



Reliable Decision Loops in Socio-Technical Collective Adaptive Systems

Marino Miculan^(✉)

University of Udine, Udine, Italy
`marino.miculan@uniud.it`

Abstract. We propose a formal framework for reasoning about *decision loops* in *socio-technical Collective Adaptive Systems*, i.e. CAS integrating heterogeneous decision-making entities such as software controllers, human operators, and generative AI modules. Modelling these as *decision operators* in feedback structures, we analyse reliability via correlated failure events rather than isolated errors, showing that adding human supervision or AI support does not automatically improve reliability: the effect depends on the joint failure structure of the components involved. We then introduce a *guarded architectural pattern* that decouples decision generation from verification: uncertain generators propose candidate actions while a separate layer enforces admissibility constraints before execution. We establish safety and monotonicity properties, illustrate the framework on a quantitative incident-response scenario, and report a preliminary pilot study on five state-of-the-art LLMs measuring marginal and joint failure probabilities; the measurements are consistent with the predicted central role of joint failure and show that failure diversity must be measured rather than assumed from vendor labels. The framework unifies rigorous CAS engineering, Human-in-the-Loop control, runtime assurance, and LLM safety in a single system-level perspective.

Keywords: Collective Adaptive Systems · Human-in-the-Loop · Large Language Models · Reliability · Runtime Assurance

1 Introduction

The rigorous engineering of Collective Adaptive Systems (CAS) has long relied on formal modelling, quantitative reasoning, and principled software engineering methods to analyse and guarantee properties such as safety, reliability, and predictability despite decentralisation, openness, and adaptation [8, 9, 24]. This perspective remains central, but the systems to which it must now be applied increasingly combine conventional software controllers with human decision-makers, machine learning components, and generative AI modules, all shaping system behaviour through distributed feedback: software components execute local control logic, humans supervise or override critical actions, and AI systems produce recommendations, plans, or directly generated decision proposals.

The CAS paradigm provides a natural foundation for modelling such systems. Architectures where LLM-based agents coordinate with each other and with human operators exhibit its defining characteristics: many interacting entities in open, dynamic environments, global behaviour emerging from local interactions, and runtime adaptation. Moreover, the collective phenomena arising in agentic AI systems—cascading failures, emergent coordination patterns, unexpected compositional behaviours—cannot be understood by analysing individual components in isolation [13]. This paper therefore treats AI-augmented multi-agent systems as CAS, bringing the discipline of their rigorous engineering to bear on the reliability challenges they raise.

This evolution raises a methodological difficulty. Much of formal reasoning in software engineering presupposes that the relevant decision-making entities admit sufficiently explicit and stable semantics to support specification, verification, and compositional analysis. In contemporary CAS, this assumption is not always justified: human decision processes are shaped by workload, situational awareness, trust calibration, and organisational context [15], while generative AI components produce outputs through probabilistic inference and may exhibit hallucinations, context sensitivity, and deviations from expected behaviour [13]. Neither can be treated as a fully specified controller in the traditional sense.

In this paper, we use the term *socio-technical CAS* to denote Collective Adaptive Systems whose evolution depends on the coupled interaction of computational, human, and organisational decision processes. In such systems, software controllers, monitoring and verification components, AI-based modules, human operators, supervisory roles, and operational procedures together determine the admissibility and execution of actions. System-level properties such as safety, reliability, and resilience therefore emerge not from isolated components alone, but from the structure of the decision processes through which observations are interpreted, actions are proposed, and execution is constrained.

This observation leads to the following question: *how can we reason rigorously about reliability in CAS when some of the decision components are uncertain, adaptive, or only partially specified?* In these settings, the object of analysis must shift from individual components to the architecture in which they participate.

To address this challenge, we propose a formal perspective based on the notion of *decision loops*. A decision loop is a feedback structure in which a decision maker observes system state, produces a decision, and thereby influences subsequent system behaviour. This abstraction allows humans, conventional controllers, and LLM-based modules to be modelled uniformly as *decision operators* embedded in the same system architecture. Such a perspective shifts the analysis from the correctness of isolated components to the organisation of the loops through which decisions are generated, verified, and executed.

Building on this abstraction, we show that reliability in socio-technical CAS depends on the correlation between decision errors. In particular, simply adding human supervision or additional AI components does not automatically improve system reliability; improvements arise only when the joint failure probability of the relevant decision components is sufficiently low. Motivated by this observation, we introduce an architectural principle based on *decoupling decision gener-*

ation from decision verification. Under this principle, adaptive decision generators, including humans and LLMs, produce candidate decisions, while a separate verification layer enforces system constraints before actions are executed.

The main contributions of this work are:

1. a formal model of decision components embedded in CAS (Sect. 3);
2. a reliability analysis of socio-technical decision loops based on probabilistic error correlation (Sect. 4);
3. a unified abstraction covering Human-in-the-Loop and LLM-in-the-Loop architectures (Sect. 5);
4. an architectural pattern for the integration of uncertain decision components in CAS, with safety, monotonicity, and cumulative reliability guarantees (Sect. 6), illustrated on an incident-response scenario (Sect. 7.1);
5. a preliminary empirical pilot study on five state-of-the-art LLMs (three Claude and two GPT models) that measures marginal and joint failure rates and supports the analytical predictions (Sect. 7.2).

Related work is discussed in Sect. 8, and conclusions and future directions in Sect. 9. The formal results are deliberately kept simple; the main contribution lies in the unifying framework that brings humans, LLMs, and conventional controllers under a common decision-loop abstraction and derives concrete architectural recommendations from simple probabilistic principles.

2 Background

Collective Adaptive Systems (CAS) are systems composed of many cooperating entities that adapt at runtime to dynamic and open environments. The REoCAS tradition has emphasised the need for theories, methods, and tools for their modelling, analysis, programming, and operation [8, 9, 17, 21, 24].

A key feature of CAS is that global properties emerge from local interactions: reliability cannot be inferred solely from the correctness of individual components, but requires reasoning about interaction structures, environmental uncertainty, and the propagation of local decisions through collective dynamics.

A rich body of formalisms has been developed to model CAS at the agent level: SCEL [7] introduces ensembles of components with knowledge bases, policies, and attribute-based interaction; AbC [1] makes attribute-based communication a first-class primitive; AbU [16] extends it with distributed memory updates; CARMA [3] provides quantitative, Markovian semantics for resource-sharing collectives. The model introduced in Sect. 3 sits at a higher level of abstraction: it treats each decision policy as a stochastic operator, irrespective of how that policy is concretely expressed in any of these calculi, and instantiating our results on top of them (e.g. identifying the operator with a policy in SCEL or a process in AbC) is a natural integration path.

Human-in-the-Loop Systems include human operators that supervise, validate, or override automated choices [11]. Classical work on supervisory control shows that the human cannot be idealised as a perfect oracle: performance depends on

trust calibration, workload, interface design, and the degree of delegated autonomy; moreover, humans may inherit framing biases from automated suggestions or become less vigilant when automation appears reliable [15, 19]. In rigorous engineering terms, they are uncertain decision components whose behaviour depends on system design and operating context.

Learning-Enabled and Generative AI Components, such as LLMs, are increasingly used as planners, coordinators, explainers, and decision-support modules. Their behaviour depends on prompts, retrieved context, hidden internal representations, and probabilistic decoding, and is therefore not fully amenable to conventional specification and verification [4, 22]. Runtime assurance has emerged as the standard architectural response for learning-enabled controllers [5, 6], and for LLMs in particular monitoring, falsification, and constraints are essential to mitigate hallucinations [13]. Hence, LLMs should not be assumed to satisfy executable system constraints unless embedded in a suitable assurance architecture.

3 A Formal Model of Decision Components in CAS

In this work, we model a CAS C as a tuple $C = (A, E, S, D, \alpha, \delta)$, where:

- $A = \{a_1, \dots, a_n\}$ is a finite set of *entities*, i.e. the loci of the system (services, resources, physical or organisational units) whose behaviour is governed and on which decisions take effect;
- E is a set of *environmental inputs*, representing exogenous signals, sensor readings, or contextual data that are not controlled by any decision component (e.g. physical measurements, network events, external user requests);
- S is the *global state space*, i.e. the set of all possible configurations of the system; a state $s \in S$ encodes the relevant variables of all entities and their shared resources at a given instant;
- $D = \{d_1, \dots, d_k\}$ is a finite set of *decision components*;
- $\alpha : A \rightarrow \mathcal{P}(D)$ is the *governance assignment*, mapping each entity to the subset of decision components that govern its behaviour;
- $\delta : S \times U_1 \times \dots \times U_k \times E \rightarrow S$ is the *global transition function*, determining the next state from the current state, the actions emitted by all decision components, and the current environmental input.

Each decision component d_i has an associated *observation space* O_i and an *action space* U_i . The observation space O_i is the projection of the global state that is visible to d_i , via an *observation map* $\pi_i : S \rightarrow O_i$; the action space U_i is the set of decisions or commands that d_i can issue. The type of d_i is therefore

$$d_i : O_i \rightarrow \mathcal{D}(U_i), \quad i \in \{1, \dots, k\}$$

where $\mathcal{D}(U_i)$ denotes the set of probability distributions over U_i . Writing $P_i(u \mid o)$ for the probability assigned to action $u \in U_i$ given observation $o \in O_i$, this captures uncertainty uniformly: a deterministic controller corresponds to a degenerate distribution (a single action with probability 1), while human operators

and LLM-based modules correspond to genuinely stochastic distributions whose support and shape depend on context.

Throughout the paper we write $u \sim d_i(o)$ as a shorthand for “ u is sampled from the distribution $d_i(o) \in \mathcal{D}(U_i)$ ”; for a deterministic controller $d_i(o)$ is a point mass and sampling reduces to function application.

The governance assignment α makes explicit which decision components are responsible for which entities. We deliberately reserve the term *agent*, and the associated notion of *agency*, for the decision components in D : it is these that observe, deliberate, and issue actions, whereas an entity $a_j \in A$ is the governed locus at which those actions take effect through the global transition δ . We require $\alpha(a_j) \neq \emptyset$ for all $a_j \in A$, that is, each entity is governed by at least one decision component. When $|\alpha(a_j)| = 1$ the entity has a single decision authority; when $|\alpha(a_j)| > 1$ multiple components jointly govern its behaviour, and the reliability analysis of Sect. 4 applies directly. Notice also that decision components can be shared among different entities.

The global evolution at time t is

$$s_{t+1} = \delta(s_t, u_1, \dots, u_k, e_t),$$

where $u_i \sim d_i(\pi_i(s_t))$, i.e. u_i is sampled from $P_i(\cdot \mid \pi_i(s_t))$, and $e_t \in E$ is the current environmental input.

Decision Loops. For each decision component d_i ($i = 1, \dots, k$), the *decision loop* of d_i is the feedback structure in which d_i observes the current system state, produces a decision, and thereby contributes to determining the successor state:

$$s_t \xrightarrow{\pi_i} o_t^{(i)} \xrightarrow{d_i} u_t^{(i)}.$$

The k individual loops operate concurrently: at every step, each component d_i observes $o_t^{(i)} = \pi_i(s_t)$ and emits an action $u_t^{(i)} \sim d_i(o_t^{(i)})$. The global transition function then combines all contributions together with the environmental input:

$$s_{t+1} = \delta(s_t, u_t^{(1)}, \dots, u_t^{(k)}, e_t),$$

and the successor state s_{t+1} becomes the input to the next iteration of every loop. Thus, the *global decision loop* of the CAS is the composite feedback cycle

$$S \xrightarrow{\langle id, \pi_1, \dots, \pi_k \rangle} S \times O_1 \times \dots \times O_k \xrightarrow{id \times d_1 \times \dots \times d_k} S \times \mathcal{D}(U_1) \times \dots \times \mathcal{D}(U_k) \xrightarrow{\delta(\cdot, e_t)} S$$

where in the last step we feed to the global transition function the samples from the distributions, and the current environmental input.

This abstraction uniformly covers conventional controllers, human supervisors, and LLM-based modules. What changes across these cases is not the existence of the loop, but the semantics of the decision operator and the type of uncertainty involved: deterministic controllers correspond to point-mass distributions, while human and LLM-based components correspond to genuinely stochastic distributions.

The decision-loop structure admits a natural coalgebraic formulation that makes the feedback and probabilistic semantics fully explicit; we leave its development to an extended version of this paper.

4 Reliability of Socio-Technical Decision Loops

We first formalise when individual decisions are correct. Let $Adm(s) \subseteq U_1 \times \dots \times U_k$ be the set of *admissible joint decisions* in state s . For a component d_i , let $Adm_i(s)$ denote its *marginal admissibility set*, i.e. the projection of $Adm(s)$ onto the i -th coordinate:

$$Adm_i(s) \triangleq \{u_i \in U_i \mid \exists (u_1, \dots, u_k) \in Adm(s) \text{ with } i\text{-th component } u_i\}.$$

Intuitively, $Adm_i(s)$ contains the actions of d_i that are compatible with at least one admissible joint decision. We define the event

$$correct_i(s) \triangleq \{u_i \sim d_i(\pi_i(s)) \mid u_i \in Adm_i(s)\},$$

i.e. the event that the action sampled from d_i belongs to the marginal admissibility set. The *state-dependent reliability function* of d_i is then

$$R_i(s) = P(correct_i(s)) \in [0, 1].$$

This state dependence is essential in socio-technical settings, where reliability may vary because of ambiguity, incomplete observability, workload, distributional shift, or prompt sensitivity.

Remark 1 (Feedback amplification). Decision loops introduce feedback: an incorrect decision at time t degrades s_{t+1} in ways that may make subsequent observations less informative, and in adaptive components the operator itself is time-dependent ($d_{i,t}$), so $R_i(s_t)$ co-evolves with the system. An incorrect AI diagnosis can bias a human operator, producing reinforcing errors; this amplification motivates architectural mechanisms that limit error propagation.

For simplicity, we assume that all k decision components participate in every global transition. (Partial activation can be modelled via a state-dependent function $active : S \rightarrow \mathcal{P}(D)$, but is not needed here.) Under this assumption, the system-level reliability at state s is

$$R_{\text{sys}}(s) = P\left(\bigcap_{i=1}^k correct_i(s)\right).$$

If the decision components were independent, one could write $R_{\text{sys}}(s) = \prod_{i=1}^k R_i(s)$. In socio-technical and AI-augmented systems, however, this assumption is generally false because decision errors may be correlated.

Consider two decision components d_1 and d_2 that jointly protect a transition, for example by redundancy, supervision, or cross-checking. Let $\epsilon_i(s)$ be the event that d_i fails to reject or avoid an unsafe decision in state s . Define

$$p_i = P(\epsilon_i(s)) \quad \text{and} \quad q = P(\epsilon_1(s) \cap \epsilon_2(s)).$$

Assume that the system is unsafe only if *both* components fail; then $R_{\text{pair}}(s) = 1 - q$: what matters is not only the marginal failure of each component, but the probability of their *simultaneous* failure.

Proposition 1. *Let d_1 and d_2 be two protective decision components in state s , with failure probabilities p_1 and p_2 , and joint failure probability q . Then:*

- (1) $R_{\text{pair}}(s) = 1 - q$;
- (2) $R_{\text{pair}}(s) \geq 1 - \min(p_1, p_2)$;
- (3) if $\epsilon_1(s)$ and $\epsilon_2(s)$ are independent, then $R_{\text{pair}}(s) = 1 - p_1 p_2$;
- (4) for fixed p_1, p_2 , $R_{\text{pair}}(s)$ is maximised when the overlap q is minimised.

The proposition formalises a simple but important engineering point: redundancy helps only when it reduces *joint failure*. Adding a human supervisor or an LLM-based checker improves reliability only if their failures are not strongly aligned with those of the primary component. For instance, when a human supervisor validates the proposals of an automated controller, the reliability gain depends not only on human skill but also on the correlation between machine and human errors, which is affected by automation bias and interface design [15, 19].

5 LLM-in-the-Loop Decision Processes

We now specialise the framework of Sect. 3 to LLM-based decision components, identifying the sources of uncertainty that distinguish them from conventional controllers and introducing a verification mechanism that constrains their outputs—and, by the same pattern, human outputs—to admissible actions.

We model an LLM-based decision component as

$$d_{\text{LLM}}(o) \sim P_{\theta}(u \mid o, c),$$

where o is the observable state, c is the prompt and interaction context, and θ denotes model parameters and inference configuration.

This representation highlights three sources of uncertainty:

- *epistemic*, from incomplete or non-representative training;
- *contextual*, due to prompt ambiguity or retrieval defects;
- *generative*, from probabilistic decoding.

Writing $R_{\text{LLM}}(s) \triangleq R_i(s)$ for any LLM-based d_i , these uncertainties imply that R_{LLM} is

- *state-sensitive*: it degrades under distributional shift;
- *prompt-dependent*: semantically equivalent contexts c, c' can give different distributions;
- *high-variance*: repeated sampling at fixed (s, c) yields qualitatively different decisions whenever $0 < R_{\text{LLM}}(s) < 1$.

Moreover, since LLM failure mechanisms (token-level prediction errors, context-encoding artefacts) differ structurally from those of conventional controllers (specification gaps, numerical instability), the joint failure $q = P(\epsilon_{\text{LLM}}(s) \cap \epsilon_{\text{ctrl}}(s))$ is expected to lie well below the bound $\min(p_{\text{LLM}}, p_{\text{ctrl}})$ of Proposition 1(2); we revisit this in Sect. 7.2.

To constrain LLM outputs, we introduce a verifier

$$V : U \times S \rightarrow \{true, false\},$$

which checks whether a candidate decision is admissible in the current state. The verified decision function is

$$d^*(s) = \begin{cases} d_{\text{LLM}}(s) & \text{if } V(d_{\text{LLM}}(s), s) = true, \\ fallback(s) & \text{otherwise.} \end{cases}$$

The fallback may be a rule-based controller, a conservative safe action, or escalation to a human operator. This pattern aligns with runtime-assurance approaches for learning-enabled systems, where an advanced but uncertain component is bounded by a trusted safety mechanism [4–6].

The same pattern applies verbatim to human-generated decisions, replacing d_{LLM} with a human component d_H . This common formulation supports a unified socio-technical perspective: humans and LLMs differ in semantics, accountability, and causal structure, yet both can be modelled as uncertain decision generators constrained by deterministic verification and runtime assurance layers.

6 Guarded Decision Architectures

Since system-level reliability depends on joint failure probabilities (Sect. 4) and LLM-based components introduce irreducible uncertainty (Sect. 5), a natural question arises: how should a CAS be architected so that uncertain decision components—whether human, LLM-based, or adaptive—can be integrated without compromising system-level safety?

The central architectural principle we propose is: *Decision generation and decision verification should be decoupled.*

This principle prescribes a separation between three roles:

- a *decision generator* d_{gen} , which proposes candidate actions;
- a *decision verifier* V , which enforces explicit admissibility constraints;
- an *execution layer*, which commits only verified actions.

The generator, possibly adaptive or probabilistic, may be a conventional controller, a human operator, or an LLM; the verifier may be based on contracts, temporal rules, policy constraints, runtime monitors, safety envelopes, etc.

Formally, given a decision component d_i acting as generator in state s_t , the *guarded decision loop* extends the decision loop of Sect. 3 by interposing a verification step between generation and execution:

$$s_t \xrightarrow{\pi_i} o_t \xrightarrow{d_i} \hat{u} \xrightarrow{V(\cdot, s_t)} u,$$

where $\hat{u} \sim d_i(o_t)$ is the candidate action proposed by the generator, and the executed action is

$$u = \begin{cases} \hat{u} & \text{if } V(\hat{u}, s_t) = true, \\ u_f & \text{otherwise,} \end{cases}$$

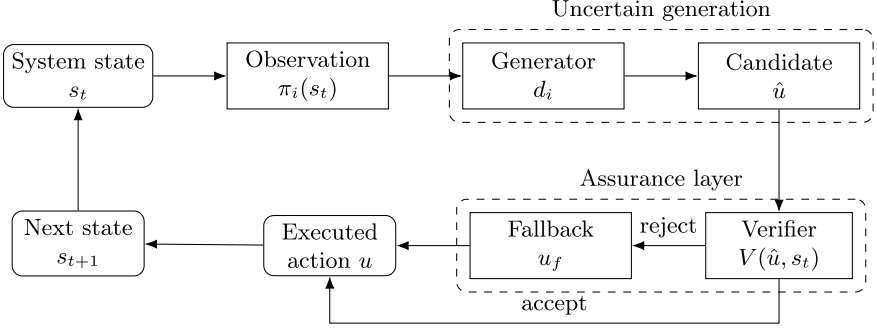


Fig. 1. Guarded decision loop architecture for CAS. Uncertain decision generators (human operators, LLMs, or adaptive controllers) propose candidate decisions that are validated by a verification layer before execution. The architecture ensures that only admissible actions are executed, independently of generator behaviour.

with $u_f \in \text{Adm}(s_t)$ a safe fallback action. The successor state is then determined by the global transition as in the unguarded case: $s_{t+1} = \delta(s_t, u, e_t)$, and s_{t+1} becomes the input to the next iteration of the loop.

Figure 1 illustrates this architecture: candidate decisions from an uncertain generator are evaluated by a verification layer enforcing admissibility constraints, and rejected decisions are replaced by safe fallback actions.

We now establish two properties of this architecture: *safety* and *monotonicity*. Both hold independently of the behaviour of the generator; notably, monotonicity holds independently of the quality of the verifier as well.

Proposition 2 (Safety and monotonicity of guarded execution). *Let $V : U \times S \rightarrow \{\text{true}, \text{false}\}$ be a verifier, and let $u_f \in \text{Adm}(s)$ for all s . Then:*

- (1) *Safety: if V is sound ($V(u, s) = \text{true} \Rightarrow u \in \text{Adm}(s)$) then the executed action satisfies $u \in \text{Adm}(s)$, regardless of the candidate $\hat{u}$ given by the generator;*
- (2) *Monotonicity: for an arbitrary (possibly unsound) V , let $P_{\text{safe}}^{\text{gen}} = P(\hat{u} \in \text{Adm}(s))$ and $P_{\text{safe}}^{\text{ver}}$ denote the probability that the executed action is admissible without and with verification, respectively. Then we have $P_{\text{safe}}^{\text{ver}} \geq P_{\text{safe}}^{\text{gen}}$.*

Proof. (1) If $V(\hat{u}, s) = \text{true}$ then $u = \hat{u} \in \text{Adm}(s)$ by soundness; if $V(\hat{u}, s) = \text{false}$ then $u = u_f \in \text{Adm}(s)$ by assumption. (2) The executed action is admissible iff either V accepts and $\hat{u} \in \text{Adm}(s)$, or V rejects (in which case $u = u_f \in \text{Adm}(s)$). Hence $P_{\text{safe}}^{\text{ver}} = P(\hat{u} \in \text{Adm}(s) \wedge V \text{ accepts}) + P(V \text{ rejects}) \geq P(\hat{u} \in \text{Adm}(s) \wedge V \text{ accepts}) + P(\hat{u} \in \text{Adm}(s) \wedge V \text{ rejects}) = P_{\text{safe}}^{\text{gen}}$. Note that only fallback admissibility is used. $\square$

Remark 2 (Standing assumptions on verifier and fallback). The guarantees above rest on three explicit assumptions. (A1) *Verifier soundness:* $V(u, s) = \text{true} \Rightarrow u \in \text{Adm}(s)$; the verifier never certifies an inadmissible action, though it may be *incomplete*, i.e. reject admissible ones. (A2) *Fallback admissibility:*

$u_f \in \text{Adm}(s)$ for every state s . (A3) *Fallback availability*: such a u_f can actually be computed in the current state. (A1) makes the safety of Proposition 2(1) *independent* of the generator, and is realistic when the verifier is a formally specified, mechanistically simple checker (contracts, monitors, policy rules). Completeness is deliberately *not* assumed: an incomplete verifier may reject admissible candidates, which costs decision quality but not safety. Absolute safety does hinge on soundness holding *in practice*: specification gaps may cause a deployed verifier to accept inadmissible actions, a residual risk that Proposition 1 quantifies through the joint failure q , with the verifier as second protective component. Even then the guarantee degrades gracefully: monotonicity (Proposition 2(2)) requires only (A2), so verification never *decreases* safety, however unreliable the verifier. (A2)–(A3) are thus the load-bearing assumptions: a universally admissible, computable fallback (a conservative safe action, a rule-based controller, or human escalation) may be unavailable or expensive in some domains, and without it both guarantees are lost—verification can then only be used in reject-only mode, blocking execution instead of substituting a safe action. We return to the role and limits of the fallback in Sect. 9.

Remark 3 (Quantifying the verification gain). With a sound verifier and a safe fallback, $P_{\text{safe}}^{\text{ver}} = 1$ and the verification gain $\Delta P_{\text{safe}} = p_{\text{err}}$ equals exactly the generator’s error rate $p_{\text{err}} = P(\hat{u} \notin \text{Adm}(s))$; incompleteness affects only how often admissible proposals are replaced by the fallback, not safety. When soundness fails in practice (Remark 2), $P_{\text{safe}}^{\text{ver}} < 1$ and the analysis of Proposition 1 applies to the residual joint failure.

The single-step guarantees above extend naturally to trajectories. Define the *cumulative reliability* over a trajectory of T steps starting from s_0 as the probability that every executed action is admissible:

$$R_{\text{cum}}^T(s_0) \triangleq P\left(\bigcap_{t=0}^{T-1} \{u_t \in \text{Adm}(s_t)\}\right).$$

The following proposition lifts the single-step view to whole trajectories. Its purpose is to make precise a simple but consequential intuition: *without* architectural safeguards, per-step imperfections accumulate, so an unguarded loop cannot preserve admissibility indefinitely; the guarded loop, by contrast, preserves it by construction at every step. To capture the unguarded case we use the *best-case* per-step system reliability $R_{\text{max}} \triangleq \sup_{s \in S} R_{\text{sys}}(s)$: if not even the most favourable reachable state is perfectly reliable, then cumulative reliability is bounded by a geometrically decaying sequence.

Proposition 3 (Cumulative reliability). *Let $R_{\text{max}} \triangleq \sup_{s \in S} R_{\text{sys}}(s)$. Then:*

- (1) *In the unguarded case, $R_{\text{cum}}^T(s_0) \leq (R_{\text{max}})^T$. In particular, if $R_{\text{max}} < 1$ (no reachable state is perfectly reliable), the cumulative reliability decays exponentially in T .*
- (2) *In the guarded case, under the hypotheses of Proposition 2(1) (sound verifier, safe fallback), $R_{\text{cum}}^T(s_0) = 1$ for all T and all s_0 .*

Proof. (1) Fix the sequence of environmental inputs $e_0, e_1, \dots$ (the argument holds for every such sequence, hence also after averaging over any distribution on them). Since the transition δ is deterministic, conditioning on the realised past actions $u_0, \dots, u_{t-1}$ then fixes the state s_t , and the conditional probability that $u_t \in \text{Adm}(s_t)$ is exactly $R_{\text{sys}}(s_t) \leq R_{\text{max}}$ (the bound holds for every reachable state). By the chain rule, conditioning at each step on the admissible histories and averaging over them, $R_{\text{cum}}^T(s_0) = \prod_{t=0}^{T-1} P(u_t \in \text{Adm}(s_t) \mid u_{<t} \in \text{Adm}) \leq \prod_{t=0}^{T-1} R_{\text{max}} = (R_{\text{max}})^T$. (2) By Proposition 2(1), $u_t \in \text{Adm}(s_t)$ holds deterministically at every step, so the intersection has probability 1. $\square$

This result connects to the feedback-amplification concern of Remark 1: in an unguarded loop, even a small per-step failure probability compounds over time, and state degradation caused by early errors may further reduce reliability at later steps (so the bound $(R_{\text{max}})^T$ is pessimistic but not vacuous). The guarded architecture eliminates this compounding entirely.

By Proposition 1, system-level reliability hinges on the joint failure probability q , which induces two regimes. In a *homogeneous* pair (e.g. two LLM-based components) the failure events largely overlap, so $q \approx p_1$ and the redundancy gain is marginal; in a *diverse* pair (e.g. an LLM generator and a formal policy verifier) the failure mechanisms differ structurally, yielding $q \ll p_1$ and $R_{\text{pair}} = 1 - q \gg 1 - p_1$. This contrast yields a precise design principle: pairing an LLM-based generator with a structurally diverse, formally specified verifier directly reduces q and improves R_{pair} , whereas composing two LLM-based components, or using the same model for both generation and self-verification, offers little benefit regardless of individual accuracy.

7 Illustration and Empirical Evaluation

We exercise the framework through two complementary examples. Section 7.1 develops an *illustrative* incident-response scenario, mapping a realistic socio-technical CAS onto the formal model and the guarded architecture; its numbers instantiate Proposition 1 within literature-reported ranges and are *not* measurements. Section 7.2 then reports an *empirical pilot study* on Blocksworld planning with five LLMs, where the framework’s quantities $(p_i, q, R_{\text{pair}})$ are actually *measured* and compared against those predictions.

7.1 An Illustrative Incident-Response Scenario

We instantiate the framework on a distributed incident-response system managing a large-scale infrastructure platform. We first map the scenario onto the formal model, then apply the guarded architecture and analyse its reliability implications; Fig. 2 sketches the components and their relations.

System Components. The system continuously monitors service latency, error rates, and resource utilisation across the platform. Detected anomalies trigger a

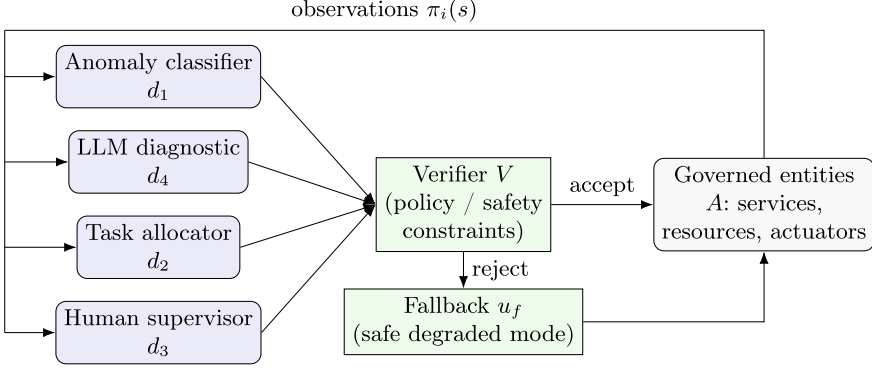


Fig. 2. Component sketch of the incident-response scenario. Four decision components (generators d_1 – d_4 : anomaly classifier, task allocator, human supervisor, LLM diagnostic) propose actions that a single verification layer V checks for admissibility before they take effect on the governed entities A ; rejected proposals are replaced by a safe fallback u_f . Both human and LLM proposals pass through the same verifier.

coordination service that assigns mitigation tasks; automated remediation controllers carry them out; human operators supervise critical interventions; and an LLM-based diagnostic assistant proposes candidate mitigations.

In terms of the formal model, the CAS is $C = (A, E, S, D, \alpha, \delta)$ where the entity set A comprises the monitored services, shared infrastructure resources, and the remediation actuators. The decision component set is $D = \{d_1, d_2, d_3, d_4\}$, corresponding to: (1) automated anomaly classifiers, (2) the collective task allocator, (3) human supervisors, and (4) the LLM-based diagnostic module. The governance assignment α maps each entity to the subset of decision components that govern its behaviour; for instance, the remediation actuators are jointly governed by the task allocator and the human supervisor, while the diagnostic workflow is governed by the LLM module subject to human oversight.

Guarded LLM Integration. The LLM module d_4 receives contextual information describing the detected anomaly—log excerpts, metric snapshots—and proposes candidate mitigation strategies: $\hat{u} \sim d_4(\pi_4(s_t))$. These proposals are evaluated by a verification layer V checking policy compliance, safety constraints, resource availability, and consistency with the current state; rejected candidates trigger a safe degraded mode u_f that preserves service continuity. Crucially, human actions are subject to the same mechanism: potentially disruptive operator actions (service shutdowns, reconfigurations, transfers of control) are validated against the same admissibility constraints before execution, a direct application of the common abstraction of Sect. 5.

Reliability Analysis. The architecture reduces unsafe transitions at two levels. If the policy checker is sound, by Proposition 2(1) every executed action belongs to $\text{Adm}(s)$ regardless of generator behaviour. If soundness fails in practice (Remark

Table 1. Reliability comparison across architectural configurations in the illustrative incident-response scenario. Columns p_1 , p_2 , q are *assumed* inputs (chosen within literature-reported ranges); $R_{\text{pair}} = 1 - q$ is *computed* via Proposition 1(1). These are illustrative values, not measurements.

Configuration	p_1	p_2	q	R_{pair}
Unguarded LLM (routine)	0.12	—	—	0.88
Unguarded LLM (distributional shift)	0.30	—	—	0.70
LLM + LLM self-check (routine)	0.12	0.12	0.10	0.90
LLM + formal verifier	0.12	0.02	0.002	0.998
LLM + LLM (distributional shift)	0.30	0.25	0.22	0.78

2), the checker still acts as a second protective component whose failure modes are structurally different from those of the LLM generator, so the joint failure probability q is expected to be far below the marginals—and, by Proposition 2(2), verification never decreases safety in any case.

Quantitative Example. Recall the quantities of Proposition 1: p_i is the *marginal* failure probability of component d_i (it fails to reject an unsafe decision), q is their *joint* failure probability, and $R_{\text{pair}} = 1 - q$ is the resulting pair reliability.

Configuration (Assumed). We instantiate Proposition 1 with values that, while illustrative, are within the range reported for LLM planning (10–40% failure depending on task complexity [12, 13]) and for industrial formal policy checkers ($\leq 10^{-2}$, dominated by specification gaps [5, 6]). We set $p_{\text{LLM}} = 0.12$ in routine, in-distribution states and $p_{\text{LLM}} = 0.30$ under distributional shift [13], and choose the second component’s marginal p_2 and the joint q per configuration.

Predicted Reliability (Computed). Column R_{pair} of Table 1 is then *derived* from these inputs via Proposition 1(1). A second LLM as self-checker overlaps heavily with the generator’s errors ($q/p_1 = 0.83$, in line with reports that same-model self-checking is unreliable [13]) and gains little ($R_{\text{pair}} = 0.90$ vs. baseline 0.88). A formal policy verifier with mechanistically different failure modes (probabilistic token prediction vs. symbolic rule matching) drives q near zero and R_{pair} near one ($q = 0.002$, $R_{\text{pair}} = 0.998$). Under distributional shift, a homogeneous LLM pair can be *worse* than the unguarded baseline ($R_{\text{pair}} = 0.78$): joint failure approaches the marginal failure of the weaker component, nullifying the redundancy.

The quantitative analysis reinforces the main architectural recommendation: reliable integration of LLM-based components in CAS requires verification layers that are *mechanistically diverse* from the generator. Formally specified policy checkers, contract-based monitors, or rule-based safety envelopes satisfy this requirement; second LLM-based components generally do not.

7.2 An Empirical Pilot Study: Blocksworld Planning

The scenario of Sect. 7.1 is illustrative; we now provide *preliminary empirical evidence* that the framework’s quantities can be measured on real systems and that its central prediction—joint failure q , not marginal accuracy, governs the redundancy gain—is borne out in at least one concrete setting.

Domain. We use *Blocksworld*, a classical planning benchmark in which an agent must rearrange a set of labelled blocks from an initial configuration into a goal configuration using four actions (*pickup*, *putdown*, *stack*, *unstack*) [20]; it is a standard testbed for assessing the planning abilities of LLMs [23]. A decision component here is an LLM asked to output a plan; the “unsafe/failed” event is producing an invalid or non-goal-reaching plan.

Setup. The benchmark comprises $N = 30$ solvable tasks: 10 *easy* (3 blocks, optimal plans of length ≤ 6), 10 *medium* (4–5 blocks, ≤ 14), and 10 *hard* (6–8 blocks, up to 24). We query five LLMs (Claude Haiku 4.5, Sonnet 4.6, Opus 4.7, and OpenAI GPT-4o-mini and GPT-4o) with identical prompts, one independent sample per task, temperature 0 where supported, and no chain-of-thought, few-shot examples, retrieval, or tool use. A deterministic validator simulates each returned plan on the initial state, checking well-formedness, preconditions at every step, and goal satisfaction; it thus implements a sound and complete formal verifier in the sense of Proposition 2. All task files, model responses, validator code, and the aggregation script of this experiment are publicly available [14].

Results. Marginal failure rates span $p \in [0.000, 0.767]$, with a clear capability ordering within each vendor: 0.000 for Opus 4.7; 0.133 for Sonnet 4.6; 0.333 for both Haiku 4.5 and GPT-4o; 0.767 for GPT-4o-mini. Table 2 reports the pairwise joint-failure measurements for the most informative configurations; a fifth informative LLM pair, Haiku + GPT-4o, has $q = 0.167$ against $q_{\text{indep}} = 0.111$ (pairs involving Opus all have $q = 0$, since Opus solves every task). Its columns instantiate the quantities of Proposition 1: p_a, p_b are the two marginal failure rates; q the measured joint failure; $q_{\text{indep}} = p_a p_b$ the value q would take under independence of p_a, p_b ; $q/\min p \in [0, 1]$ a normalised correlation index (0 = no shared failures, 1 = every failure of the stronger component is shared by the weaker); and $R_{\text{pair}} = 1 - q$.

Despite the obvious limitations of this simple pilot (single domain, small sample, in-distribution), three findings emerge:

1. Measured q exceeds the independence baseline in four of the five LLM-pair configurations, confirming that LLM failures are typically correlated and that the joint-failure analysis of Proposition 1 is the right object of study.
2. Cross-vendor pairing is not automatically diverse: Sonnet + GPT-4o achieves $q = 0$, whereas for Sonnet + GPT-4o-mini every Sonnet failure is shared by GPT-4o-mini ($q/\min p = 1.00$). Diversity, in our framework’s sense, is a property of the joint failure structure, not of the vendor label.
3. Measured $q/\min p$ ratios range from 0.50 to 1.00, bracketing the illustrative 0.83 figure of Table 1: the qualitative conclusion of Proposition 1 stands, but the magnitude of the redundancy gain is pair-specific.

Table 2. Measured pairwise joint failure on the Blocksworld pilot. $q_{\text{indep}} = p_a p_b$ is the value q would take under independence; $R_{\text{pair}} = 1 - q$.

Config.	Pair	p_a	p_b	q	q_{indep}	$q/\min p$	R_{pair}
<i>Same vendor (homogeneous LLM pair)</i>							
Anthropic	Haiku + Sonnet	0.333	0.133	0.067	0.044	0.50	0.933
OpenAI	GPT-4o-mini + GPT-4o	0.767	0.333	0.300	0.256	0.90	0.700
<i>Cross vendor (heterogeneous LLM pair)</i>							
A + O (best)	Sonnet + GPT-4o	0.133	0.333	0.000	0.044	0.00	1.000
A + O (worst)	Sonnet + GPT-4o-mini	0.133	0.767	0.133	0.102	1.00	0.867
<i>Generator + sound formal verifier (diverse)</i>							
LLM + checker	Haiku + plan checker	0.333	0.000	0.000	0.000	—	1.000

These results are *preliminary evidence* only: they do not warrant quantitative claims, and the perfect complementarity of Sonnet + GPT-4o is a benchmark artefact. What the pilot does establish is that the framework’s quantities are operationally measurable; a broader evaluation is left to future work.

8 Related Work

There is a long tradition of modelling, abstraction, and verification techniques for adaptive ensembles and collective systems; see e.g. [8, 9, 17, 21, 24]. This line of research established the importance of rigorous engineering for dynamic and decentralised systems, but generally did not address the integration of generative AI within a unified decision-loop abstraction, nor the role of error correlation in system-level reliability; the present paper contributes to this agenda.

Classical work on human oversight [15, 19] shows that human supervision is context-sensitive and systematically biased, supporting our treatment of humans as uncertain decision generators correlated with automated components, rather than as infallible fallback oracles.

Urban and Miné [22] survey formal methods for ML, and runtime-assurance architectures have been proposed to bound the behaviour of learning-enabled controllers [4–6]. The guarded decision-loop architecture inherits the spirit of the *Simplex* pattern of Sha [18] and its probabilistic and learning-oriented descendants [2, 5, 6]: a complex but uncertain decision generator is bounded by a simpler, trustworthy verifier with safe-fallback behaviour. Relative to that tradition, we (i) instantiate the pattern for socio-technical CAS, treating humans and LLM-based components uniformly as stochastic decision generators; (ii) make the joint failure probability q the primary object of analysis, deriving guarantees sensitive to the mechanistic diversity between generator and verifier; and (iii) embed the pattern in the CAS feedback structure, enabling reasoning about cumulative reliability along trajectories (Proposition 3).

Recent work has proposed specification and enforcement mechanisms specifically targeting LLM-based agents. Ferrando [10] argue for a symbiotic integration of Runtime Verification and LLM agents, where RV acts as a guardrail for

LLM autonomy while LLMs can assist in the acquisition of RV specifications. Complementarily, work on bridging LLM planning and formal methods [12] proposes translating natural-language action plans into Kripke structures for LTL verification. Compared to these agent-level approaches, our contribution is at the system level: we embed verification not only as a constraint on individual LLM outputs but as an architectural layer whose effect on system reliability is analysed through joint failure probability across all decision components in a CAS.

Huang et al. [13] identify monitoring and verification as essential to constrain generative behaviour. We differ in embedding these concerns into a CAS-oriented formal model that quantitatively relates verification architecture choices to system-level reliability guarantees.

9 Conclusion

We have proposed a formal framework for reasoning about reliable decision loops in socio-technical Collective Adaptive Systems that integrate conventional controllers, human operators, and LLM-based components, modelled uniformly as stochastic operators in feedback loops. We have shown that system reliability depends critically on the joint failure probability of the components involved, not merely on their individual accuracy; the guarded decision loop of Sect. 6 guarantees that only admissible actions are executed, independently of generator behaviour. The pilot study of Sect. 7.2 provides a preliminary empirical sanity check on these claims.

Four implications follow. (i) *Architectural diversity* matters at least as much as component accuracy: pairing generators with verifiers with structurally different failure modes is the primary mechanism for reducing q . (ii) Integrating generative AI into CAS should focus less on raw model performance and more on *robust system architectures*: even an unreliable generator is safe under sound verification. (iii) The same pattern applies to human and AI components, yielding a *unified socio-technical engineering* approach. (iv) Emphasis shifts from component-centric to *loop-centric assurance*: what matters is whether the loop as a whole preserves admissibility.

Future Work. Scaling the empirical evaluation to larger samples, richer domains, and out-of-distribution regimes is the natural next step. The reliability functions $R_i(s)$ are taken as known in the analysis: estimating them online may require runtime monitoring, calibration, or statistical testing. The model also abstracts from strategic or adversarial behaviour (e.g. adversarial prompt injection), whose treatment would require game-theoretic models.

Reviewer-suggested directions sharpen this agenda. (i) *Stochastic dynamics*: allowing the global transition δ to be a Markov kernel, so that even admissible actions may occasionally fail, would extend the analysis to environmental non-determinism. (ii) *The role of the fallback*: what does an uncertain generator buy over the safe fallback itself (Remark 2), and how to reason when no fallback exists and verification can only reject? (iii) *Multi-level loops*: real CAS comprise

decision loops at several abstraction levels, with humans and LLMs variously *in*, *on*, *out of*—or even *being*—the loop, calling for models of cross-level interaction. (iv) *Deliberative governance*: components jointly governing an entity could form a deliberative assembly with explicit aggregation semantics.

References

1. Abd Alrahman, Y., De Nicola, R., Loreti, M., Tiezzi, F., Vigo, R.: A calculus for attribute-based communication. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC), pp. 1840–1845 (2015). <https://doi.org/10.1145/2695664.2695668>
2. Bak, S., Johnson, T.T.: Run-time assurance for learning-enabled systems. In: Run-time Verification. LNCS, vol. 12399, pp. 306–319. Springer (2020)
3. Bortolussi, L., et al.: CARMA: collective adaptive resource-sharing markovian agents. Electron. Proc. Theor. Comput. Sci. **194**, 16–31 (2015)
4. Burton, S., Herd, B.: Addressing uncertainty in the safety assurance of machine-learning. Front. Comput. Sci. **5**, 1132580 (2023)
5. Chen, S., Sun, Y., Li, D., Wang, Q., Hao, Q., Sifakis, J.: Runtime safety assurance for learning-enabled control of autonomous driving vehicles. In: 2022 International Conference on Robotics and Automation (ICRA), pp. 8978–8984. IEEE (2022)
6. Cofer, D., et al.: Run-time assurance for learning-enabled systems. In: NASA Formal Methods Symposium, pp. 361–368. Springer (2020)
7. De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: the SCEL language. ACM Trans. Autonom. Adapt. Syst. **9**(2), 7:1–7:29 (2014). <https://doi.org/10.1145/2619998>
8. De Nicola, R., Wirsing, M.: Rigorous engineering of collective adaptive systems: special section on ensemble-based autonomic computing. Int. J. Softw. Tools Technol. Transfer **22**, 1–5 (2020)
9. De Nicola, R., Wirsing, M.: Rigorous engineering of collective adaptive systems. In: ISO/FA 2022. LNCS, vol. 13702, pp. 3–10. Springer (2022)
10. Ferrando, A.: Watchdogs and oracles: runtime verification meets large language models for autonomous systems. In: Proceedings of the 4th International Workshop on Formal Methods for Autonomous Systems (FMAS 2025). Electronic Proceedings in Theoretical Computer Science, vol. 436, pp. 80–87 (2025)
11. Gil, M., Albert, M., Fons, J., Pelechano, V.: Designing human-in-the-loop autonomous cyber-physical systems. Int. J. Hum Comput Stud. **130**, 21–39 (2019)
12. Hao, Y., Chen, Y., Zhang, Y., Fan, C.: Large language models can solve real-world planning rigorously with formal verification tools. In: Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 3434–3483 (2025)
13. Huang, X., Ruan, W., Huang, W., et al.: A survey of safety and trustworthiness of large language models through the lens of verification and validation. arXiv preprint [arXiv:2305.11391](https://arxiv.org/abs/2305.11391) (2023)
14. Miculan, M.: CASLLM Blocksworld pilot study: task files, model responses, validator, and analysis scripts. <https://github.com/miculan/casllm-pilot-study> (2026)
15. Pasuraman, R., Riley, V.: Humans and automation: use, misuse, disuse, abuse. Hum. Factors **39**(2), 230–253 (1997)
16. Pasqua, M., Miculan, M.: AbU: A calculus for distributed event-driven programming with attribute-based interaction. Theoret. Comput. Sci. **958**, 113841 (2023). <https://doi.org/10.1016/j.tcs.2023.113841>

17. Brogi, P., Maggesi, C., M.: Analysing collective adaptive systems by proving theorems. In: International Symposium on Leveraging Applications of Formal Methods, pp. 223–237. Springer (2024)
18. Sha, L.: Using simplicity to control complexity. *IEEE Softw.* **18**(4), 20–28 (2001). <https://doi.org/10.1109/MS.2001.936213>
19. Sheridan, T.B.: Telerobotics, Automation, and Human Supervisory Control. MIT Press (1992)
20. Slaney, J., Thiébaux, S.: Blocks world revisited. *Artif. Intell.* **125**(1–2), 119–153 (2001)
21. Tribastone, M., Vandin, A.: Quantitative abstractions for collective adaptive systems. In: Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems. LNCS, vol. 9700, pp. 161–196. Springer (2016)
22. Urban, C., Miné, A.: A review of formal methods applied to machine learning. arXiv preprint [arXiv:2104.02466](https://arxiv.org/abs/2104.02466) (2021). <https://arxiv.org/abs/2104.02466>
23. Valmeekam, K., Marquez, M., Sreedharan, S., Kambhampati, S.: On the planning abilities of large language models - a critical investigation. In: Advances in Neural Information Processing Systems (NeurIPS), vol. 36, (2023)
24. Wirsing, M., Hölzl, M. (eds.): Software Engineering for Collective Autonomic Systems: The ASCENS Approach, Springer (2015)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

