $h_t = 1$ and $b_t = 0$ is a violation the calibration missed, which the reported violation rate still counts.

4.3 MinFix Operator

A localized violation leaves a gap between the state reached and the preconditions the intent requires, and MinFix closes it while keeping the intent. For the formula set $\mathcal{P}(u)$ the registry attaches to an action, the precondition test and the unsatisfied part are

$$\text{Pre}(s, u) = \mathbb{I}\Big\{s \models \bigwedge_{\varphi \in \mathcal{P}(u)} \varphi\Big\}, \qquad \mathcal{P}_{\text{miss}}(s, u) = \{\varphi \in \mathcal{P}(u) : s \not\models \varphi\}, \tag{8}$$

and MinFix looks for the shortest legal sequence that supplies them,

$$\Delta_t^\star \in \arg\min_{\Delta \in \mathcal{U}^*} |\Delta| \quad \text{s.t.} \quad \text{Pre}(q_{|\Delta|}, u_t) = 1, \quad \text{Legal}(q_{i-1}, \Delta_i) = 1 \;\; (i = 1, \ldots, |\Delta|), \tag{9}$$

where $q_0 = s_{t-1}$, $q_i = \mathcal{T}(q_{i-1}, \Delta_i)$ and Δ_i is the i-th bridge action. The target requires every precondition to hold at the end of the bridge, so a repair cannot trade one satisfied condition for another. The search is a bounded A* over the symbolic state graph with cost $f_{\text{search}}(s) = g_{\text{search}}(s) + h_{\text{search}}(s)$, where g_{search} counts the actions already taken and h_{search} the preconditions still unsatisfied. It returns a legal bridge with a completion flag, and $\perp$ when the reachable space holds no bridge, so the two stay distinct. The shortest bridge is the target, and the solver's return is what we report. Appendix B.6 gives the heuristic, the limits and the cost analysis. Splicing the bridge into the proposal and replaying the remainder keeps the model's own steps,

$$\mathbf{u}^\star = \bigoplus_{t=1}^{K} \text{Patch}(u_t, b_t), \qquad \text{Patch}(u_t, b_t) = \begin{cases} u_t & \text{if } b_t = 0, \\ \Delta_t \parallel \langle u_t \rangle & \text{if } b_t = 1 \text{ and } \Delta_t \neq \perp, \\ \texttt{STOP} & \text{if } b_t = 1 \text{ and } \Delta_t = \perp, \end{cases} \tag{10}$$

where $\Delta_t = \langle\rangle$ when the blocked action was already executable, which leaves the action in place. The blocked action is executed after the bridge, and a repair with no bridge ends the episode, so the sequence contains no step the environment refuses. Because an inserted bridge changes resources, positions and the clock, the suffix is replayed from the updated state, and the original rollout supplies only the steps that were not blocked. Preserving the intent has an operational meaning. The bridge keeps the blocked action and the remaining proposal, and it does not promise that the goal set is still reachable, since a bridge may spend a resource a later step needs. MinFix targets the preconditions of a blocked action, so a soft rule that fails while the action stays executable leaves the bridge empty and the violation appears in the soft-constraint rate.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Environments. Town-Benchmark supplies the mechanism analysis, ScienceWorld the external validation. Town-Benchmark holds 1,000 instances over five domains, built from 250 base tasks that each admit a verified solution, so feasibility is decidable and the violation labels are exact. Figure 2 orders the five domains by how strictly each one constrains an action, and Appendix A lists the instance composition and the ten registered rules. ScienceWorld supplies 30 task types with a graded rubric of its own (Wang et al., 2022), which gives this evaluation the property Town-Benchmark lacks, a scorer written by other authors, and we adapt it by registering one predicate for the 38 preconditions the simulator exposes as explicit validity checks.

Protocol. The 1,000 instances divide in half at the base task, so 500 fit the calibration and 500 carry every reported rate. The language model, the demonstration set, the horizon and the retry allowance are identical across methods, and the guard parameters are fitted once, on the violation signal the environment returns for the training half, and then frozen for every reported rate. The symbolic checker is shared by every method that invokes it, so a difference comes from the decision the guard makes. The repair search uses the released depth limit of five actions and node-expansion limit of 2,000, and every reported rate covers the same 500 instances.

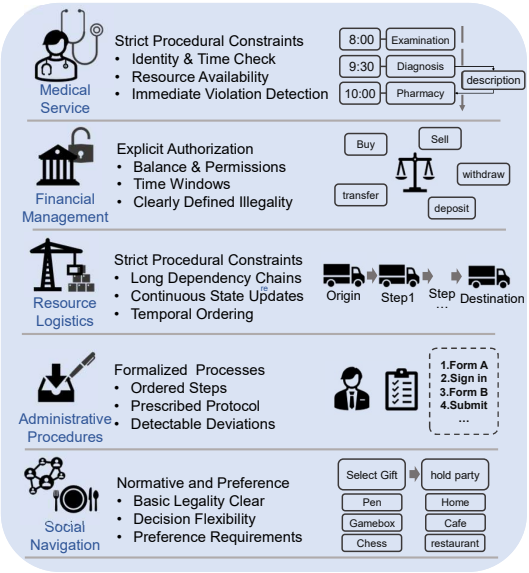


Figure 2: The five domains, ordered by constraint strictness, with one example each.

Metrics. Task completion rate (TCR) counts the instances that reach a goal state inside the horizon. The overall hallucination rate (OHR), the hard-constraint violation rate (HCVR) and the soft-constraint violation rate (SCVR) count violated step indicators of Equation 2 over the executed steps. Cost is language-model calls, total tokens and wall-clock latency in seconds, all measured per instance of the evaluation half and with retries included. Guard precision (GP) and guard recall (GR) score an intervention against the exact symbolic checker, and localisation accuracy (LOC) is the share of interventions whose first-violation localisation names the rule that the checker identified. ScienceWorld additionally reports the environment's graded score and the fraction of executed actions that satisfied their preconditions. Appendix C.3 accounts for the inference cost of each configuration, domain by domain, alongside the pooled average Table 1 reports, so a domain can be read against that pool.

Baselines. Prompting baselines are direct generation (Base), chain-of-thought (Wei et al., 2022) and self-refinement (Madaan et al., 2023), and retrieval baselines are RAG and a knowledge graph (Yu & McQuade, 2025; Zhao et al., 2025). The fine-tuned baseline is Qwen2.5-14B (Song et al., 2024), the released checkpoint, evaluated on the inference backbone. Constrained decoding, in the registry variant of the three the appendix lists, best-of-N with symbolic reranking, and AgentSpec (Wang et al., 2026a) with rules we write for the ten registered constraints all consult the registry. Every method reads the same observation, goal statement and instruction and differs only in what it reads from the registry, so Appendix C tabulates all thirteen configurations and names the variants that appear only there, every row of it on the same instances and backbone.

5.2 OVERALL HALLUCINATION MITIGATION AND EFFICIENCY

This experiment asks whether the guard and the repair stage improve violation, task completion and cost at the same time under one protocol. Table 1 carries the evidence for every configuration. NSAG reaches a completion rate of 0.99 against 0.72 for direct generation, and the hard-constraint violation rate moves the other way, from 0.072 to 0.003. The prompting and retrieval baselines move both

Table 1: Overall results on Town-Benchmark, averaged over the five domains and the instances, with completion a mean over five seeds of the 500 instances of the evaluation half. Tokens and Latency are per instance, retries included, and Appendix C gives both by domain.

Method	TCR ↑	OHR ↓	HCVR ↓	Tokens	Latency (s)
Base	0.72	0.102	0.072	200k	1567
CoT	0.85	0.133	0.082	197k	1479
Self-refine	0.82	0.091	0.059	491k	3089
KG	0.90	0.108	0.082	275k	1602
RAG	0.90	0.126	0.068	178k	1686
SFT (released 14B)	0.88	0.070	0.044	200k	1568
Constrained decoding	0.86	0.074	0.041	214k	1704
Best-of-N + reranking	0.93	0.056	0.029	1180k	4120
AgentSpec	0.94	0.041	0.019	231k	1815
NSAG	0.99	0.018	0.003	190k	1507

quantities less. Chain-of-thought reaches 0.85 at 0.082, self-refinement 0.82 at 0.059, and retrieval 0.90 at 0.068, so a stronger prompt changes which actions are proposed while the constraint check stays inside the generator. The released fine-tune reaches 0.88 at 0.044, and it was trained on a 14B model and evaluated on the inference backbone, so that row mixes the training procedure with the model that generates the steps, and we read it as the released configuration. Among the comparators that do consult the registry, constrained decoding reaches 0.041, best-of-N with symbolic reranking reaches 0.029 and AgentSpec reaches 0.94 at 0.019, so consulting the registry at each step moves the violation rate further than a stronger prompt does on its own.

On cost, NSAG spends 190k tokens and 1507 seconds per instance against 200k and 1567 for direct generation, while self-refinement and best-of-N spend 491k and 1180k. The symbolic work is a small share of that total, at 11.4 predicate evaluations per step and 38.6 seconds of symbolic CPU time against 1507 seconds of end-to-end latency. The decision layer therefore costs less than generating one more proposal, and repair accounts for the remainder.

The joint result is that one protocol yields fewer violations and a higher completion rate at a lower token count. The boundary is that we implement the AgentSpec rules ourselves, so its row reflects our encoding of the ten constraints, and it does not measure the accuracy its own rule-generation pipeline would reach if it wrote the rules itself from task text.

5.3 Contributions of Learned Intervention and Repair

This experiment separates what the learned calibration contributes from what the repair stage contributes. Four configurations share the backbone and the repair budget. *NSAG* learns the rule sensitivities w_j and the tolerance γ and anneals the temperature τ_{temp}, and it schedules its checks with the Rule Trigger Predictor. *NSAG w/o calibration* keeps the active set the predictor returns and replaces the learned parameters with one fixed uniform weight, so the repair stage and the set of checked rules stay in place. *NSAG w/ random gate* keeps the same coverage and draws its check decision from a fixed Bernoulli distribution, which separates a learned schedule from a sampled one. *NSAG w/o MinFix* ends the episode at the first detected violation, so its completion rate reads off what detection alone delivers. Figure 3 carries the evidence for the task outcome and the diagnosis quality, and the table in Appendix D adds the soft-constraint rate, the call and token counts and the per-step checking time. It reports the one-time training cost apart from the inference cost of the evaluation, so that the one-time and the per-instance figures never share a column.

Learned calibration reaches a completion rate of 0.99 at a hard-constraint violation rate of 0.003 against 0.96 and 0.028 for uniform weighting, which puts the gap at 0.025. Its guard precision is 0.94 against 0.78 and its localisation accuracy is 0.89 against 0.71, so the calibration buys both a tighter boundary and a more accurate statement of why a step was blocked, at 1507 seconds per instance against 1498. The random gate reaches 0.041 with precision 0.52, which separates what the guard learns from what it samples while checking the same rules at the same cost.

Removing the repair stage moves completion to 0.68 while the hard-constraint violation rate stays at 0.004, so the two stages cover different failures. Detection alone stops the episode at the first blocked step, so a task whose plan a repair could have continued counts as a failure, and the gap between 0.99 and 0.68 is the size of that population. The repair is cheap on this benchmark, and its

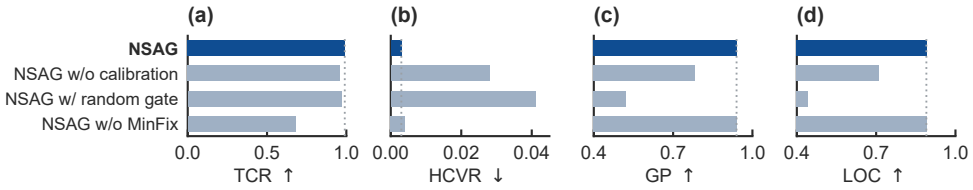


Figure 3: Four configurations of the learned intervention share the backbone and the repair budget, and the panels report task completion (a), hard-constraint violation (b), guard precision (c) and localisation accuracy (d). The repair stage accounts for most of the completion gain.

budget study covers 28 of the 500 instances over five seeds. On those 140 episodes MinFix fires on 312 blocked steps, 2.2 an episode, at 47.3 node expansions per repair, and it hits a search limit on 10 of the 312 repairs, at 3.2%. The boundary is that the comparison concerns one environment and ten rules, and that the calibration is fitted on the same five domains it is evaluated on, so a category absent from the training half is not covered, at 42 executed actions per instance. Appendix B.8 records the module shapes, the loss weights and the training schedule behind them.

5.4 Sensitivity to a Degraded Registry

This experiment asks what the guard and the repair stage depend on.

Both read every rule from the registry, so we remove $k \in \{0, 10, 20, 30, 50\}$ percent of the hard preconditions from it and rerun unguarded generation, the guard without repair and NSAG on the same instances. Figure 4 plots completion and hard-constraint violation against k for all three, which are the solvers the rule-completeness table of Appendix E lists.

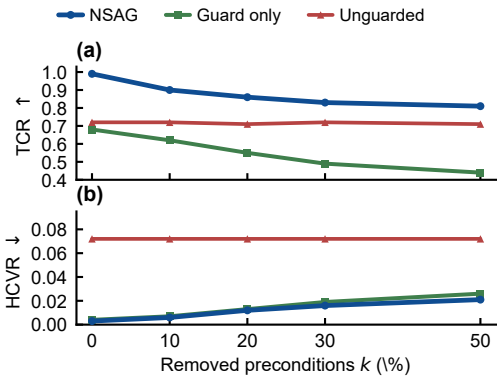


Figure 4: Completion (a) and violation (b) as the registry loses hard preconditions, for unguarded generation, the guard alone and NSAG.

Unguarded generation is insensitive to the removal because it never reads the registry, which is also why its violation rate stays at 0.072 across the whole sweep. NSAG reaches 0.90 at $k = 10$ and 0.81 at $k = 50$, and its violation rate rises from 0.003 to 0.021, which stays below the unguarded rate at every level of removal. Where the registry holds no goal predicate to search for and the goal reaches the guard only as text, NSAG reaches 0.88.

The boundary is that removal is simulated by deleting registry entries, so the sweep models a registry with missing entries. Appendix E covers the separate case of entries that are wrong, where the guard blocks legal steps and the violation rate alone no longer shows the cost.

5.5 Generalization Across Environments and Backbones

This experiment asks whether the decision layer and the repair stage hold outside the environment and the backbone we tuned on, where neither is fixed in deployment. Figure 5 reports two pairs of panels that share a protocol but no metric, one for the external environment and one for the backbones, and each pair is read on its own terms rather than pooled with the other.

Panels (a) and (b) repeat the pipeline on ScienceWorld, whose preconditions we register without changing the guard. The graded score replaces the binary success indicator and the violation rate counts executed actions that failed their preconditions. Direct generation reaches a normalised score of -0.07 with a violation rate of 0.237, while NSAG reaches 0.61 at a violation rate of 0.069. Coverage sets the boundary here, since a precondition we did not register is a violation the guard cannot see, and Table 2 records how much of the environment it reaches.

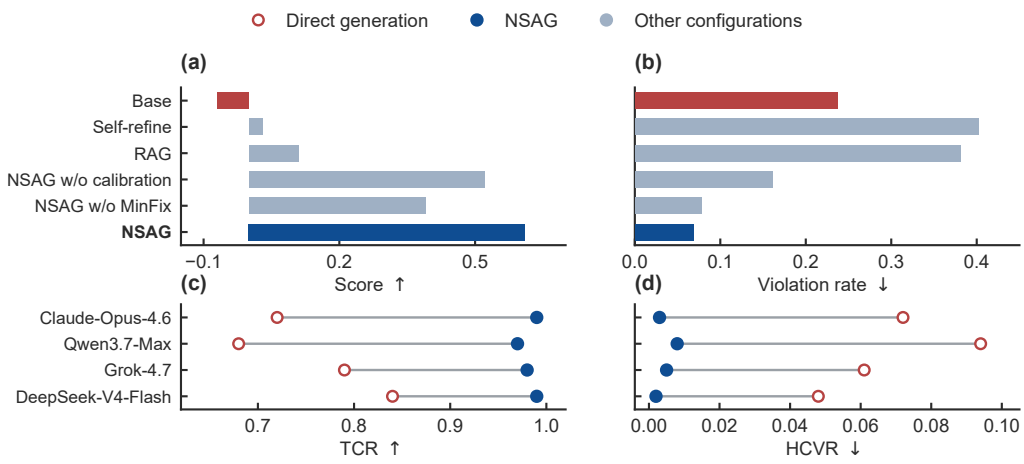


Figure 5: Panels (a) and (b) repeat the pipeline on ScienceWorld, where the environment's own graded score replaces the binary success indicator. Panels (c) and (d) pair each backbone with itself, so a row shows the gain and the point the backbone started from.

Panels (c) and (d) pair each backbone against itself, with direct generation as the open marker and NSAG as the filled one, so a row is read across. Completion rises and the hard-constraint violation rate falls in every row, and we keep the absolute values because a pooled gain would hide where each backbone starts. The largest gains sit in the row that starts worst, at 0.68 completion and 0.094 violation, and the smallest in the one that starts best, so recovery follows the starting point, whichever family the model comes from. Those backbones still make stochastic errors on long-horizon tasks, so the failures the guard removes do not follow parameter count alone. Appendix F reports the full ScienceWorld comparison and the exact values behind both pairs of panels, so every plotted number can be checked against its table, in both pairs.

5.6 Scaling and Robustness

The final question is where the repair search and the guard stop working. Enlarging the action space to $|\mathcal{A}| = 500$ registered action types by adding distractor locations and interaction targets raises the share of repairs that find no bridge inside the budget from 0.032 at $|\mathcal{A}| = 10$ to 0.142. The node-expansion limit drives the growth, so the search runs out of budget on a still-solvable instance. Under partial observability at $\alpha = 0.5$, stochastic transitions at $\beta = 0.2$ and noisy observations at $\gamma = 0.25$ the completion rate moves to 0.94, 0.92 and 0.90 while the hard-constraint violation rate stays near 0.011 because the rules the guard reads are unchanged under all three perturbations. The engine-only control row is the unguarded environment under the same perturbation, and a configuration that matches it has added nothing. The budget sweep degrades the search and the perturbation sweep degrades what the agent observes, so a degraded view of the state costs completion before it costs safety. Appendix G reports both sweeps in full, with the unfavourable rows and the failure conditions kept in place, so a reader can see which conditions the layer does not survive.

6 Conclusion

We presented NSAG, a governance layer that separates rule enforcement from the proposal a language model generates. Its Rule Trigger Predictor selects the rules a task category can activate, a differentiable guard calibrates the intervention decision from exact symbolic violation signals, and MinFix repairs a blocked step by supplying the preconditions that blocked it. Under one protocol on Town-Benchmark the layer lowers the hard-constraint violation rate and raises task completion, the same ordering holds on ScienceWorld, and four backbones improve without extra generation. The reductions are empirical, since the intervention is a learned threshold, the calibration is fitted on the five domains it is evaluated on, and the layer governs individual actions, so behaviour that breaks no registered rule stays outside it. Next steps are transitions the guard does not model and registries an operator cannot complete by hand.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish the writing, to draft and revise sections of the manuscript, to search for and check related work, to support research ideation and the execution of the experiments, and to generate the synthetic instances of Town-Benchmark. The same tools wrote and revised the plotting and the audit scripts that produce the figures and the layout reports. We did not use generative AI tools for any other item on the policy's required-disclosure list. We reviewed all AI-assisted work. Every data figure is generated from a CSV whose values were checked against the table the figure summarises, and the layout reports that the figures were measured with are reproducible from the released scripts. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

REFERENCES

Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. Semantic strengthening of neuro-symbolic learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 10252–10261. PMLR, 2023.

Bikram Pratim Bhuyan, Amar Ramdane-Cherif, Ravi Tomar, and TP Singh. Neuro-symbolic artificial intelligence: a survey. *Neural Computing and Applications*, 36(21):12809–12844, 2024.

Trishna Chakraborty, Udita Ghosh, Xiaopan Zhang, Fahim Faisal Niloy, Yue Dong, Jiachen Li, Amit K Roy-Chowdhury, and Chengyu Song. Heal: An empirical study on hallucinations in embodied agents driven by large language models. *arXiv preprint arXiv:2506.15065*, 2025.

Jiale Cheng, Xiao Liu, Cunxiang Wang, Xiaotao Gu, Yida Lu, Dan Zhang, Yuxiao Dong, Jie Tang, Hongning Wang, and Minlie Huang. Spar: Self-play with tree-search refinement to improve instruction-following in large language models. In *ICLR*, 2025.

Jiazhan Feng, Ruochen Xu, Junheng Hao, Hiteshi Sharma, Yelong Shen, Dongyan Zhao, and Weizhu Chen. Language models can be deductive solvers. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4026–4042, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.254. URL https://aclanthology.org/2024.findings-naacl.254/.

Xudong Guo, Kaixuan Huang, Jiale Liu, Wenhui Fan, Natalia Vélez, Qingyun Wu, Huazheng Wang, Thomas L Griffiths, and Mengdi Wang. Embodied llm agents learn to cooperate in organized teams. In *Language Gamification-NeurIPS 2024 Workshop*, 2024.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14953–14962, June 2023.

Hanxu Hu, Hongyuan Lu, Huajian Zhang, Yun-Ze Song, Wai Lam, and Yue Zhang. Chain-of-symbol prompting for spatial reasoning in large language models. In *First Conference on Language Modeling*, 2024.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024a.

Yidong Huang, Jacob Sansom, Ziqiao Ma, Felix Gervits, and Joyce Chai. Drivlme: Enhancing llm-based autonomous driving agents with embodied and social experiences. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3153–3160. IEEE, 2024b.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents, 2024. URL https://arxiv.org/abs/2402.03610.

Shyam Sundar Kannan, Vishnunandan L. N. 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)*, pp. 12140–12147, 2024. doi: 10.1109/IROS58592.2024.10802322.

Callie Y Kim, Christine P Lee, and Bilge Mutlu. Understanding large-language model (llm)-powered human-robot interaction. In *Proceedings of the 2024 ACM/IEEE international conference on human-robot interaction*, pp. 371–380, 2024.

Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*, 2024.

Xixun Lin, Yucheng Ning, Jingwen Zhang, Yan Dong, Yilong Liu, Yongxuan Wu, Xiaohua Qi, Nan Sun, Yanmin Shang, Kun Wang, et al. Llm-based agents suffer from hallucinations: A survey of taxonomy, methods, and directions. *arXiv preprint arXiv:2509.18970*, 2025.

Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. Deductive verification of chain-of-thought reasoning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 36407–36433. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/72393bd47a35f5b3bee4c609e7bba733-Paper-Conference.pdf.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Lesly Miculicich, Mihir Parmar, Hamid Palangi, Krishnamurthy Dj Dvijotham, Mirko Montanari, Tomas Pfister, and Long T. Le. Veriguard: Enhancing LLM agent safety via verified code generation. *arXiv preprint arXiv:2510.05156*, 2025.

Tong Mu, Alec Helyar, Johannes Heidecke, Joshua Achiam, Andrea Vallone, Ian Kivlichan, Molly Lin, Alex Beutel, John Schulman, and Lilian Weng. Rule based rewards for language model safety. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 108877–108901. Curran Associates, Inc., 2024. doi: 10.52202/079017-3457. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/c4e380fb74dec9da9c7212e834657aa9-Paper-Conference.pdf.

Uzma Nawaz, Mufti Anees-ur Rahaman, and Zubair Saeed. A review of neuro-symbolic ai integrating reasoning and learning for advanced cognitive systems. *Intelligent Systems with Applications*, pp. 200541, 2025.

Mengxu Pan, Alexandra Kitson, Hongyu Wan, and Mirjana Prpa. Ellma-t: an embodied llm-agent for supporting english language learning in social vr. DIS '25, pp. 576–594, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714856. doi: 10.1145/3715336.3735786. URL https://doi.org/10.1145/3715336.3735786.

Aske Plaat, Max van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. Agentic large language models, a survey. *arXiv preprint arXiv:2503.23037*, 2025.

Tanawan Premsri and Parisa Kordjamshidi. Neuro-symbolic training for reasoning over spatial language. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2395–2414, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.128. URL https://aclanthology.org/2025.findings-naacl.128/.

Amit Ray. Ai agent engineering: Building llm-cot-powered intelligent ai agents (total guide). *Compassionate AI*, 2(4):87–89, 2025.

Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. Meminsight: Autonomous memory augmentation for llm agents, 2025. URL https://arxiv.org/abs/2503.21760.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Agentbank: Towards generalized llm agents via fine-tuning on 50000+ interaction trajectories. *CoRR*, 2024.

Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2023.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11888–11898, October 2023.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE '26)*, pp. 2938–2950, New York, NY, USA, 2026a. Association for Computing Machinery. doi: 10.1145/3744916.3764546. URL https://doi.org/10.1145/3744916.3764546.

Haoyu Wang, Christopher M. Poskitt, Jiali Wei, and Jun Sun. Probguard: Proactive runtime monitoring for LLM agent safety via probabilistic prediction. 2026b. To appear; arXiv:2508.00500.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.775. URL https://aclanthology.org/2022.emnlp-main.775/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Yuxi Wei, Zi Wang, Yifan Lu, Chenxin Xu, Changxing Liu, Hao Zhao, Siheng Chen, and Yanfeng Wang. Editable scene simulation for autonomous driving via collaborative llm-agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15077–15087, 2024.

Ruoxuan Yang, Xinyue Zhang, Anais Fernandez-Laaksonen, Xin Ding, and Jiangtao Gong. Driving style alignment for llm-powered driver agent. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11318–11324. IEEE, 2024a.

Xiao-Wen Yang, Jie-Jing Shao, Lan-Zhe Guo, Bo-Wen Zhang, Zhi Zhou, Lin-Han Jia, Wang-Zhou Dai, and Yu-Feng Li. Neuro-symbolic artificial intelligence: towards improving the reasoning abilities of large language models. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 10770–10778, 2025.

Yijun Yang, Tianyi Zhou, Kanxue Li, Dapeng Tao, Lusong Li, Li Shen, Xiaodong He, Jing Jiang, and Yuhui Shi. Embodied multi-modal agent trained by an llm from a parallel textworld. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 26275–26285, 2024b.

Ziyi Yang, Shreyas S. Raman, Ankit Shah, and Stefanie Tellex. Plug in the safety chip: Enforcing constraints for llm-driven robot agents. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14435–14442, 2024c. doi: 10.1109/ICRA57147.2024.10611447.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*, 2025.

Hong Qing Yu and Frank McQuade. Rag-kg-il: A multi-agent hybrid framework for reducing hallucinations and enhancing llm reasoning through rag and incremental knowledge graph learning integration. *arXiv preprint arXiv:2503.13514*, 2025.

Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pan, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, Bo An, and Qingsong Wen. A survey on trustworthy llm agents: Threats and countermeasures. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, pp. 6216–6226, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714542. doi: 10.1145/3711896.3736561. URL https://doi.org/10.1145/3711896.3736561.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Xinjie Zhao, Moritz Blum, Fan Gao, Yingjian Chen, Boming Yang, Luis Marquez-Carpintero, Mónica Pina-Navarro, Yanran Fu, So Morikawa, Yusuke Iwasawa, et al. Agentigraph: A multi-agent knowledge graph framework for interactive, domain-specific llm chatbots. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 6757–6761, 2025.

Zhilun Zhou, Jingyang Fan, Yu Liu, Fengli Xu, Depeng Jin, and Yong Li. Synergizing llm agents and knowledge graph for socioeconomic prediction in lbsn. *CoRR*, 2024.

Tinghui Zhu, Kai Zhang, Jian Xie, and Yu Su. Deductive beam search: Decoding deducible rationale for chain-of-thought reasoning. In *First Conference on Language Modeling*, 2024.

APPENDIX GUIDE

The appendix follows the paper from the environment and the shared protocol through the method details to the component analyses and the external evidence, and it closes with where the layer sits in the neuro-symbolic literature and what the reported reductions do not establish. Every table and figure is named in the subsection that describes its experiment.

A Environment and Evaluation Details p. 15
- A.1 Town-Benchmark: Domains and Instances 15
- A.2 ScienceWorld: Registered Preconditions and Coverage 15
- A.3 Shared Protocol and Frozen Parameters 15

B Method Details p. 16
- B.1 Prompt Construction and Decoding 16
- B.2 Distributional Mismatch 16
- B.3 Rule Trigger Predictor 17
- B.4 Training the Guard 17
- B.5 Formal Definitions 17
- B.6 Search Configuration and Cost 18
- B.7 Activation Logs and Pruning Threshold 19
- B.8 Mechanism Specification 19

C Complete Overall Results p. 19
- C.1 Every Comparator on the Shared Protocol 19
- C.2 Per-Domain Task Metrics 20
- C.3 Per-Domain Cost Axes 20

D Additional Component Analyses p. 21
- D.1 Mechanism Ablation in Full 21
- D.2 Inference Cost of the Four Variants 22
- D.3 Rule Granularity: Imperative, Template and First-Order 22
- D.4 Pruning-Threshold Sensitivity 22

E Registry Completeness and Rule Quality p. 23
- E.1 Missing Hard Preconditions 23
- E.2 Wrong and Induced Rules 23

F External and Cross-Backbone Results p. 24
- F.1 ScienceWorld: Full Comparison 24
- F.2 Cross-Backbone Paired Values 24

G Scaling and Robustness p. 24
- G.1 Repair Budget at the Released Constants 24
- G.2 Action-Space Scaling 25
- G.3 Perturbations of the Environment 25

H Neuro-Symbolic Foundations p. 26

I Boundaries of the Reported Claims p. 27

A Environment and Evaluation Details

A.1 Town-Benchmark: Domains and Instances

The five domains are medical service, financial management, resource logistics, administrative procedures and social navigation, and Figure 2 in the main text orders them by how strictly each one constrains an action. The order matters for the guard. A domain whose rules a single state settles gives the checker an exact answer at the step, while a domain whose rules are normative becomes decidable only across a sequence of steps, and the calibration weighs the second kind with less evidence at the step. The 250 base tasks are split 50 per domain, and each base task carries four initial-parameter settings that vary the starting location, the resource levels and the documents the agent holds, so an instance can be illegal at the first step or become illegal later in the plan. The four settings give 1,000 instances, 200 instances in each domain, and the environment grades the 250 base tasks 96 easy, 104 medium and 50 hard. The action types are MOVE, WAIT and INTERACT, and the ten registered rules cover location, opening hours, energy, financial limits, possession, prescription requirements, documentation, restricted locations, access rights and qualifications. A task admits at least one verified solution, which the environment's own solver produces from the complete registry, so a violation label comes from the checker and needs no annotation. The 1,000 instances divide in half at the base task, so the training half and the evaluation half hold 500 each, and every reported rate is a five-seed mean over the evaluation half. The guard's calibration is fitted once on the training half and then frozen for every reported configuration.

A.2 ScienceWorld: Registered Preconditions and Coverage

ScienceWorld states the precondition of each action in its own simulator, so the adaptation reads that precondition, registers one predicate per precondition and applies the guard to the executed actions of an episode. Table 2 records the correspondence and the part of the environment the adaptation leaves out. A precondition the simulator keeps implicit cannot be registered, and a violation against it is invisible to the guard and absent from the violation labels, which is why Appendix F reports coverage next to every number that the guard produced on that task set.

Table 2: Mapping the guard interface onto ScienceWorld. The registered set covers 38 of the 61 preconditions the environment exposes as explicit validity checks, and the remaining 23 are exposed but have no predicate the interface can register, so the guard cannot reach them and no violation label covers them. The external comparison of Table 12 runs under that coverage.

NSAG concept	ScienceWorld construct	Coverage and boundary
Hard constraint	Precondition of an action, exposed by the environment's validity check	38 / 61 preconditions expressed as predicates
Rule r_j	One precondition predicate	38 predicates over 6 typed slots
Violation $v_{t,j} = 1$	Proposed action outside the valid set at s_t	Exact, no manual labelling
Task success	Rubric score of the episode	Graded, scale 100
Task type coverage	6 of 30 task types	12 variations, first exclusion `chemistry-mix`
Excluded	Preconditions the adaptation does not express as predicates	Not checkable, so no violation against them can be detected

A.3 Shared Protocol and Frozen Parameters

The default backbone is Claude-Opus-4.6, and every number in the main text is produced with it unless a study explicitly varies the language model. Every method receives the same observation, the same instruction text and the same goal statement, and the demonstration set is fixed across methods so that a prompt difference cannot be confused with a registry difference. The horizon of 60 actions, the retry allowance of 20 and the decoding temperature are held fixed, an instance executes 42 of those actions on average, and the guard parameters are fitted once on the training half and then frozen for every reported run. A retry is one re-planning attempt after a refused action: the refusal leaves the state unchanged, so the method proposes again, and the episode ends when it has spent all twenty or when the horizon expires. A refused action is charged to the method

that proposed it, since the call and latency columns of Table 4 count every attempt. The symbolic checker that supplies the violation labels of Appendix B is the same one that the guard and the repair search call, so a difference between two rows comes from the decision layer and from the repair it triggers, and two rows can share one implementation of the registry. Appendix B.1 lists the context construction, the number of demonstrations and the decoding temperature used for every method, on the same retry allowance and the same 500 instances of the evaluation half, which is the denominator of every rate the paper reports. The training half never enters a reported number.

B Method Details

B.1 Prompt Construction and Decoding

The context concatenates the system instruction $\mathcal{I}_{sys}$, k demonstration trajectories $\{\mathcal{D}_{few}^{(i)}\}_{i=1}^{k}$, the linearized initial state $\mathrm{lin}(s_0)$ and the textual goal statement $x_{\mathcal{G}}$,

$$\mathbf{x}_{prompt} = \mathcal{I}_{sys} \oplus \left(\bigoplus_{i=1}^{k} \mathcal{D}_{few}^{(i)} \right) \oplus \mathrm{lin}(s_0) \oplus x_{\mathcal{G}}, \tag{11}$$

with $k = 4$ and decoding temperature $\tau_{\mathrm{dec}} = 0.7$. Here lin is the linearization of a structured state, and $x_{\mathcal{G}}$ is the goal written as text, whose set $\mathcal{G}$ is the one that scores success. The generator emits a token sequence $\mathbf{w}$ one token at a time, so the probability of a single token v reads off the entry of a softmax vector that corresponds to v,

$$P_\phi(v \mid \mathbf{w}_{<j}, \mathbf{x}_{prompt}) = \left[\mathrm{softmax}\left(\tau_{\mathrm{dec}}^{-1} \mathbf{h}_j^\top \mathbf{W}_E \right) \right]_v, \tag{12}$$

where $\mathbf{h}_j \in \mathbb{R}^{d_{model}}$ is the hidden state before position j, $\mathbf{W}_E \in \mathbb{R}^{d_{model} \times |\mathcal{V}|}$ is the output matrix, $[\cdot]_v$ selects the entry for token v and $\mathcal{V}$ is the token vocabulary. A parser $\Psi : \mathcal{V}^* \to \mathcal{U}^* \cup \{\text{PARSE-FAILURE}\}$ segments the decoded string into discrete actions, so the trajectory likelihood factorizes over steps,

$$P_\phi(\mathbf{u}_{1:K} \mid \mathbf{x}_{prompt}) = \prod_{t=1}^{K} P_\phi\big(\Psi^{-1}(u_t) \mid \mathbf{u}_{<t}, \mathbf{x}_{prompt}\big), \tag{13}$$

and the product is defined only when the parse succeeds. Several token strings can realise the same action, and we sum their probabilities, so $P_\phi(\Psi^{-1}(u_t) \mid \cdot)$ is the mass of the action itself, summed over every phrasing of it. A step the parser cannot resolve is counted as a violation, which is how the unparseable share of the proposal enters the reported violation rate.

B.2 Distributional Mismatch

The policy spreads probability over the whole action vocabulary while the environment admits only $\mathcal{U}_{\text{valid}}(s_{t-1})$ of Section 3, so the mass assigned to refused transitions is

$$\mathcal{H}_{gap}(s_{t-1}) = \sum_{u \notin \mathcal{U}_{\text{valid}}(s_{t-1})} P_\phi(u \mid \mathbf{u}_{<t}, \mathbf{x}_{prompt}). \tag{14}$$

Let V_t be the event that the t-th proposed action is executable, read against the state that precedes it. When every conditional violation probability is bounded away from zero the probability of an entirely executable proposal is

$$P(V_{1:K}) = \prod_{t=1}^{K} P(V_t \mid V_{1:t-1}) \le (1-\epsilon)^K \le e^{-\epsilon K}, \tag{15}$$

for some $\epsilon > 0$. The factorisation needs the conditional form and nothing else, so the actions need not be independent, and the bound says nothing about a single step. This quantity motivates the external guard, since an in-context instruction bounds neither the support of the policy nor the set of transitions the environment accepts. Sections 3 and this one both work with the environment's own transition function $\mathcal{T}$. Where a guard replays a proposal against its own model of the environment, we write $\mathcal{T}_{\mathrm{model}}$ for that replay and keep $\mathcal{T}$ for the environment.

B.3 Rule Trigger Predictor

A task category c is projected into a dense vector $\mathbf{e}_c \in \mathbb{R}^{d_{model}}$ by a learnable embedding matrix $\mathbf{E}$, and a lightweight MLP with parameters ψ estimates the Bernoulli probability that each rule is active,

$$\mathbf{p}(c) = \sigma(\mathrm{MLP}_\psi(\mathbf{e}_c)) = [p_1, \ldots, p_M]^\top, \qquad M = |\mathcal{R}|, \tag{16}$$

and a deterministic threshold $m_j(c) = \mathbb{I}(p_j(c) > \tau_{prune})$ yields the reduced active set $\mathcal{R}_{active}(c)$, which lowers verification cost from $\mathcal{O}(K \cdot |\mathcal{R}|)$ to $\mathcal{O}(K \cdot |\mathcal{R}_{active}|)$. The predictor is trained with binary cross-entropy against the applicability labels $\mathbf{y} \in \{0,1\}^M$ of Appendix B.7,

$$\ell_{prior}(\psi) = -\frac{1}{M} \sum_{j=1}^{M} \left[y_j \log p_j + (1 - y_j) \log(1 - p_j) \right], \tag{17}$$

and it is trained on its own objective, so a hard prune at run time is not differentiated through. Appendix B.7 states where the applicability labels come from and how τ_{prune} is selected from those labels.

B.4 Training the Guard

The guard is supervised by the exact checker, so the label of a step is the violation indicator h_t of Equation 2 and the per-rule status $v_{t,j}$ of Equation 4 is the reason for it. Both are read off the proposal before any repair, so the steps the guard blocks also contribute labels, and the length K in the supervision uses the length of the proposal, and the rectified sequence has its own length. The predictor's applicability labels are collected in the same pass. The compound objective is $\ell_{total} = \ell_{rule} + \lambda_{pg}\ell_{pg}$, where the class-balanced binary cross-entropy aligns the intervention probability q_t with the label,

$$\ell_{rule}(\Theta) = -\sum_{t=1}^{K} \Big[\alpha^+ y_t \log q_t + \alpha^- (1 - y_t) \log(1 - q_t) \Big], \qquad y_t = h_t, \tag{18}$$

with α^+ and α^- weighting the violating and clean steps by the inverse of their frequency, and the sampling term rewards interventions that preserve task completion. Writing the expected return of the sampling policy as $J(\Theta) = \mathbb{E}_{\mathbf{b} \sim P_\Theta}[R(\mathbf{b})]$, with $R \in \{0,1\}$ the reward of the rectified trajectory, the score-function estimator is

$$\nabla_\Theta J = \mathbb{E}\Big[(R - R_{\mathrm{base}}) \sum_{t=1}^{K} \nabla_\Theta \log P_\Theta\big(b_t \mid \mathrm{hist}_t\big) \Big], \qquad P_\Theta(b_t \mid \mathrm{hist}_t) = q_t^{b_t}(1 - q_t)^{1 - b_t}, \tag{19}$$

with $\mathrm{hist}_t = (s_0, x, \mathbf{u}_{1:t}, v_{t,\cdot})$, and ℓ_{pg} is the sampled surrogate of $-J$. The baseline R_{base} is a function of the history alone and carries a stop gradient, so it does not depend on the action sampled at step t. The name is kept distinct from the intervention decision b_t. Parameters follow $\Theta \leftarrow \Theta - \eta \nabla_\Theta \ell_{total}$, and at a fixed active set the gradient for a sensitivity follows from Equation 18 and Equation 7,

$$\frac{\partial \ell_{rule}}{\partial w_j} = \sum_{t:\, r_j \in \mathcal{R}_{\mathrm{active}}(c)} \alpha_t \left(q_t - y_t \right) \frac{p_j(c)\, \sigma(w_j)\, v_{t,j}}{\tau_{\mathrm{temp}}}, \qquad \alpha_t = \begin{cases} \alpha^+ & y_t = 1, \\ \alpha^- & y_t = 0, \end{cases} \tag{20}$$

which shows what the calibration does. The summand carries $v_{t,j}$, so an update touches only the rules that were actually broken at that step. When a violation was missed, $y_t = 1$ and $q_t < 1$ make the gradient negative and the descent step raises w_j. When the guard fires on a clean step, $y_t = 0$ and $q_t > 0$ make it positive and lowers w_j. The parameters therefore move from a uniform initialization toward a category-aligned profile. The exact module, tensor shapes, optimizer and training data source appear in Appendix B.8 with the optimizer and the schedule.

B.5 Formal Definitions

Definition B.1 (Environment)**.** An environment is a fully observable deterministic transition system $\mathcal{E} = (\mathcal{S}, \mathcal{U}, \mathcal{T}, \mathcal{R})$. A state $s = (\ell, \mathbf{o}, \mathbf{z}) \in \mathcal{S}$ consists of a discrete topology location $\ell \in \mathcal{L}$, discrete object states $\mathbf{o} \in \mathcal{O}$ and the remaining task-relevant quantities $\mathbf{z} \in \mathbb{R}^d$, which hold the clock, the resource levels and the progress flags a rule may read. An action $u \in \mathcal{U}$ is executable if and only

if its precondition holds, and the executable actions are $\mathcal{U}_{\text{valid}}(s) = \{u \in \mathcal{U} : \text{Pre}(s, u) = 1\}$. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$ defines the state evolution $s' = \mathcal{T}(s, u)$ upon execution of an executable action, so it is defined on $\mathcal{U}_{\text{valid}}(s)$, and a refused action is never executed. It leaves the state unchanged and spends one of the retry attempts the protocol grants, so an episode ends at the horizon or when its attempts are spent. The unguarded baseline of Section 5 spends none of them. Town-Benchmark instantiates this definition. Full observability and determinism are assumptions of that environment, and the interface itself needs only that the rules can be registered, which is how ScienceWorld is adapted to the same interface in Appendix A.

Definition B.2 (Rule Semantics and Verification)**.** A rule is a predicate $r : \mathcal{S} \times \mathcal{U} \to \{0, 1\}$ indicating validity. Partitioning the rule set into hard constraints $\mathcal{R}_{\text{hard}}$ (safety) and soft constraints $\mathcal{R}_{\text{soft}}$ (preferences) gives the collective verification functions

$$\text{Legal}(s, u) \triangleq \prod_{r \in \mathcal{R}_{\text{hard}}} r(s, u), \qquad \text{Comply}(s, u) \triangleq \prod_{r \in \mathcal{R}} r(s, u), \tag{21}$$

which evaluate to 1 if and only if all corresponding constraints are strictly and simultaneously satisfied. The hard rules are registered so that $\text{Legal}(s, u) = 1$ and $u \in \mathcal{U}_{\text{valid}}(s)$ agree on the actions the registry covers, so an action is executable exactly when the registered hard constraints hold, while a soft constraint may fail on an executable action, so the two properties are separate.

Definition B.3 (Task Specification and Success Criteria)**.** A task instance is characterised by an initial state and a target goal set, $(s_0, \mathcal{G})$. Given an executed action sequence $\mathbf{u}_{1:K'}$, the system evolves as $s_t = \mathcal{T}(s_{t-1}, u_t)$, and the success indicator is $\text{Succ}(\mathbf{u}_{1:K'}) \triangleq \mathbb{I}(\exists t \in \{0, \ldots, K'\} : s_t \in \mathcal{G})$, where s_0 is included and K' counts every executed action, including the ones a repair inserts. A trajectory is executable when every one of its actions satisfies $\text{Legal}(s_{t-1}, u_t) = 1$, and it is successful when it reaches a state in $\mathcal{G}$ before the horizon expires. The two properties are independent, since a trajectory can stay executable and miss the goal, and an inserted bridge counts against the same horizon as the steps the model proposed, so an inserted bridge always costs part of the budget that the remaining steps of the plan still need to run.

B.6 Search Configuration and Cost

The A* search evaluates each candidate state with $f_{\text{search}}(s) = g_{\text{search}}(s) + h_{\text{search}}(s)$, where $g_{\text{search}}(s)$ is the number of bridge actions already taken and the heuristic counts the preconditions still unsatisfied,

$$h_{\text{search}}(s) = |\mathcal{P}_{\text{miss}}(s, u)|. \tag{22}$$

The count biases expansion toward actions that satisfy an outstanding prerequisite. It is a lower bound on the remaining bridge length only when no single legal action satisfies two unmet preconditions at once, because one action that clears two of them would make the count exceed the true remaining length. We therefore report the bridge the solver returns and its length, treat the shortest bridge of Equation 9 as the design target, and make no claim that the solver always attains it. The default budget is a depth limit of 5 actions and a node-expansion limit of 2,000, matching `search_utils.py` in the released code, and Algorithm 1 lists the loop. A step costs at most $|\mathcal{R}|$ predicate evaluations of one state, and the guard queries the proposal and the state its replay of the proposal reaches, so a step costs at most $2|\mathcal{R}|$ of them; a repair costs at most $N_{\max}$ node expansions. The per-instance symbolic work therefore decomposes as $2K|\mathcal{R}| + R_{\text{repair}} N_{\max}$, where R_{repair} counts blocked steps and K is the number of executed actions. An instance executes 42 of them on average, which is what turns the per-step checking cost of Table 8 into a per-instance figure. The direct checks and the replay account for 42 of those milliseconds and the repair for the rest, and they close on the 38.6 s of symbolic work Section 5.2 quotes: a blocked step costs its 47.3 expansions at about 350 ms each, and the 2.2 blocked steps an episode reports in Table 14 come to 36.8 s of that total. That expression counts predicate evaluations and node expansions, so it is not a time complexity. The wall-clock cost of one expansion also depends on the successor branching factor, on the transition function and on the priority queue that orders the frontier.

A node carries the state reached along its own path together with that path, so the goal test reads $\text{Pre}(s', u)$ on the reached state. Node records are keyed by state and by remaining depth, and a state reached again with a smaller g_{search} replaces the stored record. The returned bridge is the one the search completed with, and the flag records how the search ended: COMPLETE with a bridge, BUDGET-STOPPED when the expansion limit ended the search before the space was exhausted, and

Algorithm 1 Bounded A* repair (MinFix)

Input: state s, blocked action u, depth limit $d_{\max}$, expansion limit $N_{\max}$
Output: a bridge with a completion flag, or $\perp$ when no bridge was found
expansions $\leftarrow 0$; $depthStopped \leftarrow$ FALSE
push the node $(s,\ \Delta = \langle\rangle,\ f = h_{\text{search}}(s))$ onto the priority queue
while the queue is not empty **do**
    if expansions $= N_{\max}$ **then**
        return $(\perp,$ BUDGET-STOPPED$)$
    end if
    pop the node with the smallest f: let s' be its state and Δ its path
    expansions $\leftarrow$ expansions $+1$
    if $\mathrm{Pre}(s', u) = 1$ **then**
        return $(\Delta,$ COMPLETE$)$
    end if
    if $|\Delta| \geq d_{\max}$ **then**
        $depthStopped \leftarrow$ TRUE; **continue**
    end if
    for all $a \in \mathcal{U}_{\text{valid}}(s')$ **do**
        $s'' \leftarrow \mathcal{T}(s', a)$; push $(s'',\ \Delta \| a,\ g_{\text{search}}{+}1{+}h_{\text{search}}(s''))$
    end for
end while
return $(\perp,$ DEPTH-STOPPED$)$ **if** $depthStopped$ **else** $(\perp,$ SEARCH-EXHAUSTED$)$

DEPTH-STOPPED when every node that was left sat at the depth limit. The last two are the two counts Table 14 reports for the released constants, as counts and as shares.

B.7 Activation Logs and Pruning Threshold

The Rule Trigger Predictor is trained on logs recorded while the guard runs with exhaustive checking on the training half. Each entry records the task category, the executed action, the status of every registered rule on that action and the resulting reward. The label y_j of Equation 17 records applicability, which is a separate quantity from violation. It is 1 when rule j is one that can constrain at least one action of the task category, which is a property of the rule and the category, and it is 0 when the rule never applies. A rule that applied and was obeyed is therefore a positive example, and a rule that never applied is a negative one, which keeps the predictor from learning to fire on rules only because they were never violated. Applicability is aggregated over every trajectory recorded for the category, so one positive anywhere in the category makes $y_j = 1$ for that category. The activation logs are kept apart from the violation matrix $\mathbf{V}$ of Appendix B.4, which carries the violation status of a step and is a separate label from the applicability of a rule. The sensitivity of completion and violation rates to $\tau_{\text{prune}} \in \{0.05, 0.10, 0.20, 0.30, 0.50\}$ and to the tolerance γ appears in Table 9, and the value used for the main tables is 0.20, the middle value of the grid.

B.8 Mechanism Specification

Table 3 lists the module, the tensor shapes, the loss terms, the optimizer and the source of the supervision signal, so that every parameter the calibration learns can be reproduced from the description. The supervision is the violation matrix $\mathbf{V}$ computed by the symbolic checker on the trajectories the guard checks, which is exact and requires no manual annotation. Only the collection of those trajectories costs API calls, and the collection is the whole of the one-time cost.

C Complete Overall Results

C.1 Every Comparator on the Shared Protocol

Table 4 lists every comparator on the protocol of the main table, including the configurations the main text compresses. Its released fine-tuned baseline was trained on a 14B model and evaluated against a larger inference backbone, so its violation rate mixes the effect of the training procedure

Table 3: Specification of the differentiable calibration. $M = 10$ is the registered rule count, $d_{model} = 256$ the hidden width, and the loss weights and the optimizer appear below.

Component	Symbol	Shape / value
Category embedding	$\mathbf{e}_c$	256
Prediction MLP	MLP_ψ	$256 \to 256 \to M$
Rule sensitivities	w_j	M, init 0.5
Tolerance	γ	scalar, init 0.5
Temperature	τ_{temp}	scalar, annealed $1.0 \to 0.1$
Pruning threshold	τ_{prune}	0.20
Loss weights	$\lambda_{pg}, \alpha^+, \alpha^-$	0.1, class-balanced
Optimizer	Adam	lr 1×10^{-3}, wd 1×10^{-4}
Epochs / batch	n/a	30 / 64
Training data	n/a	symbolic checker, on the training half

with the effect of the model that generates the steps, and the row is read as that released configuration. The constrained-decoding, best-of-N and AgentSpec rows each carry both the variant that the main text reports and the variants that only appear in this table.

Table 4: Every comparator on one protocol. Calls and Latency count every attempt a run made, which includes the re-planning call a refused action costs. SFT (released 14B) was trained on a 14B model and evaluated on the inference backbone, so its violation rate mixes the training procedure with the model that generates the steps. Every rate is a mean over the five seeds of the 500 instances of the evaluation half, the denominator the columns share, so the rows compare directly.

Method	TCR↑	OHR↓	HCVR↓	SCVR↓	#Calls	Tokens	Latency (s)
Base (direct)	0.72	0.102	0.072	0.030	123	200k	1567
CoT	0.85	0.133	0.082	0.051	97	197k	1479
Self-Refine	0.82	0.091	0.059	0.032	286	491k	3089
KG	0.90	0.108	0.082	0.026	88	275k	1602
RAG	0.90	0.126	0.068	0.058	87	178k	1686
SFT (released 14B)	0.88	0.070	0.044	0.026	123	200k	1568
Constrained decoding (grammar)	0.75	0.100	0.069	0.031	106	180k	1394
Constrained decoding (registry)	0.86	0.074	0.041	0.033	131	214k	1704
Constrained decoding (planner)	0.64	0.045	0.024	0.021	256	312k	2999
Best-of-N + symbolic rerank	0.93	0.056	0.029	0.027	738	1180k	4120
Best-of-N + early stop	0.92	0.061	0.034	0.027	523	772k	2687
AgentSpec	0.94	0.041	0.019	0.022	121	231k	1815
NSAG	0.99	0.018	0.003	0.015	119	190k	1507

C.2 Per-Domain Task Metrics

Table 5 breaks the main table down by domain and by metric for every method. Figure 6 carries the pattern of that breakdown, and the table carries every rate of it at full precision.

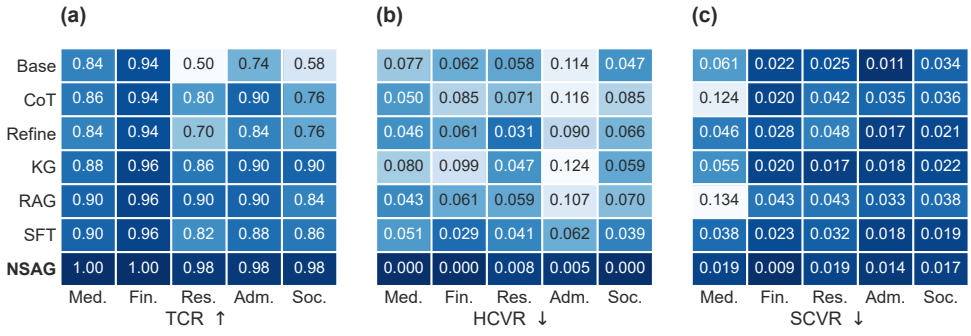


Figure 6: Each panel breaks one metric of the main table down by domain, and shading is darker where the value is better inside that panel. The method that leads the pooled table leads in every domain, and the gap is widest where the environment is hardest.

C.3 Per-Domain Cost Axes

Table 6 reports the cost side of the same breakdown, with the calls, tokens and wall-clock latency of each method in every domain alongside the pooled averages.

Table 5: Per-domain task metrics behind Table 1. Domain order is Med., Fin., Res., Adm., Soc., and the Avg. column repeats the pooled value so a single row can be read on its own. Rates are pooled violation counts over the executed steps of the evaluated instances, which are the 500 of the evaluation half, 100 in each domain, and each rate is a five-seed mean.

Method	Metric	Med.	Fin.	Res.	Adm.	Soc.	Avg.
Base	TCR ↑	0.84	0.94	0.50	0.74	0.58	0.72
	OHR ↓	0.135	0.084	0.083	0.125	0.081	0.102
	HCVR ↓	0.077	0.062	0.058	0.114	0.047	0.072
	SCVR ↓	0.058	0.022	0.025	0.011	0.034	0.030
CoT	TCR ↑	0.86	0.94	0.80	0.90	0.76	0.85
	OHR ↓	0.174	0.105	0.113	0.151	0.121	0.133
	HCVR ↓	0.050	0.085	0.071	0.116	0.085	0.082
	SCVR ↓	0.124	0.020	0.042	0.035	0.036	0.051
Refine	TCR ↑	0.84	0.94	0.70	0.84	0.76	0.82
	OHR ↓	0.092	0.089	0.079	0.107	0.087	0.091
	HCVR ↓	0.046	0.061	0.031	0.090	0.066	0.059
	SCVR ↓	0.046	0.028	0.048	0.017	0.021	0.032
KG	TCR ↑	0.88	0.96	0.86	0.90	0.90	0.90
	OHR ↓	0.135	0.119	0.064	0.142	0.081	0.108
	HCVR ↓	0.080	0.099	0.047	0.124	0.059	0.082
	SCVR ↓	0.055	0.020	0.017	0.018	0.022	0.026
RAG	TCR ↑	0.90	0.96	0.90	0.90	0.84	0.90
	OHR ↓	0.177	0.104	0.102	0.140	0.108	0.126
	HCVR ↓	0.043	0.061	0.059	0.107	0.070	0.068
	SCVR ↓	0.134	0.043	0.043	0.033	0.038	0.058
SFT	TCR ↑	0.90	0.96	0.82	0.88	0.86	0.88
	OHR ↓	0.089	0.052	0.073	0.080	0.058	0.070
	HCVR ↓	0.051	0.029	0.041	0.062	0.039	0.044
	SCVR ↓	0.038	0.023	0.032	0.018	0.019	0.026
NSAG	TCR ↑	**1.00**	**1.00**	**0.98**	**0.98**	**0.98**	**0.99**
	OHR ↓	**0.019**	**0.009**	**0.024**	**0.019**	**0.017**	**0.018**
	HCVR ↓	**0.000**	**0.000**	**0.008**	**0.005**	**0.000**	**0.003**
	SCVR ↓	**0.019**	**0.009**	**0.016**	**0.014**	**0.017**	**0.015**

Table 6: Per-domain inference cost of the same seven methods, with the domain order of the task table and the same pooled Avg. column. Latency is per instance, retries included.

Method	Metric	Med.	Fin.	Res.	Adm.	Soc.	Avg.
Base	#Calls	90	67	176	136	145	123
	Tokens	146k	106k	296k	211k	238k	200k
	Latency (s)	966	723	2250	1359	2540	1567
CoT	#Calls	96	66	129	72	120	97
	Tokens	200k	129k	260k	142k	255k	197k
	Latency (s)	1476	905	1949	1024	2040	1479
Refine	#Calls	258	234	366	252	321	286
	Tokens	447k	372k	635k	407k	593k	491k
	Latency (s)	2762	2130	3986	2440	4125	3089
KG	#Calls	96	76	104	73	90	88
	Tokens	305k	233k	325k	224k	289k	275k
	Latency (s)	1842	1269	1811	1223	1867	1602
RAG	#Calls	90	63	112	73	99	87
	Tokens	173k	105k	264k	123k	226k	178k
	Latency (s)	1572	1024	2127	1290	2418	1686
SFT	#Calls	90	68	175	135	146	123
	Tokens	148k	108k	292k	211k	240k	200k
	Latency (s)	972	732	2237	1349	2549	1568
NSAG	#Calls	**86**	**64**	**178**	**130**	**138**	**119**
	Tokens	**139k**	**101k**	**282k**	**202k**	**228k**	**190k**
	Latency (s)	**933**	**690**	**2151**	**1295**	**2466**	**1507**

D Additional Component Analyses

D.1 Mechanism Ablation in Full

Table 7 carries the mechanism evidence behind Figure 3 in full, with the soft-constraint rate and with the two controls the figure omits. The untrained row reports the same architecture with the calibration at its initialization, so it separates what the architecture contributes from what the fitted parameters contribute. The released-scheduler row reports the check schedule that ships with the code, which differs from the schedule the calibration selects.

Table 7: Mechanism evidence behind Figure 3, in full, including the two further controls. *NSAG w/ untrained calibration* reports the initialization before training, and *NSAG w/ released scheduler* reports the check schedule shipped with the code. Table 8 adds the cost columns. Completion is averaged over the five seeds of the 500-instance evaluation half, and the row for the guard without repair is the configuration Table 10 reports at $k = 0$, on the same half.

	Task-level				Guard quality (↑)		
Method	TCR ↑	OHR ↓	HCVR ↓	SCVR ↓	GP	GR	LOC
NSAG	0.99	0.018	0.003	0.015	0.94	0.91	0.89
NSAG w/o calibration	0.96	0.049	0.028	0.021	0.78	0.83	0.71
NSAG w/ random gate	0.97	0.068	0.041	0.027	0.52	0.47	0.44
NSAG w/o MinFix	0.68	0.019	0.004	0.015	0.94	0.91	0.89
NSAG w/ untrained calibration	0.93	0.061	0.036	0.025	0.71	0.76	0.62
NSAG w/ released scheduler	0.98	0.025	0.006	0.019	0.92	0.89	0.86

D.2 Inference Cost of the Four Variants

Table 8 accounts for the calls, tokens, latency, per-step predicate evaluations and per-step checking time of the same four configurations, and it reports the one-time training cost separately from inference, so per-step and one-time costs never share a column.

Table 8: Inference cost of the same four variants, with the one-time training cost of the calibration reported separately. Checks is the number of predicate evaluations per generated action and Chk/step the symbolic time one executed action costs, including the repair search a blocked action triggers, which is why the variant without MinFix sits two orders of magnitude below the others. An instance executes 42 actions on average, so Chk/step times 42 is the per-instance CPU time.

Method	#Calls	Tokens	Latency (s)	Checks	Chk/step (ms)	Train (min)
NSAG	119	190k	1507	11.4	919	74
NSAG w/o calibration	121	189k	1498	11.4	1067	21
NSAG w/ random gate	120	191k	1503	11.4	955	21
NSAG w/o MinFix	88	132k	1412	11.4	42	74

D.3 Rule Granularity: Imperative, Template and First-Order

The rule-granularity study fixes the abstraction level for constraint modelling. We compare imperative rules with lightweight static checks, schema templates with rigid parameter matching, and the first-order formulation the main tables use, whose predicates are state-dependent. Imperative rules are the fastest and blind to non-local dependencies such as the ordering constraints in logistics, and schema templates validate structure yet resist dynamic state transitions. The logic formulation captures the cross-step constraints that template-based rules miss, at a higher checking cost per step. Figure 7 reports the comparison on the same four task-level and three cost metrics.

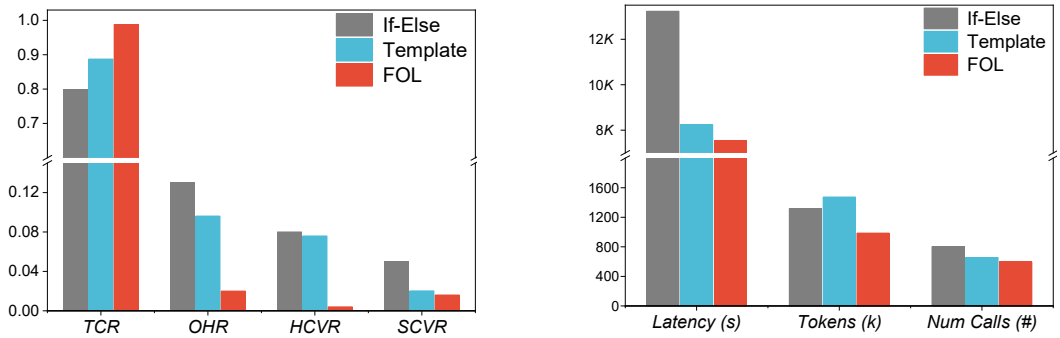


Figure 7: Bars compare imperative, schema-template and first-order logic rules over four task-level metrics and three inference-cost metrics of the same protocol.

D.4 Pruning-Threshold Sensitivity

Table 9 reports the sensitivity of completion and violation rates to the pruning threshold, together with the number of predicate evaluations per generated action that pruning is meant to reduce. A

low threshold keeps more rules active and checks more predicates, and a high threshold drops rules that the task needs, so the table locates the value used for the main tables between the two failure modes.

Table 9: Sensitivity to the pruning threshold τ_{prune}. Checks is the number of predicate evaluations per generated action, which is the quantity pruning is meant to reduce: the guard evaluates the rule set against the proposed action and against the state its replay of the action reaches, so a step costs at most $2|\mathcal{R}|$ of them. Completion is a five-seed mean over the 500 evaluation instances.

τ_{prune}	TCR↑	HCVR↓	Checks
0.05	0.97	0.005	15.2
0.10	0.98	0.004	13.7
0.20	0.99	0.003	11.4
0.30	0.96	0.008	9.1
0.50	0.92	0.019	6.3

E Registry Completeness and Rule Quality

E.1 Missing Hard Preconditions

Table 10 reports the full rule-completeness sweep behind Figure 4. Every removal level carries three solvers on the same instances, and the PlanFail column counts the instances a solver cannot plan at all, which leaves the instances it plans illegally as the remainder of that row's shortfall. NSAG keeps the highest completion rate at every level. The guard on its own ends an episode at the first blocked step, so it completes fewer instances than unguarded generation at every level. It is the repair stage that supplies the whole of the distance from the guard row to NSAG, at every removal level of the sweep, so the five levels and the three solvers of the grid are read the same way.

Table 10: Rule completeness against three solvers. k is the percentage of hard preconditions removed from the registry. PlanFail counts the instances on which a solver reaches no legal plan at all, so it is a share of instances rather than of steps, and the rest of that row's shortfall from full completion is the instances it plans illegally, a split the violation rate alone would hide. The open-vocabulary row reports a goal statement the registry cannot express as a predicate. Completion is averaged over the five seeds of the 500-instance evaluation half, and the guard row is the *NSAG w/o MinFix* configuration of Table 7, which is "the guard without repair" at every level.

k (%)	Method	TCR (↑)	HCVR (↓)	PlanFail (%)
0	LLM only	0.72	0.072	23.5%
0	LLM + guard	0.68	0.004	27.6%
0	NSAG	0.99	0.003	0.3%
10	LLM only	0.72	0.072	23.6%
10	LLM + guard	0.62	0.007	33.2%
10	NSAG	0.90	0.006	4.0%
20	LLM only	0.71	0.072	24.4%
20	LLM + guard	0.55	0.013	39.1%
20	NSAG	0.86	0.012	5.6%
30	LLM only	0.72	0.072	23.8%
30	LLM + guard	0.49	0.019	44.4%
30	NSAG	0.83	0.016	6.9%
50	LLM only	0.71	0.072	24.6%
50	LLM + guard	0.44	0.026	48.7%
50	NSAG	0.81	0.021	7.6%
open-vocab	NSAG	0.88	0.023	4.9%

E.2 Wrong and Induced Rules

Table 11 covers the second kind of registry defect, where an entry is present but wrong or irrelevant. Deleting rules removes the evidence the calibration learns from, since a sensitivity is trained on violations that the predictor and the checker both observe, and an incorrect rule turns a legal step into

a blocked one, which the false-block column measures directly. The rule-induction configuration derives predicates from task text and reaches precision 94.3% and recall 68.4% against the gold registry, which is comparable to the 95.56% precision and 70.96% recall reported for AgentSpec's own rule generation. Where the goal statement is written only in natural language, the registry holds no predicate for it, so the open-vocabulary rows of Table 10 report the case in which the proposal carries the intent in its own words and the registry cannot represent it.

Table 11: Registry entries that are missing or wrong. The five *deletion* levels remove k percent of the hard preconditions, and the three *injection* levels add ϵ percent of incorrect ones, so only the first block shares its levels with Table 10. FBlk is the share of legal steps the guard blocks, and rule induction derives predicates from task text, with precision 94.3% and recall 68.4% against the gold registry that an operator wrote by hand for these ten rules. Completion is averaged over the five seeds of the 500-instance evaluation half, and its deletion block is the NSAG row of Table 10 at the same level. Deletion hides a violation, injection blocks a legal step.

Level (%)	Calibration	HCVR↓	TCR↑	FBlk↓
0	NSAG	0.003	0.99	0.006
0	w/o calibration	0.028	0.96	0.019
10	NSAG	0.006	0.90	0.011
10	w/o calibration	0.041	0.92	0.037
20	NSAG	0.012	0.86	0.020
20	w/o calibration	0.058	0.87	0.062
30	NSAG	0.016	0.83	0.033
30	w/o calibration	0.077	0.82	0.091
50	NSAG	0.021	0.81	0.052
50	w/o calibration	0.104	0.74	0.128
5	NSAG	0.004	0.96	0.031
5	w/o calibration	0.030	0.92	0.052
10	NSAG	0.005	0.92	0.058
10	w/o calibration	0.033	0.86	0.089
20	NSAG	0.007	0.84	0.104
20	w/o calibration	0.038	0.76	0.146

F External and Cross-Backbone Results

F.1 ScienceWorld: Full Comparison

Table 12 reports the full ScienceWorld comparison, including the baselines the main text omits. The six task types behind it are 6 of the 30 the environment ships, so the external validation is a sample of the environment rather than all of it. The Score column is the environment's own rubric normalised by its maximum, Valid is the fraction of executed actions that satisfied their preconditions, and Viol is its complement. The rubric penalises illegal and futile steps, so a normalised score can fall below zero. The boundary is the coverage recorded in Table 2, since a precondition the adaptation did not register is a violation the guard cannot see, and the graded score still rewards progress inside the horizon.

F.2 Cross-Backbone Paired Values

Table 13 carries the exact values behind Figure 5, with the ScienceWorld panel in (a) feeding its panels (a) and (b) and the paired comparison in (b) feeding its panels (c) and (d).

G Scaling and Robustness

G.1 Repair Budget at the Released Constants

Table 14 reports the repair budget consumption with the released constants, including the share of repairs that exhaust the node-expansion limit and the share that stop at the depth limit.

Table 12: External validation on ScienceWorld. Score is the environment's own rubric normalised by its maximum (verified as 100 by replaying the gold path), Valid is the fraction of executed actions that satisfied their preconditions, and Viol is its complement. The rubric penalises illegal and futile steps, so a normalised score can fall below zero. All rows use the same backbone and the same evaluation split, so the comparison is matched on both sides, and the two ablation rows carry the names the main text gives them. Rates are pooled over the 12 episodes of the six registered task types. Table 13 plots six of these rows and carries the same numbers.

Method	Score (↑)	Valid (↑)	Viol (↓)	#Calls	Tokens
Base	-0.07	0.763	0.237	285	165k
CoT	-0.05	0.732	0.268	301	188k
Refine	0.03	0.598	0.402	294	312k
KG	0.06	0.749	0.251	274	224k
RAG	0.11	0.619	0.381	281	188k
SFT	0.04	0.795	0.205	287	168k
NSAG w/o calibration	0.52	0.839	0.161	279	171k
NSAG w/o MinFix	0.39	0.922	0.078	271	168k
NSAG	0.61	0.931	0.069	292	172k
Δ NSAG − Base	+0.68	+0.168	−0.168	+7	+7k
Δ NSAG − uniform	+0.09	+0.092	−0.092	+13	+1k

Table 13: Exact values behind Figure 5. Panel (a) reports the external validation on ScienceWorld, where Score is the environment's own rubric normalised by its maximum and Viol is the fraction of executed actions that failed a precondition, and it feeds panels (a) and (b). The six rows are the ones those panels plot, a subset of the nine Table 12 reports. Panel (b) pairs each backbone with itself, so the comparison is read along a row, Cost is the NSAG-to-base latency ratio, and it feeds panels (c) and (d). The four backbone names are the versions the runs used.

(a) ScienceWorld

Method	Score ↑	Viol ↓	Tokens
Base	-0.07	0.237	165k
Self-refine	0.03	0.402	312k
RAG	0.11	0.381	188k
NSAG w/o calibration	0.52	0.161	171k
NSAG w/o MinFix	0.39	0.078	168k
NSAG	0.61	0.069	172k

(b) Cross-backbone paired comparison

	TCR ↑		HCVR ↓		
Backbone	Base	NSAG	Base	NSAG	Cost ×
Claude-Opus-4.6	0.72	0.99	0.072	0.003	0.96
Qwen3.7-Max	0.68	0.97	0.094	0.008	1.12
Grok-4.7	0.79	0.98	0.061	0.005	0.94
DeepSeek-V4-Flash	0.84	0.99	0.048	0.002	0.91

G.2 Action-Space Scaling

Table 15 enlarges the action space to $|\mathcal{A}| \in \{10, 50, 100, 250, 500\}$ by adding distractor locations and interaction targets, so $|\mathcal{A}|$ counts the action types a configuration registers against the $|\mathcal{A}(s)|$ that Table 14 reports for one state, where a single type can span several executable actions. It reports repair success, node expansions, wall-clock time and the failure rate against the size of the action space. The failure rate rises from 0.032 at $|\mathcal{A}| = 10$ to 0.142 at $|\mathcal{A}| = 500$, and the heuristic ablation changes repair success by 0.061. The large action spaces are the ones where the depth limit and the expansion limit bind, which is where the failure rate comes from.

G.3 Perturbations of the Environment

Table 16 reports the guard and the repair stage under three perturbations of Town-Benchmark. *PO-*α hides the contents of a fraction α of the locations, *Stoch-*β makes each action fail with probability

Table 14: Repair budget consumption with the released constants. Repairs counts blocked steps over the 140 episodes of the repair study, which covers 28 of the evaluation half's instances over five seeds, at 2.2 blocked steps an episode, and Expansions the A* nodes they consumed. A bounded search gives up in one of two ways, and the two counts that follow carry the share of those repairs they account for, which is the failure rate Table 15 reports for the same action space, 10 of 312. The bridge length is what the search returned, not the shortest bridge the graph admits.

Quantity	Value
Actions executable per state $\|\mathcal{A}(s)\|$ (min / max / mean)	3 / 28 / 11.6
Depth limit $d_{\max}$ / expansion limit $N_{\max}$	5 / 2,000
Repairs	312
Mean expansions per repair	47.3
Repairs hitting the depth limit	4 (1.3%)
Repairs hitting the expansion limit	6 (1.9%)
Returned-bridge length (mean)	2.4
Repairs whose shortest bridge exceeds $d_{\max}$	3 (1.0%)

Table 15: Repair under a growing action space. Distractor locations and interaction targets enlarge $|\mathcal{A}|$, the number of action types a configuration registers, and the registry and the instances are unchanged. Fail is the share of repairs that find no bridge inside the budget, so it counts the repairs the search abandons and is the complement of the success column. Latency is the wall clock of one repair, which is its expansions at about 350 ms each. The $|\mathcal{A}| = 10$ row carries the released constants of Table 14, whose $|\mathcal{A}(s)|$ averages 11.6 over the states it visits, so its 16.5 s per repair is what the repair term of Appendix B.6 is built from and the repair columns come from that study. Completion is averaged over the five seeds of the 500-instance evaluation half at every action space, so the rows are comparable, and a repair that hits the cap is counted as a failure.

$\|\mathcal{A}\|$	Repair success	Expansions	Latency (ms)	TCR	HCVR	Fail (%)
10	0.968	47.3	16550	0.99	0.003	3.2
50	0.951	74.2	25970	0.98	0.005	4.9
100	0.929	108.6	38010	0.97	0.007	7.1
250	0.897	178.4	62440	0.94	0.010	10.3
500	0.858	302.7	105950	0.90	0.014	14.2

β, and *Noisy-obs-*γ corrupts observation fields with probability γ. Completion moves from 0.94 at $\alpha = 0.5$ to 0.92 at $\beta = 0.2$, and the hard-constraint violation rate holds at 0.012 under both, against the 0.003 that the same configuration reaches with no perturbation at all. The engine-only control row is the unguarded environment under the same perturbation, so a configuration that matches it has added nothing under that condition. The boundary is that the rollout the guard replays assumes the transition function it uses matches the environment, and a stochastic transition whose parameters the guard does not model breaks that assumption for the whole episode.

H Neuro-Symbolic Foundations

Neuro-symbolic AI combines neural generalization with symbolic rigor, which makes it a natural setting for verifiable decision-making (Ahmed et al., 2023). Neuro-symbolic approaches to LLM reasoning are reviewed by Yang et al. (2025). Search-based methods verify intermediate reasoning steps (Zhu et al., 2024) or refine instruction-following responses (Cheng et al., 2025). Existing frameworks typically integrate formal logic or finite state machines to serve as *shields* that block unsafe actions (Feng et al., 2024; Premsri & Kordjamshidi, 2025; Mu et al., 2024) or as *planners* that generate high-level symbolic skeletons for the neural agent to instantiate (Gupta & Kembhavi, 2023; Surís et al., 2023; Trinh et al., 2024). Rigid shielding can lead to task dead-ends when actions are rejected, while heavy symbolic planning requires exhaustive domain modelling that limits scalability (Ling et al., 2023; Hu et al., 2024). NSAG adds a repair stage to that picture. When the guard blocks an action, MinFix searches the symbolic state graph locally and reconstructs the faulty segment, so the blocked action is repaired and the episode continues from the reconstructed state. The layer sits on top of the backbone and leaves its weights untouched.

Table 16: Robustness under partial observability, stochastic transitions and noisy observations. MFixFail is the share of interventions the search cannot repair within budget. The three engine-only rows are the unguarded environment under the headline setting of each perturbation, so a guarded row that matches one of them has added nothing under that condition, and an unguarded run has no repair to fail. Completion is a five-seed mean over the 500 evaluation instances.

Condition	Guard	TCR↑	HCVR↓	SCVR↓	MFixFail	#Calls
PO-$\alpha = 0.25$	NSAG (learned)	0.97	0.009	0.017	0.024	133
PO-$\alpha = 0.25$	uniform	0.93	0.018	0.024	0.048	152
PO-$\alpha = 0.25$	random gate	0.94	0.028	0.031	0.061	168
PO-$\alpha = 0.50$	NSAG (learned)	0.94	0.012	0.021	0.041	146
PO-$\alpha = 0.50$	uniform	0.89	0.024	0.030	0.076	168
PO-$\alpha = 0.50$	random gate	0.90	0.034	0.035	0.094	189
PO-$\alpha = 0.75$	NSAG (learned)	0.88	0.013	0.031	0.073	174
PO-$\alpha = 0.75$	uniform	0.80	0.031	0.040	0.121	203
PO-$\alpha = 0.75$	random gate	0.82	0.042	0.046	0.148	232
Stoch-$\beta = 0.05$	NSAG (learned)	0.96	0.010	0.018	0.036	140
Stoch-$\beta = 0.05$	uniform	0.92	0.021	0.024	0.064	159
Stoch-$\beta = 0.05$	random gate	0.93	0.028	0.031	0.080	178
Stoch-$\beta = 0.10$	NSAG (learned)	0.94	0.011	0.023	0.058	152
Stoch-$\beta = 0.10$	uniform	0.88	0.026	0.031	0.092	176
Stoch-$\beta = 0.10$	random gate	0.90	0.036	0.037	0.113	199
Stoch-$\beta = 0.20$	NSAG (learned)	0.92	0.012	0.028	0.094	178
Stoch-$\beta = 0.20$	uniform	0.84	0.034	0.041	0.146	209
Stoch-$\beta = 0.20$	random gate	0.86	0.044	0.049	0.182	241
Noisy-obs-$\gamma = 0.10$	NSAG (learned)	0.96	0.009	0.019	0.029	138
Noisy-obs-$\gamma = 0.10$	uniform	0.90	0.019	0.027	0.053	157
Noisy-obs-$\gamma = 0.10$	random gate	0.92	0.029	0.033	0.068	174
Noisy-obs-$\gamma = 0.25$	NSAG (learned)	0.90	0.011	0.026	0.051	156
Noisy-obs-$\gamma = 0.25$	uniform	0.84	0.027	0.035	0.088	181
Noisy-obs-$\gamma = 0.25$	random gate	0.85	0.038	0.042	0.106	206
Engine-only control	unguarded, PO-$\alpha = 0.50$	0.66	0.066	0.029	n/a	116
Engine-only control	unguarded, Stoch-$\beta = 0.20$	0.52	0.064	0.028	n/a	142
Engine-only control	unguarded, Noisy-obs-$\gamma = 0.25$	0.61	0.067	0.029	n/a	129

I Boundaries of the Reported Claims

The violation reductions we report are empirical. The intervention decision is the learned threshold of Equation 7 and no separate check overrides it, so a hard rule that the predictor prunes from the active set contributes nothing to E_t, and a retained rule whose energy stays below the tolerance γ does not raise q_t above $1/2$. An action can therefore be released while one of its registered hard rules is false, and the measured violation rates bound how often that happens on the evaluated instances, and they stop short of proving that every released step satisfies the registry. Three further conditions bound where the measurement applies. First, the environment assumes a fully observed deterministic transition system, and Appendix G perturbs that assumption without removing it, so a stochastic transition whose parameters the guard does not model lies outside what the transition function describes. Second, the calibration is fitted on the same five domains and the same benchmark it is evaluated on, so a domain with no fitting instance is not covered. Third, the rules are registered by an operator, and the induction configuration of Appendix E reaches precision 94.3% and recall 68.4%, which bounds how much of the registry can be obtained automatically.