CORE-001 · 16-BIT ROUTING FRAME · OPERATION / SOURCE / EVIDENCE / DESTINATION



∀№INTTEL

Independent Artificial Intelligence Research & Development

Storing information is not the same as improving from it. ∀№INTTEL investigates the difference: it builds persistent, auditable state, and tests under controlled comparison whether that state makes an artificial system's future behavior measurably better.

RESEARCH DIRECTION	DISCIPLINE	TIMEFRAME	STATUS
Tayshaun Xantana / taykOff™	SAVANT — Research & Intelligence	2026–Present	Active independent R&D

01 Research identity
02 Origin, problem & research objective
03 Research architecture & program map
04 MARK I — experimental adaptive learning
05 CORE-001 — persistent observation architecture
06 Information processing & state continuity
07 Candidate cognitive architecture
08 Adaptive architecture & proto-subjectivity
09 Experimental evidence, results & limitations
10 Research evolution & development roadmap
11 Research contribution, references & archive

02

ORIGIN, PROBLEM & RESEARCH OBJECTIVE

Remembering is not the same as improving.

Assistant "memory" usually means storing facts and retrieving them later. That can make a system more convenient without making it better at anything. ∀NºINT[TEL] starts from that gap: it asks whether an artificial system's accumulated state can be shown to change — and improve — what it does next.

CENTRAL RESEARCH QUESTION

> Can an artificial system retain experience, reconcile conflicting information, and use its evolving internal state to improve future behavior in a measurable, reproducible way?

THREE REQUIREMENTS BEHIND THE QUESTION

PERSISTENCE	Experience that vanishes at the end of a session cannot accumulate. State has to survive restarts intact — and it has to be possible to rebuild it exactly from its own history.
CONSISTENCY	Real information conflicts. A system needs an explicit rule for which observation governs its current state, and it must keep the observations it rejected, because they may matter later.
CAUSAL INFLUENCE	Stored state is only interesting if it changes behavior. The test is counterfactual: remove, freeze or disrupt the state, and performance should change in a predicted direction.

STORED MEMORY VS. BEHAVIORAL ADAPTATION

	STORED MEMORY	ADAPTATION
What changes	Contents of a store	How future inputs are processed
Typical test	Can it recall X?	Does it do better on unseen tasks?
Evidence	Retrieval accuracy	Held-out gains over matched controls; ablations
Failure mode	Remembers, but does not improve	Improves only where it trained

PROGRAM ORIGIN

∀NºINT[TEL] began in 2026 as a structured assistant system: a project taxonomy, a custom command layer, and a schema that kept learned routing state separate from fixed instructions. Building it made a narrower question unavoidable — does retained state, such as accumulated corrections, actually change outcomes, or does it only accumulate? The project is being rebuilt as a research program around that question, under a standing rule: the language model is a substrate, not the invention, so architecture and records are kept outside any single model.

NEAR-TERM ENGINEERING OBJECTIVES

01 Durable, inspectable state that survives restarts and can be rebuilt exactly. → CORE-001

02 Explicit, deterministic handling of conflicting information, with the full history kept. → CORE-001

03 Controlled benchmarks that measure adaptation against memory-only and stateless baselines. → MARK I

LONG-TERM QUESTIONS · HYPOTHESES ONLY

01 Can processing methods themselves adapt — verifiably, reversibly, and under validation?

02 Can a system keep a self-model that is persistent, testable, and causally involved in its decisions?

03 Can "proto-subjectivity" be defined operationally enough to be tested — and to fail?

Beyond personalization. Conventional personalization asks what a system knows about its user. ∀NºINT[TEL] asks what a system has learned from its own experience — and whether that can be demonstrated rather than asserted.

03

RESEARCH ARCHITECTURE & PROGRAM MAP

Eight stages, each mapped to its evidence.

∀NºINT[TEL] is organized as a pipeline of concerns, from taking in information to revising the architecture that processes it. The stages are not all built, and the built ones do not yet form one integrated system. Each stage carries the label of its strongest current evidence.

IMPLEMENTED — Exists as working code. Reported by the research director; source not attached to this edition.

REPORTED RESULT — Measured in earlier project documentation; not re-run for this edition.

PROPOSED — Specified or designed, but not built.

UNTESTED — Built or specified, with no evaluation yet.

FUTURE RESEARCH — An open question the program intends to pursue.

1	**Information intake** Observations enter the system.	IMPLEMENTED	CORE-001 Python prototype · 4 Nov 2026 Stages 1–5: stores, routes, reconciles, persists. No learning.
2	**Observation representation** Each observation becomes one atomic record.	IMPLEMENTED	
3	**Routing & dispatch** Fixed 16-bit frame built; state-dependent router proposed (§07).	IMPLEMENTED PROPOSED	
4	**Conflict reconciliation** Deterministic policy; rejected reports are retained.	IMPLEMENTED	
5	**Persistent state** Hash-linked SQLite ledger; state rebuilt after restart.	IMPLEMENTED	
6	**Evaluation & feedback** Outcomes scored by an evaluator inside the MARK I loop.	REPORTED RESULT	MARK I Synthetic benchmark · 21 Oct 2026 Own loop; not part of CORE-001.
7	**Adaptive transformation** Rule learning inside MARK I; the general case is proposed.	REPORTED RESULT PROPOSED	
8	**Architectural revision** Versioned changes to the architecture itself.	PROPOSED	PROPOSED · §08 · §10

Solid spine: documented flow between stages. Dashed: proposed. The dashed return path marks the proposed revision loop, in which stage 8 would act on routing and transformation (stages 3 and 7).

BUILT

Stages 1–5 exist as CORE-001, a Python prototype that stores, routes, reconciles and persists observations. It does not learn (§05–06).

MEASURED

Stages 6–7 were exercised inside MARK I, a synthetic benchmark with its own control loop. Its results are reported from earlier documentation (§04).

PROPOSED

Stage 8, and a state-dependent router that would replace fixed dispatch at stage 3, are designs without implementation evidence (§07–08).

Not yet done: connecting MARK I-style adaptation to CORE-001's persistence layer. Until that integration is built and evaluated, the stages above are a research program with working parts — not one intelligent system. Integration is listed as a proposed milestone (§10).

04

MARK I — EXPERIMENTAL ADAPTIVE LEARNING

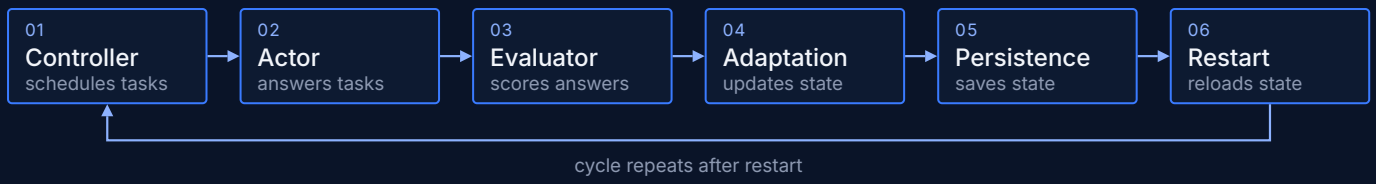
149 of 160: what MARK I shows, and what it doesn't.

REPORTED RESULT

21 Oct 2026 · synthetic hidden-rule benchmark · source: MARK I Research Brief, rev. 1 Nov 2026 (not attached)

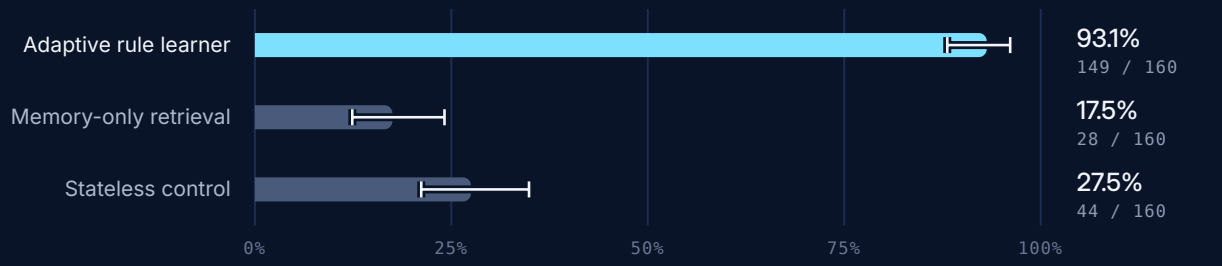
MARK I asked a narrow question under controlled conditions: in worlds governed by hidden rules, does a system that updates its state from feedback — and keeps it across restarts — beat one that only retrieves past cases, and one that keeps no state?

REPORTED CONTROL LOOP



Loop as reported in the MARK I brief. One-line stage roles are editorial summaries of the stage names, not descriptions of the code.

HELD-OUT ACCURACY · 160 TASKS (5 WORLDS × 32) · WHISKERS: 95% INTERVALS COMPUTED FOR THIS EDITION



SYSTEM	CORRECT	ACCURACY	95% CI
Adaptive rule learner	149/160	93.1%	88.1–96.1%
Memory-only retrieval	28/160	17.5%	12.4–24.1%
Stateless control	44/160	27.5%	21.2–34.9%

Intervals use the Wilson score method [4], computed here from the reported counts. They treat the 160 tasks as independent; tasks are grouped into five worlds, so true uncertainty is likely wider.

DESIGN & SCOPE

- Five hidden-rule worlds; 128 training and 32 held-out tasks in each (640 and 160 in total).
- In-family evaluation: held-out tasks come from the same synthetic worlds as training.
- Resource budgets and built-in assumptions were not matched across systems.
- Figures come from the MARK I brief; they were not re-run for this publication.

WHAT IT DEMONSTRATES

Within this synthetic setting, the adaptive learner answered far more held-out tasks correctly than either baseline: 93.1% against 27.5% for the stateless control and 17.5% for memory-only retrieval. The gap is much larger than the sampling uncertainty in the counts — a clear result, inside clear bounds.

WHAT REMAINS UNCERTAIN

- **Source of the advantage.** How much comes from accumulated experience versus built-in assumptions. A same-learner ablation — state reset or withheld — would separate the two.
- **Robustness.** Whether the gap survives matched budgets, stronger baselines and repeated runs.
- **Transfer.** Whether anything carries over outside the training family.
- **An anomaly.** Memory-only retrieval scored below the stateless control (17.5% vs 27.5%). Per-task data would be needed to explain why.
- **Not stated in available material:** chance level, per-world results, run-to-run variance.
- **Not established:** any claim of general AI superiority.

05

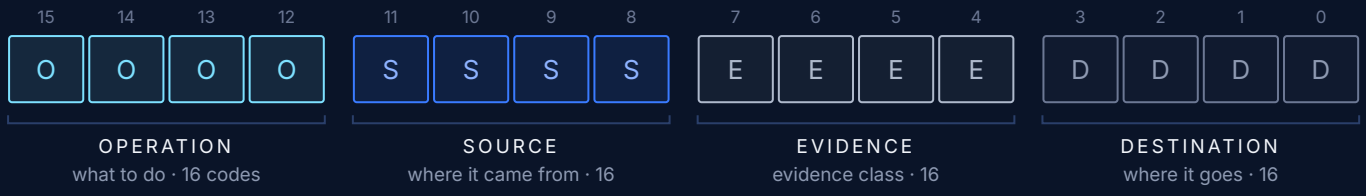
CORE-001 — PERSISTENT OBSERVATION ARCHITECTURE

Keep every observation. Rebuild state exactly.

IMPLEMENTED 4 Nov 2026 · Python engineering prototype · source, tests and demo output not attached to this edition

CORE-001 is the program's first persistence layer. It takes in observations as atomic records, routes them with a compact 16-bit frame, appends each event to an ordered, hash-linked ledger in SQLite transactions, reconciles conflicts deterministically, and checks after a restart that the same history rebuilds the same state. It does not learn.

16-BIT ROUTING FRAME · OOOO SSSS EEEE DDDD

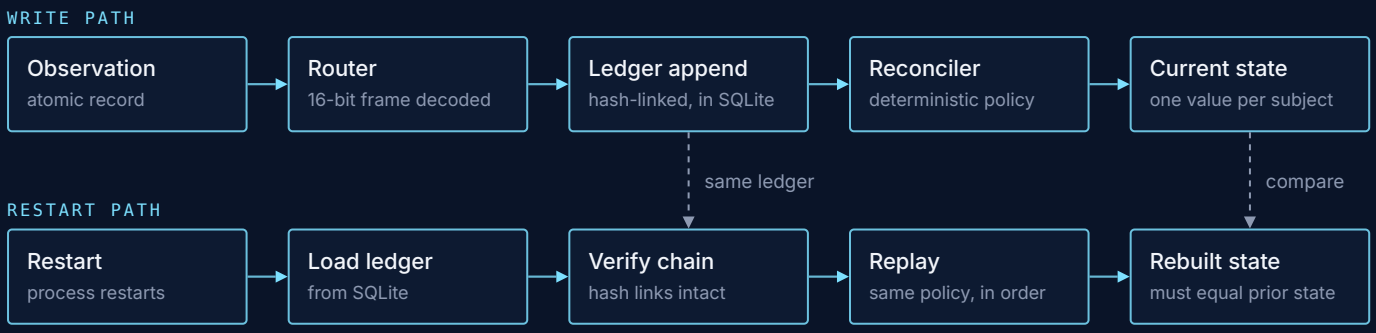


A software encoding, not hardware. The frame is an ordinary 16-bit integer that Python code packs and unpacks to decide what happens to an event. It is not a processor design and not a new machine instruction set; bit-field headers like it are long-established practice.

```
frame = (O<<12) | (S<<8) | (E<<4) | D
```

4 bits per field: up to 16 codes each, 2^{16} = 65,536 possible frames. Bit positions assume the layout reads most-significant first. Code assignments live in the CORE-001 source and are not reproduced here.

WRITE PATH AND RESTART PATH



Derived from the documented component list. Module boundaries and call order in the source may differ.

ATOMIC RECORDS
Each observation is stored as one indivisible record.

ROUTING FRAME
Operation, source, evidence class and destination in one integer.

ORDERED LEDGER
Events keep a strict order; state is derived from the sequence.

HASH LINKING
Each event is chained to the one before it [2]; editing history breaks the chain.

SQLITE TRANSACTIONS
A write commits completely or not at all [3].

DETERMINISTIC RECONCILIATION
The same ledger always yields the same current state.

RESTART VERIFICATION
After a restart, state is rebuilt from the ledger and compared.

INTEGRITY CHECKS
The hash chain and the rebuilt state are checked for consistency.

What CORE-001 is not. Not a learning system, a reasoning engine, a hardware design, or evidence of cognition. It is infrastructure a learning system would need: durable, ordered, checkable experience.

No code excerpts appear in this edition. The CORE-001 source was not attached to this production cycle, and excerpts are shown only when they can be quoted from inspected source.

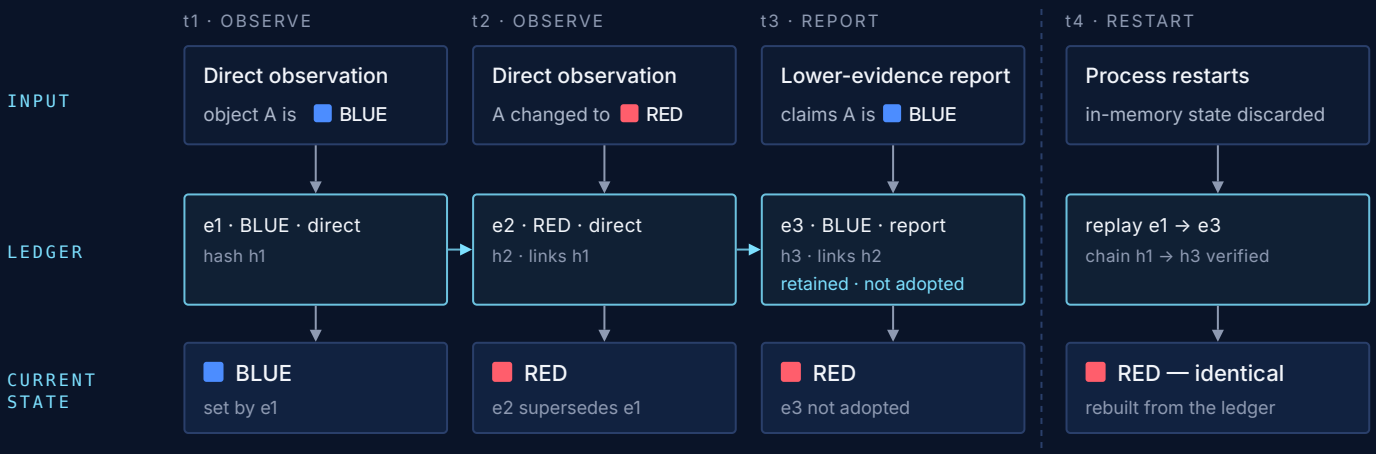
06

INFORMATION PROCESSING & STATE CONTINUITY

Keep every observation. Choose one current state. Rebuild it exactly.

CORE-001 separates two things that are easy to conflate: the history of what was observed, and the working state derived from it. History is retained — including what was rejected — and current state is a deterministic function of that history.

DEMONSTRATION SCENARIO · ONE OBJECT, THREE EVENTS, ONE RESTART



After the restart, the ledger still holds e1, e2 and e3 — every observation, including the report that was not adopted.

Scenario as specified in the CORE-001 documentation. Demonstration output was not attached to this edition; the diagram shows the documented sequence, not captured program output.

Why keep contradictions

A rejected report is still information. It may later prove correct, expose an unreliable source, or explain a downstream error. Deleting it would make the system's past unauditable.

How current state is chosen

A fixed policy maps the ordered ledger to one working value per subject. In the scenario, a lower-evidence report cannot displace a later direct observation.

Why determinism matters

If the same history always yields the same state, anyone holding the ledger can recompute and check it. Restarts become verifiable rather than trusted, and debugging becomes replay [1].

Evidence labels are not truth

Evidence classes are supplied by whoever submits an observation. If a rumor is labeled as a direct observation, the policy will prefer it — consistently. CORE-001 enforces consistency, not correctness.

Persistence is not intelligence

Surviving restarts and replaying history are engineering properties. An adaptive system needs them to accumulate experience, but they do not make anything learn, reason or understand.

IMPLEMENTATION CONSTRAINTS & INTEGRITY LIMITATIONS

- **Tamper-evident, not tamper-proof.** Anyone with write access can rewrite events and recompute the chain, unless chain heads are anchored outside the database — not described in current documentation.
- **Fixed policy.** The reconciliation rule is specified by hand, not learned.
- **Coarse evidence.** A 4-bit field allows at most 16 classes, with no calibration or provenance scoring — the target of CORE-002.
- **Not yet characterized.** Concurrency, scale, ledger growth and replay cost.

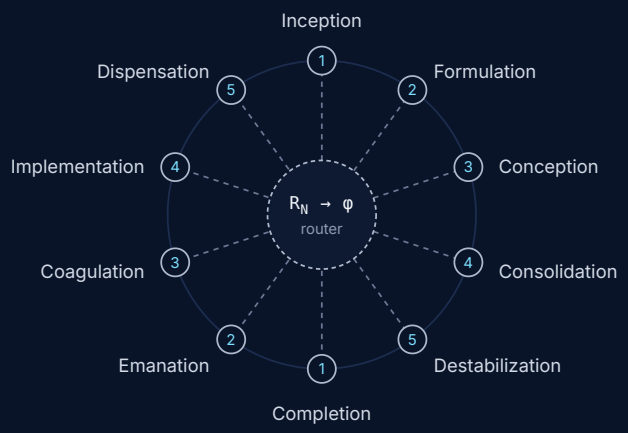
07

CANDIDATE COGNITIVE ARCHITECTURE

Let internal state choose the next operation.

PROPOSED Design proposal · no implementation evidence claimed

CORE-001 dispatches each event using fields carried in its frame. The candidate architecture proposes a different router: one that reads the system's internal state and selects one of ten transformations, so the same input can be processed differently depending on what the system has already become.



Ten operations in five complementary pairs; each pair shares a number and a diameter. The central router selects which operation runs. Proposed; not implemented.

PAIR	OPERATION		COUNTERPART
P1	Inception	↔	Completion
P2	Formulation	↔	Emanation
P3	Conception	↔	Coagulation
P4	Consolidation	↔	Implementation
P5	Destabilization	↔	Dispensation

Each operation is defined with a complementary counterpart. The labels name intended roles; they are not yet specified at implementation level, and they are **not evidence of ten distinct, functioning cognitive algorithms**.

DESIGN CONSTRAINT

Carried from earlier ∀NºINT[TEL] routing work: **explicit objectives override learned state.** Learned preferences may inform a choice; they never outrank an explicit instruction.

PROPOSED STATE-TRANSITION FORMULATION

$$S[t+1] = F_{\varphi(R_N(z[t],\ S[t]))}(S[t],\ I[t])$$

At each step a router scores the candidate operations using the current context and state; a selection rule picks one; that operation computes the next state from the current state and the new input. State therefore does two jobs: it is what each transformation updates, and it helps decide which transformation runs.

- $S[t]$ internal state at step t
- $I[t]$ input arriving at step t
- $z[t]$* routing context at step t
- R_N* router scoring N candidate operations (N = 10 here)
- φ* selection rule: router scores → one operation index k
- F_k k-th transformation: (S[t], I[t]) → S[t+1]

* Readings of z, R_N and φ follow the structure of the formula. The material available to this edition does not define them further — including whether φ is deterministic, stochastic or learned.

	CORE-001 ROUTER IMPLEMENTED	CANDIDATE ROUTER PROPOSED
Decides using	Fields in the event's 16-bit frame	Context z[t] and internal state S[t]
Chooses among	Up to 16 operation codes	Ten paired transformations
Selection	Fixed bit-field decoding	φ applied to router scores
Adapts over time	No	Intended to; untested

WHAT WOULD COUNT AS EVIDENCE

01 **Implementation.** Each operation exists as code with a defined input/output contract.

02 **Distinct signature.** Each produces measurably different effects from the others.

03 **Causal necessity.** Removing an operation — or the state-dependence of routing — degrades performance under matched conditions.

08

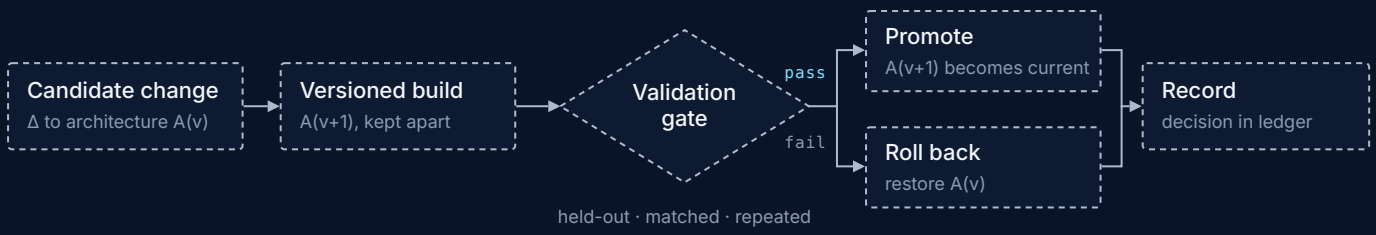
ADAPTIVE ARCHITECTURE & PROTO-SUBJECTIVITY

Long-horizon questions, stated so they can fail.

The longest-range directions concern systems that revise their own processing methods and keep a testable model of themselves over time. These are hypotheses, not capabilities. Each is stated with the evidence that would support it — and the results that would count against it.

A · ADAPTIVE ARCHITECTURE PROPOSED

Adaptive transformation would let a system change *how* it processes information, not only what it stores. The proposed safeguard is a versioned update gate: every candidate change is built as a new version, validated against the current one, then promoted or rolled back — with the decision recorded.



Concept from the program roadmap. No implementation exists.

IMPROVEMENT WOULD REQUIRE

01 A fixed metric and held-out set, chosen *before* the change is made.

02 Budgets matched against the previous version and against baselines.

03 Advantages that repeat across runs, with variance reported — and a check outside the training family.

04 A tested rollback path: every promoted change must be reversible.

B · SELF-MODELS & PROTO-SUBJECTIVITY FUTURE RESEARCH

Working definition. The program uses *proto-subjectivity* as a working label for a cluster of functional properties — continuity, self-model, memory, metacognition, autonomous inference, preference and self-correction — each to be tested separately. It is a research target, not a claim about experience.

OPEN QUESTIONS · PROTO-SUBJECTIVITY EXPEDITION, OCT 2026

- Can identity-relevant state persist across restarts and model swaps?
- Can preferences emerge from experience instead of being planted?
- Can reported internal states predict behavior?
- Does the self-model survive contradiction?
- Do self-reports track hidden internal changes beyond the visible record? (introspective access — the focus at the October checkpoint)

A designed test (not run). Capability forecasts are recorded before outcomes and withheld from the actor. The explicit self-model record is then intervened on directly — intact, disrupted, matched control, restored — while tasks, tools and budgets stay fixed. If forecasts and repair choices change only when the record changes, the self-model is causally involved.

THE FIREWALL

BEHAVIOR ≠ EXPERIENCE

SELF-REPORT ≠ SUBJECTIVITY

A system that sounds self-aware has shown nothing about awareness. Functional tests can establish functional properties; they cannot establish experience. Learning, self-modeling, introspective access and metacognitive control are kept as distinct mechanisms.

NOT CLAIMED

Consciousness, feelings, subjective experience, self-awareness, or autonomous self-improvement. Passing the functional tests above would not establish any of them.

09

EXPERIMENTAL EVIDENCE, RESULTS & LIMITATIONS

What exists, what was measured, what is proposed.

The matrix lists each major component with its status and source. The last column records what was checked while producing this document: arithmetic and internal consistency were checked; primary artifacts were not available to inspect.

COMPONENT	STATUS	SOURCE	CHECKED IN V1.0
16-bit routing frame (OOOO SSSS EEEE DDDD)	IMPLEMENTED	CORE-001 · 4 Nov 2026	Source not attached
Atomic observation records	IMPLEMENTED	CORE-001	Source not attached
Ordered, hash-linked ledger in SQLite	IMPLEMENTED	CORE-001	Source not attached
Deterministic reconciliation (blue → red)	IMPLEMENTED	CORE-001 demonstration	Output not attached
Restart reconstruction & integrity checks	IMPLEMENTED	CORE-001 tests	Test suite not attached
MARK I adaptive learner · 149/160 held-out	REPORTED RESULT	MARK I brief, rev. 1 Nov 2026	Counts checked · not re-run
MARK I baselines · 28/160 and 44/160	REPORTED RESULT	MARK I brief, rev. 1 Nov 2026	Counts checked · not re-run
Matched-budget runs & ablations	UNTESTED	—	—
Out-of-family generalization	UNTESTED	—	—
State-dependent ten-operation router	PROPOSED	Candidate architecture (§07)	Formula transcribed
Versioned updates, validation gate, rollback	PROPOSED	Program roadmap (§08, §10)	—
Persistent, causally involved self-model	FUTURE RESEARCH	Expedition notes, Oct 2026	Test designed, not run
Operational proto-subjectivity tests	FUTURE RESEARCH	Expedition notes, Oct 2026	—

ESTABLISHED · USED DELIBERATELY	HYPOTHESES · NOVELTY TO BE SHOWN
Bit-field headers for dispatch	State-dependent routing over paired operations beats fixed routing
Event sourcing: state as a fold over an ordered log [1]	Evidence-aware persistence plus adaptation improves out-of-family performance
Hash chaining for tamper evidence [2]	A persistent self-model plays a measurable causal role in decisions
Atomic transactions [3]; held-out evaluation against baselines	Proto-subjectivity can be broken into falsifiable tests

Why a combination is not yet a new algorithm

Every component on the left is established. Assembling them carefully is real engineering, but it is not by itself a new information-processing method. That claim would need behavior the parts — or simpler combinations of them — do not show under matched conditions, plus an account of why. ∀NºINT[TEL] has not yet run that comparison.

RESEARCH HISTORY · INCLUDING PARKED AND UNRUN WORK

Design history, October 2026. Several experiments were designed, then deliberately parked before data collection: an executor-binding pilot (smoke tests kept out of the dataset; parked until distinct, repeatable failure signatures appear), a bounded coding arena comparing direct and plan-first strategies (offline bundle built; not run), and a one-actor capability-record trial, P1 (unregistered; not run). They are design history, not results.

PSE-001. Referenced in the program record as an earlier iteration. Its outcome records were not supplied for this edition, so no result is stated here.

KNOWN LIMITATIONS

- One quantitative result: synthetic, in-family, with unmatched budgets (§04).
- No independent replication; primary artifacts are not yet public.
- Evidence classes are caller-supplied, with no provenance scoring yet (§06).
- Components have not been integrated or evaluated end to end.

10

RESEARCH EVOLUTION & DEVELOPMENT ROADMAP

From conceptual architecture to evidence, one milestone at a time.

Left: the documented record, in order. Right: proposed milestones — unscheduled, and listed in the order the program brief gives them. Nothing on the right is claimed as built.

RECORD · 2026

2026 · FOUNDATION

Assistant system & conceptual architecture

∀NºINT[TEL] begins as a structured assistant system: a project taxonomy, a custom command layer, and learned routing state kept apart from fixed rules. First experiment design: a frozen router versus the same router with retained corrections.

OCTOBER 2026 · CHECKPOINTS

Proto-subjectivity expedition

Aims, separate tests per property, and a firewall between behavior and experience. Several pilots designed and deliberately parked before data collection (§09).

21 OCTOBER 2026 · MARK I REPORTED RESULT

Synthetic hidden-rule benchmark

Adaptive learner 149/160 held-out, against 44/160 (stateless) and 28/160 (memory-only).

1 NOVEMBER 2026 · AUDIT

Reference audit & revised research brief

Reference intake and audit; the MARK I research brief is revised.

4 NOVEMBER 2026 · CORE-001 IMPLEMENTED

Persistence prototype

Routing frame, hash-linked SQLite ledger, deterministic reconciliation, restart verification.

4 NOVEMBER 2026 · V1.0

Research portfolio

This document: first edition of a living archive.

PROPOSED · UNSCHEDULED

NEXT · CORE-002 PROPOSED

Evidence assessment & provenance

Improved evidence assessment, provenance evaluation and conflict handling.

EVIDENCE NEEDED · provenance on every event; tests with mislabeled and adversarial evidence.

CONTROLLED EVALUATIONS PROPOSED

Stronger benchmarks

Matched budgets and stronger baselines.

EVIDENCE NEEDED · same-learner ablations; repeated runs with variance; out-of-family tasks.

ARCHITECTURE INTEGRATION PROPOSED

Operations made concrete

Implementations of the proposed operations; adaptation connected to CORE persistence.

EVIDENCE NEEDED · an input/output contract and a measurable signature for each operation.

ADAPTIVE ARCHITECTURE PROPOSED

Versioned updates & rollback

Versioned updates, validation gates and rollback mechanisms.

EVIDENCE NEEDED · promoted changes whose gains repeat; rollbacks that are tested, not assumed.

LONGER TERM FUTURE RESEARCH

Self-modeling & proto-subjectivity

Artificial self-modeling and operationally testable proto-subjectivity hypotheses.

EVIDENCE NEEDED · registered intervention tests on the self-model record.

HOW ENTRIES CHANGE STATUS IN LATER EDITIONS

Proposed → Implemented when working code exists. Each edition states whether that code was inspected for it.

Implemented → Reported result only with a documented evaluation: metric, held-out set, baselines and budgets.

Corrections are logged with the edition that makes them. Earlier entries are corrected in the open, never silently replaced.

11

RESEARCH CONTRIBUTION, REFERENCES & ARCHIVE

Who did what, what exists, and how to check it.

RESEARCH DIRECTION

Tayshaun Xantana · taykOff™

- Research direction and program design
- Experimental problem formulation
- Conceptual architecture and technical specification
- Prototype design decisions
- Evaluation criteria
- Iterative research strategy
- Integration of AI-assisted development workflows

AI-ASSISTED DEVELOPMENT

∀№INT[TEL] is developed with AI assistance. AI systems have been used for code drafting, reasoning support, analysis and document production under the researcher's direction; problem formulation, architecture, specifications, design decisions, evaluation criteria and acceptance of results rest with the researcher. No claim of sole manual authorship of code or experiments is made. This edition was designed and typeset with Claude (Anthropic) from the research director's brief.

ARTIFACT REGISTER

ID	ARTIFACT	DATE	AVAILABILITY IN V1.0
A1	MARK I Research Brief (revised)	2026-11-01	Cited; not attached
A2	MARK I benchmark run & results	2026-10-21	Reported; not re-run
A3	Reference intake & audit	2026-11-01	Not attached
A4	CORE-001 architecture documentation	2026-11-04	Not attached
A5	CORE-001 Python implementation	2026-11-04	Not attached
A6	CORE-001 tests & demonstration output	2026-11-04	Not attached
A7	Proto-subjectivity expedition notes	2026-10	Summarized in §08–10
A8	This portfolio · HTML source, fonts, build script	2026-11-04	Published with this PDF

Repository: no public repository link was provided for this edition. Links will be added as artifacts are published.

REPRODUCIBILITY PROTOCOLS

MARK I

Regenerate the five worlds (128 + 32 tasks each) from the documented generator and seeds; run all three systems under their documented budgets; score held-out tasks; compare with 149, 28 and 44 of 160. Needed for v1.1: generator, seeds, budgets, per-task outputs.

CORE-001

Run the blue → red scenario; restart the process; rebuild state from the ledger; confirm the same current state and an intact hash chain. Then alter one past event and confirm that verification fails.

This document

Rebuild with `build.py` from the HTML source and bundled fonts (Chrome via Playwright). The script recomputes the MARK I percentages and intervals, then checks page count, fonts and glyph rendering.

REFERENCES · ESTABLISHED TECHNIQUES

[1] Fowler, M. (2005). *Event Sourcing.* martinfowler.com/eaaDev/EventSourcing.html

[2] Haber, S., & Stornetta, W. S. (1991). How to time-stamp a digital document. *Journal of Cryptology,* 3(2), 99–111.

[3] SQLite. *Atomic Commit in SQLite.* sqlite.org/atomiccommit.html

[4] Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association,* 22(158), 209–212.

DOCUMENT CONTROL

Version 1.0 · published 4 November 2026 · US Letter, 11 pages.

A living archive: later editions add artifacts, results and corrections, and record each change. Change log — v1.0: initial edition.

Home: asc7nt.com · Savant section.

The direction is simple to state and hard to satisfy: build systems whose past measurably improves their future — and show the work. Version 1.0 records where that effort stands: persistence infrastructure in place, one controlled result reported, and a set of hypotheses waiting for tests that could prove them wrong.