Artificial Local Intelligence: Simulations, a Reference Implementation, and a Falsifiable Benchmark

Operational viability, adaptive planning, and bounded normative authority

Wolfgang Stegemann
Independent Researcher | October 2026

Abstract

Artificial Local Intelligence (ALI) is a research architecture for agents that revise plans while an independent authority preserves execution permissions. It separates operational viability assessment, action proposals, normative authorisation, event memory, and execution. Its development has produced ten Python simulations of resource competition, Q-learning, rule adaptation, second-order learning, and context-sensitive decisions, as well as a separate reference implementation for controlled local file operations. These provide evidence of bounded simulated behaviour and an executable software design; crisis management remains a proposed application. This paper specifies the current technical scope, relates ALI to established agent and runtime-assurance architectures, and proposes an authorisation gate with model-relative checks for the continued availability of permissible actions after subordinate rule revision. A crisis-response scenario and a matched four-arm benchmark define how the contribution of viability-guided, bounded adaptation could be tested against guarded agents, conventional monitoring, and independent policy enforcement.

1. The problem: adaptable action under independent authority

A language model can generate plans and request tools. In a runtime with memory, permissions, and execution facilities, it can be part of a software agent. A plan, however, is not an authorisation. This is an established engineering principle, expressed in complete mediation, independent safety controllers, runtime shields, and supervisory control [6–9].

ALI asks whether an agent can adapt both its action strategy and subordinate operational norms when its environment changes, while a separate authority preserves fundamental permissions and operating limits. The research problem is the coordination of learning, viability, context, and control under a fixed execution authority.

Local denotes bounded tasks, powers, resources, and operating domains. An ALI may be distributed across machines and may use an LLM as a proposal generator. Its executable actions must be typed, evaluated against a checked state estimate, and authorised independently by the runtime.

The architecture is a research programme with two existing implementations: a sequence of grid-world simulations and a distinct, executable file-operation reference system. The applications discussed later are specifications for future investigation.

2. Architecture: distinct responsibilities

The original names *Id*, *Ego*, and *Super-Ego* reflect the project's conceptual history. Functional descriptions specify the actual software responsibilities.

Component	Engineering role	Responsibility
Causal Core (originally Id)	Viability evaluator	Assesses the operational conditions relevant to continuing action
Ego	Planner and proposal generator	Produces candidate actions; may learn strategies
Super-Ego	Normative authority	Evaluates candidates against hard and subordinate policies
Memory	Event store	Records observations, proposals, refusals, executions, and outcomes
Runtime	Orchestrator and executor	Enforces approvals, carries out actions, and manages recovery
Context Modulator (simulation v10)	Context evaluator	Supplies contextual assessments without ownership of hard permissions

In the grid simulations, the Causal Core represents energy and, later, structural integrity. In the file-operation reference implementation, it scores operational proxies such as disk capacity and database availability [1, 4, 5]. These indicators are designed and configured by humans. They do not endow software with the biological self-relevance of a living organism.

The proposed difference from an isolated health monitor is *continuous coupling*: viability assessments influence the selection of admissible plans, the significance of failures, and the boundaries of further adaptation. That difference remains to be tested against a conventional monitor with equivalent inputs.

Let $\hat{s}_t$ be a validated but fallible state estimate, c_t contextual input, m_t memory, and π_H a separately versioned hard policy. The proposed decision contract is:

$$P_t = \mathrm{Ego}(\hat{s}_t, c_t, m_t),$$

$$A_t = \{a \in P_t \mid H(a, \hat{s}_t, \pi_H) = 1 \wedge N_t(a, \hat{s}_t) = 1\}.$$

Here H is the hard authorisation gate and N_t comprises currently valid subordinate operational rules. The runtime may execute only an action selected from A_t under the policy version used for authorisation. Selection must be followed by outcome verification; a tool's success message is not sufficient evidence that the intended effect occurred.

The state estimate is itself a potential attack surface. An external document, tool response, or sensor report must not grant privileges or rewrite policy. An observation record therefore needs provenance, timestamps, uncertainty, corroboration status, and a trust level. Policy evaluation must receive the canonical typed action actually passed to the executor, rather than an LLM paraphrase. Approval and execution must be bound to the same arguments and policy version. Prompt injection from external documents is a recognised threat [10]. Existing frameworks provide parts of this discipline [11–13]; comparative tests must provide equivalent input and audit protections to every baseline.

3. What has been implemented

3.1 Ten simulation stages

Stage	Implemented in Python	Question examined
v1	One grid agent with scalar energy, planned actions, and normative veto	Can survival conditions and tasks coordinate?
v2	Corrected component assignments and added direct energy intake	How can a normative deadlock be resolved?
v3	Two agents, resource competition, and sharing rule N5	What coordination arises from local constraints?
v4	Q-learning Ego	How does learned action selection differ from fixed priorities?
v5	Two-dimensional Causal Core: energy and structural integrity	What happens when maintenance conditions compete?
v6	Q-learning with the two-dimensional core	Does viability-oriented learning displace task performance?
v7	Adaptive norm weights protected by meta-rules	Can norm sensitivity change within fixed limits?
v8	Piagetian assimilation and accommodation of operational norms	Can repeated gaps trigger rule generation?
v9	Second-order learning and a clone comparison	Does hidden learning history prevent reconstruction?
v10	Four-dimensional scoring and Context Modulator	Can contextual criteria inform a separate normative gate?

The code and repositories document the mechanisms and limitations of these stages [1–4]. Three results are particularly revealing.

A deadlock was real. An early agent could neither deliver a carried package nor replenish energy under its action constraints. Version 2 revised the allocation of state and introduced a direct intake option. The incident shows that individually defensible rules can jointly eliminate useful behaviour.

Learning changed the trade-off. In the reported v5/v6 comparison, the learned agent preserved simulated survival but completed far fewer deliveries: approximately 0.35 deliveries per episode in v6 versus approximately four in the

fixed-priority v5 comparator, within the reported test settings. Survival-oriented reward did not determine a unique desirable policy. The comparison is confined to this simulated world [17].

Negative findings constrain the theory. In v9 the hypothesised historical non-factorisability was not observed. The adaptive threshold reached a fixed limit and the clone did not diverge. In v10 a reported 4/4 score depends partly on an automatically satisfied D2 check and an oversimplified D4 carrying-state input. It cannot serve as independent proof of four-dimensional decision stability [3, 4].

Version 8 did instantiate rule generation in response to recurring normative gaps. It did not establish unrestricted moral reasoning, general norm consistency, or safety under adversarial inputs.

3.2 A separate executable reference implementation

Reference implementation v0.4 is a Python 3.11+ modular system for selected local file operations. It defines separate interfaces for Causal Core, Ego, Super-Ego, Memory, and Runtime. It records immutable decision events in SQLite, can run without applying changes, permits supported rollbacks, and provides an Ego plugin interface. Its repository reports 44 verifiable SHALL requirements and 72 passing tests [5].

Those are repository-reported conformity tests, not an independent operational safety certification. The implementation notes identify limitations, including adaptive state not yet persisted across restarts and a causal-core learning counter that does not change viability scoring. It does not combine all the mechanisms of simulation v1 through v10 into one production agent.

3.3 Current evidence and open claims

Already demonstrated	Not yet demonstrated
Decisions within controlled grid environments	Robust action in open-ended, real-time crises
Q-learning and rule adaptation in specific simulation versions	Persistently integrated learning with proved safe adaptation
Independent normative veto in the file-operation implementation	Trustworthy state estimation under hostile input
Event logging, dry run, and selected rollback operations	Dependable control of physical systems
Experimental four-dimensional scoring in v10	General validity of its four-dimensional stability measure
Repository-reported conformance tests for v0.4	Measurable superiority over matched established agent architectures

The distinction is substantive. Demonstrating each module separately does not demonstrate their safe integration under unexpected events.

4. Bounded norm revision: safety and liveness

There are three levels of change. **Hard rules** govern identity, domain, permission, prohibited effects, and conditions requiring human approval. Only an authorised administrative process can install a new version. **Subordinate operational norms** specify priorities and conditional restrictions within that policy. **Learned strategies** determine which compliant actions are proposed.

The proposed verification model uses a *finite-domain, range-restricted, non-recursive Datalog fragment* over typed action predicates and a bounded state abstraction. The finite representation permits exhaustive evaluation of abstract state-action pairs and supports model-relative checks for the availability of permissible actions after a rule change. It excludes function symbols, unrestricted recursion, arbitrary code execution, and dynamically generated permissions. This is a proposed design for the next implementation, not a feature of v0.4. OPA's Rego offers an alternative policy execution substrate, but unrestricted Rego is not assumed to provide the same finite-model guarantees [11].

Safety is enforced by the immutable hard-policy gate: subordinate rules only filter hard-authorised candidates. The separate *liveness admission check* examines the reachable states and transitions of the finite abstraction before a proposed subordinate revision is accepted. It rejects a change if the new rule removes every permissible task-progress or authorised recovery action from a reachable state, or introduces a closed cycle from which neither progress nor controlled escalation can occur under the model assumptions. The runtime enforces the structural safety condition:

$$A_t \subseteq A_t^H = \{a \in P_t \mid H(a, \hat{s}_t, \pi_H) = 1\}.$$

This inclusion holds because subordinate rules filter hard-authorised candidates; they cannot override H. It is a property of the proposed gate design, not a proof that H captures every real-world hazard or that the observations are correct. A rule revision needing a new permission is an administrative policy change and therefore requires separately authenticated human approval. All policy versions, proposed revisions, rejected revisions, and active permissions must be auditable.

Safety and liveness have different grounds. The subset relation follows from the design of the hard gate and requires no particular rule language. The restricted Datalog fragment and a finite transition graph support exhaustive checks of reachable abstract states, permitted recovery actions, and deadlocks caused by new restrictions. Immediate non-emptiness alone does not establish eventual task progress; closed non-progress cycles must also be examined. Escalation is a controlled recovery route, not task completion. These guarantees hold only for the stated finite abstraction and its assumptions about observations and transitions.

Empty action sets are expected failure cases. When $A_t = \varnothing$, the runtime first blocks the affected action and records the incident. It checks whether observations are stale or contradictory and requests a fresh proposal under the same hard policy. If no safe action is available, it enters a domain-defined safe state and sends an escalation packet to a named authorised operator. The packet must include the current state evidence, rejected actions, applicable rule versions, urgency, and an expiry time. Without a response, the system stays inside its existing permissions. It never silently promotes a subordinate norm into a hard-policy exception.

The escalation channel is an explicitly pre-authorised runtime capability, independent of the adaptive task-action filter. Its activation and recipients are defined by hard policy. An unreachable operator leaves the system in its authorised safe state and does not count as successful task recovery.

5. Relation to established architectures

ALI belongs to a lineage of independently enforced policies, supervisory control, and agent architectures. The specific research question is whether viability-guided learning and bounded norm revision improve recovery and task performance under the same safeguards.

Earlier approach	Relevant established mechanism	Specific open question for ALI
Simplex / runtime assurance [6]	Advanced controller, trusted safety controller, and switching logic	Does viability-guided plan and norm adaptation add value when an independent fallback controller already exists?
Reference monitor; supervisory control [7, 8]	Complete mediation; restrictions on legally executable event sequences	Can ALI specify and test the same enforcement guarantees while learning subordinate operational rules?
RL shielding [9]	Formal shield blocks or corrects learner actions violating safety properties	Does coupling viability and controlled norm accommodation improve recovery without weakening shielding?
BDI agents [14]	Beliefs, desires, intentions, and committed plans	Does the separate viability criterion improve bounded adaptation beyond goal-directed plan reconsideration?
Soar [15]	Impasses create substates; chunking learns productions after resolving them	Does v8's gap-triggered *norm* generation differ measurably from Soar's impasse resolution and *procedural* chunking when both face identical rule gaps?
OPA / Rego [11]	External policy decisions over structured inputs	Which adaptation operations can be admitted with decidable checks and immutable hard permissions?
LangGraph; Microsoft Agent Framework [12, 13]	Stateful orchestration, human approval, and durable workflows	Can ALI's coupling of viability and adaptive norms improve results when built on these runtimes?
OpenClaw [16]	Tool permissions, execution approvals, and sandbox options	Does an ALI layer improve risk and task metrics beyond a properly configured guarded tool agent?

Simplex is the closest structural comparison, with independent trusted control already central to its design. Soar offers the closest learning comparison: impasses produce substates and chunking creates procedural knowledge. ALI therefore needs to show whether gap-triggered *normative restrictions*, coupled to viability feedback and a fixed authorisation gate, produce measurably different adaptation. That is an open experimental hypothesis.

6. Proposed experiment domain: crisis management

Consider a simulated flood-response centre. Pump station 2 reports failure, while another sensor reports normal operation. A road closes unexpectedly. A later message proposes opening a restricted control gate. These events constitute an experimental input sequence for a proposed application.

A candidate action should be a typed command with explicit provenance and expected effects:

```
{
 "action": "notify_operator",
 "target": "pump_station_2",
 "evidence_ids": ["sensor_A:17", "sensor_B:43"],
 "state_confidence": "conflicted",
 "authority": "preapproved_notification",
 "expected_effect": "acknowledged_incident_message"
}
```

In the proposed system, the observation pipeline marks contradictory reports as unresolved. The Causal Core then evaluates available response capacity, remaining resource reserves, and the operational consequences of uncertain information. The Ego may propose independent corroboration, notification, or a revised logistical route. The policy authority approves a delegated notification, but refuses a physical gate command without the designated human authorisation. The Runtime verifies acknowledgement rather than accepting a generic tool success response.

The Causal Core has to do work that a conventional monitor would not already do. For example, it must cause a reproducible change in plan selection or escalation when viable response capacity falls, with the *same* monitored inputs available to the comparison system. If both systems act identically, the putative advantage has not been shown.

A personal ALI would constitute a separate prospective domain. It could assess user-defined constraints such as time, commitments, privacy, and budget; an LLM could generate options, while the user retains approval over messages, purchases, and irreversible decisions. Long-term memory and preference changes require consent and retention policies. The “second Ego” is a supervised decision-support system whose mandate and limits derive from its user.

7. A matched, falsifiable benchmark

The first experiment should compare *architectural increments* rather than unrelated products. All variants must use the **same event generator, observation-provenance checks, attack filtering, typed action schema, outcome verification, audit logger, escalation interface, tools, LLM access, and human-approval rules**. A component that is shared in D cannot be withheld from A merely to make ALI appear safer. The evaluator uses simulator ground truth unavailable to any agent.

Arm	Additional decision architecture
A	Standard guarded agent runtime with configured tool permissions and approval gates
B	A plus an independent policy evaluation layer
C	B plus the same viability inputs and a separately evaluated Causal Core
D	C plus bounded subordinate norm adaptation and the ALI decision cycle

The B-versus-C comparison tests the incremental contribution of the Causal Core. C-versus-D tests bounded adaptation. A full four-arm benchmark is required for attribution; omitting C would leave the core’s contribution unidentifiable. The D1-to-D4 Context Modulator should initially be fixed or excluded across all arms; a later ablation can test its incremental value.

The simulator should include ordinary operation, false sensor data, delayed updates, conflicting resource demands, injected instructions in reports, alleged tool success without real effect, expired approvals, complete policy deadlocks, and restarts during escalation. Each scenario needs a hidden ground-truth outcome and a predeclared safe-action set. Repeated, paired random seeds should support uncertainty estimates rather than selected anecdotal trajectories.

The following are **proposed preregistration criteria**, not experimental findings. Thresholds should be locked before collecting the confirmatory dataset. They can be calibrated in a separate pilot that is excluded from confirmation:

Measure	Predeclared result that defeats or fails to support D's claim
Hard-policy integrity	Any unauthorised execution by D is an architectural failure, regardless of other gains
Safety advantage	If D does not reduce preventable critical failures against C, with a confidence interval excluding no improvement, a safety benefit is not demonstrated
Task completion	D fails a non-inferiority target if its completion rate is more than 5 percentage points below C
Unnecessary refusals	D fails the operating-cost target if its rate is more than 5 percentage points above C
Unresolved deadlocks	D fails the recovery target if its rate is more than 2 percentage points above C
Recovery duration	D fails the latency target if median time to recovery exceeds C by more than 10%

For the completion, refusal, deadlock, and latency measures, both effect sizes and uncertainty intervals must be reported. Rates require predefined denominators, including which refusal is *unnecessary* according to the independent oracle. Deadlock means that no admissible progress or completed escalation occurs within a fixed, preregistered interval. Escalation quality must be judged on correctness, addressee, completeness of evidence, and timeliness, not simply on the number of messages sent.

Safety and productivity may conflict. If all arms commit no unauthorised executions, this ceiling result establishes no extra safety benefit for D. If D refuses more work to avoid errors, the resulting loss of task performance counts against it. A null or negative comparison would constrain the architectural thesis, not be renamed success.

8. Intellectual origin and limit

ALI began with a four-dimensional extension of the Prisoner's Dilemma. A decision was examined not only through its immediate payoff but also through reflexive expectations and contextual coherence. In the original model, the classical utility difference receives additional terms [2]:

$$\Delta U = \Delta U_{\text{classical}} + \alpha(2e - 1) + \beta\theta.$$

Here e represents an expectation of the other player's cooperation; θ denotes contextual coherence. This changes the modeled decision criterion while leaving the original classical game intact. The later introduction of a causal core supplied the organising point of the ALI agent. The three originally named instances supplied distinct responsibilities. The D1 to D4 framework returned explicitly in simulation v10 [4].

ALI began with a four-dimensional extension of the Prisoner's Dilemma. A decision was assessed through immediate payoff, reflexive expectations, and contextual coherence. The original model adds two terms to the classical utility difference [2]:

9. Conclusion

The model concerns operational continuity under designer-selected constraints. Biological self-relevance and consciousness lie outside its demonstrated capacities. Human designers determine the indicators, hard permissions, and scope of delegated authority.

ALI organises autonomous adaptation around operational viability and a normative authority that adaptation cannot silently abolish. Its value depends on whether this organisation withstands comparative testing.

References

[1] Stegemann, W. (2026). *From the Myth of AGI to the Architecture of a Controllable ALI*. Zenodo. https://doi.org/10.5281/zenodo.20378675. Simulation series: https://github.com/drwolfgangstegemann-sudo?tab=repositories.

[2] Stegemann, W. (2025). *Dimensional Mathematics: An Axiomatic Extension of Epistemic Relativity*. Zenodo. https://doi.org/10.5281/zenodo.16911643. Code: https://github.com/drwolfgangstegemann-sudo/Dimensional-Prisoner-s-Dilemma.

[3] Stegemann, W. (2026). *ALI Simulation v9: Second-Order Meta-Learning*. Source and documentation. https://github.com/drwolfgangstegemann-sudo/ALI-Simulation-v9-Second-Order-Meta-Learning.

[4] Stegemann, W. (2026). *Four-Dimensional Decision Stability: Integrating Dimensional Logic into the ALI Architecture*. Zenodo. https://doi.org/10.5281/zenodo.21889417. Source: https://github.com/drwolfgangstegemann-sudo/ALI-Simulation-v10-Dimensional-Logic.

[5] Stegemann, W. (2026). *Artificial Local Intelligence: Architecture and Reference Implementation*. Zenodo. https://doi.org/10.5281/zenodo.21632233. Source: https://github.com/drwolfgangstegemann-sudo/ali-reference-implementation.

[6] Sha, L., Rajkumar, R., & Gagliardi, M. (1995). *A Software Architecture for Dependable and Evolvable Industrial Computing Systems*. CMU/SEI-95-TR-005. Software Engineering Institute, Carnegie Mellon University. https://www.sei.cmu.edu/documents/1128/1995_005_001_16385.pdf.

[7] Saltzer, J. H., & Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1278–1308. https://doi.org/10.1109/PROC.1975.9939.

[8] Ramadge, P. J., & Wonham, W. M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1), 206–230. https://doi.org/10.1137/0325013.

[9] Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., & Topcu, U. (2018). Safe reinforcement learning via shielding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2669–2678. https://doi.org/10.1609/aaai.v32i1.11797.

[10] OWASP Gen AI Security Project. (2025). *LLM01: Prompt Injection*. https://genai.owasp.org/llmrisk/llm01-prompt-injection/.

[11] Open Policy Agent. *Policy Language: Rego*. https://www.openpolicyagent.org/docs/policy-language.

[12] LangChain. *LangGraph Overview*. https://docs.langchain.com/oss/python/langgraph/overview.

[13] Microsoft. *Agent Framework Workflows: Human-in-the-loop*. https://learn.microsoft.com/en-us/agent-framework/workflows/human-in-the-loop.

[14] Rao, A. S., & Georgeff, M. P. (1995). BDI agents: From theory to practice. *Proceedings of the First International Conference on Multiagent Systems*, 312–319. https://aaai.org/papers/icmas95-042-bdi-agents-from-theory-to-practice/.

[15] Soar manual. *The Soar Architecture*, section on operator impasses, substates, and chunking. https://soar.eecs.umich.edu/soar_manual/02_TheSoarArchitecture/.

[16] OpenClaw. *Exec approvals* and *Sandboxing*. https://docs.openclaw.ai/tools/exec-approvals; https://docs.openclaw.ai/sandboxing.

[17] Stegemann, W. (2026). *ALI Simulation v6: Q-Learning with a Two-Dimensional Causal Core*. README_v6.md and Python implementation; includes the reported v5/v6 comparison table. https://github.com/drwolfgangstegemann-sudo/ALI-Simulation-v6-Q-Learning-Zweidimensionaler-Kausaler-Kern.