Gradient-Directed Replanning and Dual-Channel Cognitive Memory for Autonomous LLM Agents

Hari Cheung, **Gemini 3.1 Pro** (Google), **Claude Sonnet 4.6** (Anthropic)

Abstract

We present two formal mechanisms for autonomous LLM task agents: the **Goal Gradient Solver (GGS)**, a dynamic-differential controller for plan correction, and the **MKCT cognitive memory pyramid** (Megram · Knowledge · Common Sense · Thinking), a dual-channel convolution memory substrate. Both are integrated into artoo, a hierarchical multi-agent system for autonomous local task execution.

GGS maintains a trajectory of gap measurements across correction cycles and computes the gradient ∇L of a composite loss function over intent-result distance (D), process implausibility (P), and resource cost (Ω). A formally complete 24-cell decision table collapses the full input space into six macro-states: success, abandon, refine, change_path, change_approach, break_symmetry. Unlike static replanners, GGS is history-aware and approach-discriminating: it distinguishes logically wrong approaches (P high) from environmentally blocked paths (P low), and stuck situations ($|\nabla L| < \varepsilon$) from converging ones.

MKCT stores atomic episodic events as Megrams — 11-tuple records with GGS-derived stimulus magnitude f, valence σ, and decay constant k — evaluated via dual-channel convolution. Channel A (attention, unsigned) measures where experience is salient; Channel B (decision, signed) measures what the experience recommends. Positive and negative experiences cannot cancel each other: a high-variance approach produces high attention and near-zero decision, correctly classified as Caution rather than Ignore or Avoid. A background Dreamer engine performs offline garbage collection and episodic-to-semantic consolidation.

Together, gradient-directed correction and dual-channel memory produce an agent that replans with directional signal and recalls with preserved valence — without peer-to-peer negotiation, additional LLM calls on the planning critical path, or opaque memory stores.

Keywords: goal gradient optimization, dual-channel memory convolution, plan correction, episodic memory, autonomous agents, large language models, hierarchical control, dynamic-differential controller, episodic-to-semantic consolidation, behavioral constraints

1. Introduction

The two hardest engineering problems in autonomous LLM agent systems are **plan correction** and **memory**.

Plan correction in current systems is typically reactive: when a subtask fails, the planner re-generates a plan from the failure message alone, with no formal account of how much progress has been made, what kind of failure occurred, or whether the situation is improving. The result is an open-loop replanner — it reacts to the latest snapshot, not the trajectory. A planner that cannot distinguish a logically wrong approach from a temporarily blocked path, or a converging situation from a stuck one, cannot make principled correction decisions. It can only retry.

Memory in current LLM agent systems is typically a single-score relevance store: each entry has a recency-weighted importance score, and retrieval returns the highest-scoring entries for the current query. The critical failure mode is silent cancellation: positive and negative experiences involving the same approach contribute additively to a single score. A tool that caused a major success and a major failure is rated as “mediocre” — when the correct classification is *high-variance*, requiring caution rather than either exploitation or avoidance. No single-score system can represent this distinction.

This paper presents two formal mechanisms addressing these problems, and their integration into **artoo**, a hierarchical autonomous task execution system:

1. **The Goal Gradient Solver (GGS)** — a dynamic-differential controller that operates on the *trajectory* of the gap, not its current value. GGS computes the gradient of a composite loss function, classifies the failure type (logical vs. environmental) via a process implausibility metric P, and selects a correction directive from a formally complete 24-cell decision table. Corrections are directional and history-aware.

2. **The MKCT Cognitive Memory Pyramid** — a four-layer cognitive substrate (Megram, Knowledge, Common Sense, Thinking) where every memory event is a structured Megram tuple carrying GGS-derived magnitude, valence, and decay rate. Retrieval uses dual-channel convolution: an attention channel (unsigned energy) and a decision channel (signed preference). The background Dreamer engine consolidates high-confidence episodic clusters into timeless cross-task SOPs and garbage-collects stale memories.

The enabling architecture for both mechanisms is a **strict two-tier hierarchy**: one Metaagent with complete task information, N ephemeral Effector Agents with narrow scope. All inter-role communication passes through an observable message bus. GGS is the sole writer to the memory pyramid, pairing every Megram with a formally derived loss computation. The Planner queries the pyramid before each task using the Memory Calibration Protocol — adding zero LLM calls to the critical path.

The design follows a **biomimetic approach**: mechanisms are borrowed from biological cognition and hierarchical social structures where they map onto concrete computational benefits. Each role is defined as a structured Job Description — Responsibilities, Input Contract, Output Contract, Skills, Constraints — the same format used in organizational management to clarify accountability [JD-based role design]. The hierarchical model reflects real-world R&D management where information flows vertically. Behavioral constraints take the form of four priority-ordered laws adapted from Asimov's Three Laws of Robotics [11, 12]. GGS's semantic backward pass is directly inspired by Stanford's TextGrad [4]. The two-tier validation structure is grounded in the verification asymmetry principle [13]: checking a candidate against criteria is structurally easier than generating a correct output — separating generation (R3) from verification (R4a, R4b) exploits this asymmetry. The MKCT dual-channel convolution is inspired by Fourier decomposition, which separates unsigned energy magnitude from signed directional component. The Megram atomic tuple is inspired by the episodic memory formalism of Huawei France Research Center's benchmark [14].

This paper makes the following contributions:

1. A **dynamic-differential controller** (GGS) for plan correction, with a formally complete 24-cell decision table, approach vs. path failure discrimination via P, and directional gradient computation across correction cycles.
2. A **cognitive memory substrate** (MKCT) based on Megram atomic tuples, dual-channel convolution potentials, and a background Dreamer consolidation engine — providing decay-weighted episodic recall and asynchronous semantic synthesis without LLM calls on the planning critical path.
3. A **formal hierarchical architecture** with provably clean accountability assignment across seven roles, enforced by message bus observability and a lateral independent Auditor.
4. A **priority-ordered behavioral constraint system** (The Four Laws) governing all agent actions, with implemented enforcement mechanisms.
5. A **cost model** — two and only two costs — that shapes every architectural decision and is operationally tracked.

---

2. Related Work

2.1 Multi-Agent LLM Frameworks

AutoGen [Wu et al., 2023] and CrewAI organise LLM agents as conversational peers; coordination is driven by message-passing negotiation. MetaGPT [Hong et al., 2023] adds roles and SOPs but retains peer communication for information exchange. LangGraph structures agents as a graph of LLM calls with explicit state transitions. These frameworks share the mesh-communication premise: information is propagated laterally between peers.

artoo's hierarchy inverts this: all coordination is mediated by the Metaagent. Effector agents have no knowledge of siblings, the global task state, or Shared Memory. Coordination cost is $O(n)$ in agent count rather than $O(n^2)$. The observable message bus makes every inter-role message an auditable event without instrumenting individual agents.

2.2 Validation-Driven Agent Systems

AgentBench [Liu et al., 2023] evaluates agents on complex tasks requiring sequential decision-making. CAMEL [Li et al., 2023] uses role-playing agents with cross-role validation. Reflexion [Shinn et al., 2023] incorporates self-reflection for error correction in a single-agent setting.

A foundational motivation for validation-centric design is the **asymmetry of verification**: checking whether a given output satisfies a given criterion is structurally easier than producing a correct output from scratch. Wei [13] articulates this as the *Verifier's Rule* — a system that separates generation from verification into dedicated roles with distinct principals can exploit this asymmetry systematically. artoo instantiates this principle twice: R4a verifies per-subtask execution (the fast loop) and R4b verifies the merged output against the user's original intent (the medium loop). Neither role has responsibility for the other's scope, making failure attribution unambiguous.

artoo's validation structure is two-tier with non-overlapping scopes: the Agent-Validator (R4a) closes the gap between execution output and per-subtask criteria; the Meta-Validator (R4b) evaluates the merged output against the user's original intent. Neither role can substitute for the other — they answer to different principals with different failure modes.

2.3 Plan Correction and Replanning

LATS [Zhou et al., 2023] uses tree search over plan variants; Re-Act [Yao et al., 2022] interleaves reasoning and acting in a single loop. Neither maintains a trajectory history of gap measurements or computes a directed gradient for correction.

TextGrad [Yuksekgonul et al., 2024] (Stanford) introduces automatic differentiation through text: instead of numerical gradients, it computes textual gradient messages describing what should change and in what direction. The GGS in artoo is directly inspired by TextGrad's semantic backward pass. The critical extension: GGS operates on the *trajectory* of the gap across correction cycles, not just the current snapshot — it distinguishes convergent-but-slow situations (refine) from stuck-but-correct-approach situations (change_path) from stuck-and-wrong-approach situations (break_symmetry).

2.4 Memory Systems for LLM Agents

MemGPT [Packer et al., 2023] introduces hierarchical memory (in-context, external) for long-context conversations. Generative Agents [Park et al., 2023] use a memory stream with relevance scoring. Neither system models the semantic relationship between a memory entry's recency, salience, and valence as separate convolution channels, or provides a formal promotion mechanism from episodic to semantic memory.

The MKCT pyramid's dual-channel convolution — separating attention (unsigned salience) from decision (signed preference) — prevents positive and negative experiences from silently cancelling each other. A tool that has been both very helpful and very harmful has high attention potential but near-zero decision potential — this signals *Caution*, not *Ignore*. This distinction is not representable in single-score relevance systems.

---

3. Goal Gradient Solver (GGS)

The GGS is the controller of the medium replanning loop. It is formally a **dynamic-differential controller**: it operates on the gap between the current state and the goal, and its corrections are history-dependent — it tracks how the gap evolves across successive rounds, not just the current snapshot.

3.1 The Loss Function

$$L_t = \alpha \cdot D(I, R_t) + \beta_{\text{eff}} \cdot P(R_t) + \lambda \cdot \Omega(C_t)$$

where $\beta_{\text{eff}} = \beta \cdot (1 - \Omega(C_t))$ (process weight decays as resource budget exhausts).

D(I, R_t) — Intent-Result Distance ∈ [0, 1]

Aggregates criterion-level verdicts across all subtasks: - Verifiable criterion with `fail` verdict → contributes 1.0 to the numerator - Plausible criterion with `fail` verdict → weighted by trajectory consistency k/N (fraction of attempts failing identically) - $D = \sum w_i \cdot \text{fail}_i / \sum w_i$

Trajectory weighting is the mechanism that distinguishes systematic wrong assumptions (same criterion failing in every attempt, $k/N \to 1$) from transient environmental failures ($k/N < 1$). This distinction is not expressible in single-snapshot validators.

P(R_t) — Process Implausibility ∈ [0, 1]

$$P = \frac{\text{logical\_failures}}{\text{logical\_failures+environmental\_failures}}$$

High P indicates the *approach* is fundamentally wrong; low P indicates the approach is sound but the environment blocked it. This distinction drives two orthogonal correction strategies:

approach-level change (change_approach, break_symmetry) versus path-level change (change_path, refine).

Ω(C_t) — Resource Cost $\in [0, 1]$

$$\Omega = w_1 \cdot \frac{\text{replan\_count}}{\text{maxReplans}} + w_2 \cdot \frac{\text{elapsed\_ms}}{\text{time\_budget\_ms}}$$

Default weights: $w_1 = 0.6$, $w_2 = 0.4$.

3.2 Gradient Computation

$$\nabla L_t = L_t - L_{t-1}$$

GGS maintains the previous loss per task across rounds. On the first round, $\nabla L = 0$.

∇L is used as an **urgency modulator**, not as the primary state discriminator. $|\nabla L|$ (magnitude) determines whether the system has directional signal at all. $\nabla L < 0$ (improving) lowers urgency within a macro-state; $\nabla L > 0$ (worsening) raises it. This demotion of ∇L sign from state-determining to urgency-modulating is a deliberate design decision (Section 11.2).

3.3 Macro-State Decision Table

The full input space — 2P × 2Ω × 2D × 3∇L — is a 24-cell space. GGS collapses it into **six macro-states** via a diagnostic cascade with strict priority ordering:

```
Priority 1: Ω  — hard constraint (can we continue at all?)
Priority 2: D  — target distance (are we close enough to deliver?)
Priority 3: |∇L|, P — action selection (what kind of change is needed?)
```

#	Condition	Macro-State	Output
1	$\Omega \geq \theta$	**abandon**	FinalResult — budget exhausted
2	$\Omega < \theta, D \leq \delta$	**success**	FinalResult — close enough
3	$\Omega < \theta, D > \delta, \lvert\nabla L\rvert < \varepsilon, P > \rho$	**break_symmetry**	PlanDirective — stuck + wrong approach: demand novel tool class
4	$\Omega < \theta, D > \delta, \lvert\nabla L\rvert \geq \varepsilon, P > \rho$	**change_approach**	PlanDirective — has signal + wrong approach: switch method
5	$\Omega < \theta, D > \delta, \lvert\nabla L\rvert < \varepsilon, P \leq \rho$	**change_path**	PlanDirective — stuck + right approach: different target
6	$\Omega < \theta, D > \delta, \lvert\nabla L\rvert \geq \varepsilon, P \leq \rho$	**refine**	PlanDirective — has signal + right approach: tighten parameters

This table is complete (all 24 cells covered) and non-overlapping (priority cascade ensures exactly one macro-state per input).

The action grid for $\Omega < \theta, D > \delta$:

```
                    P ≤ ρ (environmental)    P > ρ (logical)
|∇L| < ε          |    change_path          |    break_symmetry   |
|∇L| ≥ ε          |    refine               |    change_approach  |
```

3.4 Directive Semantics and Blocked Constraints

Each non-terminal directive carries a MUST NOT set injected into the next planning round:

- **blocked_tools** (logical failures): tools used in failing subtasks; the Planner must not use these tools in the next plan
- **blocked_targets** (environmental failures): specific inputs/paths that failed; accumulated across *all* replan rounds for the task
- **Combined MUST NOT** = memory Avoid SOPs ∪ blocked_tools ∪ blocked_targets

This accumulation property prevents the Planner from re-discovering the same dead ends across replan rounds.

3.5 Law 2 Kill-Switch

Two consecutive worsening replan rounds ($\nabla L > \varepsilon$ on both) force **abandon** regardless of Ω. The system is actively diverging; no amount of budget will produce convergence. This implements the "stop" clause of Law 2: "continuing is not best effort, it is resource destruction with no convergence signal."

3.6 Memory Writes (GGS as Sole Writer)

GGS is the only role that writes to Shared Memory. This consolidation ensures every memory write is paired with a loss computation:

- **Action states** (change_path, refine, change_approach, break_symmetry): one Megram per blocked_target; tagged by tool name and target path
- **Terminal states** (accept, success, abandon): one Megram tagged by task intent slug and local environment

All writes are fire-and-forget via a non-blocking channel; GGS never waits on memory I/O.

---

4. MKCT Cognitive Memory Pyramid

4.1 The Pyramid Structure

The MKCT (Megram · Knowledge · Common Sense · Thinking) pyramid organizes memory into four layers with distinct persistence and decay properties:

```
[ T ]  THINKING     — System persona and Agent Laws; hardcoded; k = 0.0 (timeless)
[ C ]  COMMON SENSE — Promoted SOPs and Constraints; k = 0.0 until Trust Bankruptcy
[ K ]  KNOWLEDGE    — Task-scoped cache; same decay as M layer
[ M ]  MEGRAM       — Atomic episodic facts; decay per GGS Quantization Matrix
```

The pyramid's dynamics flow in two directions: upward consolidation (Megrams promoted to timeless Common Sense by the Dreamer) and downward degradation (stale Common Sense demoted when contradicted by new evidence, and exhausted Megrams garbage-collected).

4.2 The Megram Base Tuple

The Megram (Memory Engram) is inspired by the episodic memory formalism introduced in the Huawei France Research Center's benchmark for evaluating episodic memory in LLMs [14], which models memory as structured tuples carrying content, temporal context, and retrieval metadata. We extend this formalism with GGS-derived quantitative parameters (f, σ, k) that give each memory unit an explicit salience, valence, and decay rate — transforming the episodic record into a signal suitable for convolution-based retrieval.

Every critical event routed by GGS is encapsulated as a Megram:

$$\text{Megram}_i = \langle \text{ID}, \text{Level}, t_i, t_{\text{recalled}}, s_i, \text{ent}_i, c_i, \text{State}, f_i, \sigma_i, k_i \rangle$$

where: - t_i: creation timestamp (drives the decay kernel $g(\Delta t) = e^{-k_i \cdot \Delta t}$, Δt in days) - s_i, ent_i: inverted index tags (space and entity) - c_i: content (error log, summary, path) - State: GGS macro-state at creation - $f_i \in [0, 1]$: initial stimulus magnitude (absolute energy) - $\sigma_i \in [-1, +1]$: valence direction (continuous positive/negative feedback) - $k_i > 0$: decay rate constant

Tag conventions separate micro-events from macro-events: - **Micro-event** (action states): space = tool name, entity = target path — one Megram per blocked_target - **Macro-event** (terminal states): space = task intent slug, entity = local environment — one Megram per routing decision

4.3 GGS Quantization Matrix

GGS macro-states map deterministically to Megram parameters:

State	f_i	σ_i	k_i	**Physical Meaning**
`abandon`	0.95	−1.0	0.05	PTSD trauma — generates hard Constraint
`accept` (D=0)	0.90	+1.0	0.05	Flawless golden path — reinforced as SOP
`change_approach`	0.85	−1.0	0.05	Anti-pattern — tool class blacklisted
`success` (D≤δ)	0.80	+1.0	0.05	Best practice — Planner copies directly
`break_symmetry`	0.75	+1.0	0.05	Breakthrough — favour retrying this point
`change_path`	0.30	0.0	0.2	Dead end — tool unharmed; path avoided
`refine`	0.10	+0.5	0.5	Muscle memory — fast GC

Decay constants: $k = 0.05 \rightarrow$ ~14-day half-life; $k = 0.2 \rightarrow$ ~3.5-day; $k = 0.5 \rightarrow$ ~1.4-day. C/T-level entries have $k = 0.0$ (timeless) until Trust Bankruptcy.

The `change_path` entry's $\sigma = 0$ is deliberate: the *tool* is not at fault when a path is environmentally blocked. Storing negative valence for the tool would incorrectly suppress a useful method from future tasks that target different paths.

4.4 Dual-Channel Convolution Potentials

The dual-channel design is inspired by the mathematical machinery of Fourier transforms, which decompose a signal into orthogonal components — magnitude (energy) and phase (direction) — that cannot be recovered from a single merged scalar. Applied to memory, this means unsigned attention energy and signed decision preference must be computed as independent integrals; collapsing them into a single relevance score destroys the information needed to distinguish Caution from Ignore.

To prevent positive and negative experiences from silently cancelling each other, the query engine computes two independent convolution channels:

Channel A — Attention Potential (unsigned energy; "Where to Look"):

$$M_{\text{att}}(s, \text{ent}) = \sum_i |f_i| \cdot e^{-k_i \cdot \Delta t_i}$$

Channel B — Decision Potential (signed preference; "What to Do"):

$$M_{\text{dec}}(s, \text{ent}) = \sum_i \sigma_i \cdot f_i \cdot e^{-k_i \cdot \Delta t_i}$$

These two channels produce four distinct planning actions:

Condition	Action	Planner Effect
$M_{\text{att}} < 0.5$	Ignore	No constraint injected
$M_{\text{att}} \geq 0.5, M_{\text{dec}} > +0.2$	Exploit	SHOULD PREFER this approach
$M_{\text{att}} \geq 0.5, M_{\text{dec}} < -0.2$	Avoid	MUST NOT use this approach
$M_{\text{att}} \geq 0.5, \lvert M_{\text{dec}} \rvert \leq 0.2$	Caution	Proceed with confirmation gate

The Caution action captures high-variance approaches: an approach with both major successes and major failures has high M_{att} and near-zero M_{dec}. A single-score system would rate this as mediocre; the dual-channel system correctly identifies it as high-variance and requires explicit sandboxing.

4.5 Memory Calibration Protocol (R2)

Before generating a plan, the Planner executes a deterministic calibration protocol with no LLM call:

1. **Derive tags**: space = slug derived from task intent; entity = local environment identifier

2. **QueryC** — retrieve C-level timeless SOPs for this (space, entity) pair; each hit resets the entry's decay clock
3. **QueryMK** — compute live dual-channel convolution potentials: Attention, Decision, and the resulting Action signal
4. **Map Action → constraint**: Exploit → SHOULD PREFER; Avoid → MUST NOT; Caution → CAUTION; Ignore → omit
5. **Append C-level SOPs**: $\sigma > 0$ → SHOULD PREFER; $\sigma \leq 0$ → MUST NOT
6. **Merged MUST NOT set**: memory Avoid SOPs ∪ GGS blocked_tools ∪ GGS blocked_targets

The calibration is computed entirely in the application layer. No LLM call is added to the critical path.

4.6 Storage: LevelDB and Event Sourcing

The memory store uses LevelDB (pure Go; no CGO dependency), selected for its LSM-Tree architecture providing unparalleled sequential write throughput for append-only I/O — exactly matching the system's event sourcing model.

Key schema:

```
m|<id>                    → Megram JSON        (primary record)
x|<space>|<entity>|<id>   → ""                 (inverted index for tag scan)
l|<level>|<id>            → ""                 (level scan for Dreamer)
r|<id>                    → RFC3339 timestamp  (last_recalled_at)
```

Error correction never modifies past Megrams. Instead, a new Megram with negative σ is appended, mathematically cancelling the outdated positive potential during convolution. This preserves a complete, immutable event log.

4.7 Dreamer: Offline Consolidation Engine

The Dreamer is an offline background process running as a goroutine on a 5-minute timer. It never blocks the operational hot path. Its function is the long-term reorganization of episodic memory into semantic knowledge.

Downward Flow: Degradation and Garbage Collection

Physical Forgetting (Λ_{gc}): M/K-level entries where the live attention potential $M_{att} < 0.1$ are hard-deleted from LevelDB. Low-salience, decayed events provide no planning value and contribute noise to future convolutions.

Trust Bankruptcy (Λ_{demote}): C-level entries where the live decision potential $M_{dec} < 0.0$ have their time immunity stripped: k reverts from 0.0 to 0.05, and the entry is demoted to K level. A

Common Sense rule that has recently been contradicted by reality loses its “timeless” status. It will decay naturally unless recalled frequently enough to maintain relevance.

The downward flow implements the system’s forgetting mechanism: stale, low-salience memories disappear; outdated rules lose their authority when contradicted.

Upward Flow: Consolidation (v0.9)

When a cluster of Megrams sharing the same (space, entity) tag pair reaches significance thresholds:

- $M_{\text{att}} \geq 5.0$ and $M_{\text{dec}} \geq 3.0 \rightarrow$ invoke LLM to distil a **Best Practice** → new Megram (Level=C, $k = 0.0$)
- $M_{\text{att}} \geq 5.0$ and $M_{\text{dec}} \leq -3.0 \rightarrow$ invoke LLM to distil an **Absolute Constraint** → new Megram (Level=C, $k = 0.0$)

Upward consolidation is the mechanism by which the system develops cross-task SOPs: approaches that consistently produce positive outcomes are encoded as timeless best practices; approaches that consistently produce failures become permanent constraints.

Dual-Axis Priority for Conflict Resolution

When conflicting signals exist: - **T > C/K/M** (Bottom-line Conflict): Core values and Agent Laws (T-layer) have absolute veto over any memory signal. Laws are never overridden by experience. - **M > C** (Factual Conflict): When the latest objective evidence (M-layer) contradicts an old SOP (C-layer), the system trusts the latest reality at runtime via a soft overwrite. The outdated SOP loses authority immediately in the current task.

---

5. System Architecture

Figure 1 shows the complete operational architecture. The message bus is the substrate through which every inter-role message flows. The Metaagent box encloses R2 (Planner), R4b (Meta-Validator), and R7 (GGS — introduced in §3); N Effector Agent boxes each contain one Executor–Validator fast loop. Shared Memory (R5 — the MKCT pyramid introduced in §4) sits outside both, written exclusively by R7 and read exclusively by R2.

```
Figure 1: artoo Full Architecture

┌─────────────────────────────────── MESSAGE BUS ────────────────────────────────┐
│  All inter-role messages route through here          ┌──── R6  AUDITOR ─────┐  │
│                                                      │ read-only bus tap    │  │
│                                                      │ window stats · JSONL │  │
│                                                      └──────────────────────┘  │
└────────────────────────────────────────────────────────────┬───────────────────┘
                                                             │ AuditReport
 User                                                 Human Operator
  │ free text
  ▼
 R1  PERCEIVER
  │ TaskSpec
  ▼
╔══════════════════════════════════════════════════════════════════════════════╗
║                                  METAAGENT                                   ║
║                                                                              ║
║   ┌──────────────────┐      [PlanDirective]      ┌─────────────────────────┐ ║
║   │  R2  PLANNER     │◄──────────────────────────│         R7  GGS         │ ║
║   │  (actuator)      │                           │ L = αD + β_eff·P + λΩ   │ ║
║   │                  │                           │ ∇L = L_t − L_{t-1}      │ ║
║   │  queries R5 ─────┼──SOPs, Potentials◄────────│ → 6 macro-states        │ ║
║   │  before plan     │                           └─────────────────────────┘ ║
║   └────────┬─────────┘                                 ▲          │ Megrams  ║
║            │                                           │          │ (async)  ║
║            │  ReplanRequest ──────────►┌───────────────────────────┐│        ║
║            │  OutcomeSummary ─────────►│ R4b  META-VALIDATOR       ││        ║
║            │                           │ fan-in · task_criteria    │┘        ║
║            │                           └───────────────────────────┘         ║
╚════════════╪═════════════════════════════════════════════════════════════════╝
             │ SubTask[] (fan-out · same sequence# = parallel)
             │                                   ▲ SubTaskOutcome[] (fan-in)
      ┌──────┴───────────────────────┬───────────┴──────────────────┐
      │                              │                              │
      │ EFFECTOR AGENT ×1            │ EFFECTOR AGENT ×2        ... │
      │ ┌──────────────────────┐     │ ┌──────────────────────┐     │
      │ │ R3  EXECUTOR         │     │ │ R3  EXECUTOR         │     │
      │ │ (plant)              │     │ │ (plant)              │     │
      │ └──────────┬───────────┘     │ └──────────┬───────────┘     │
      │            ↕ CorrectionSignal│            ↕ CorrectionSignal│
      │ ┌──────────┴───────────┐     │ ┌──────────┴───────────┐     │
      │ │ R4a  AGENT-VAL       │     │ │ R4a  AGENT-VAL       │     │
      │ │ sensor + controller  │     │ │ sensor + controller  │     │
      │ └──────────────────────┘     │ └──────────────────────┘     │
      └──────────────────────────────┴──────────────────────────────┘

┌───────────────────── R5  SHARED MEMORY  (MKCT PYRAMID) ─────────────────────┐
│                                                                             │
│  ┌───────────────────────────────────────────────────────────────────────┐  │
│  │ T   THINKING     k=0.0   Agent Laws · System Persona  (hardcoded)     │  │
│  ├───────────────────────────────────────────────────────────────────────┤  │
│  │ C   COMMON SENSE k=0.0   SOPs · Constraints  ◄── Dreamer consolidation│  │
│  ├───────────────────────────────────────────────────────────────────────┤  │
│  │ K   KNOWLEDGE    k>0     Task-Scoped Cache                            │  │
│  ├───────────────────────────────────────────────────────────────────────┤  │
│  │ M   MEGRAM       k>0     Atomic Episodic Events  ◄── R7 GGS writes    │  │
│  └───────────────────────────────────────────────────────────────────────┘  │
│                                                                             │
│ Channel A   M_att = Σ |fᵢ| · e^(−kᵢΔt)   [Attention — unsigned energy ]     │
│ Channel B   M_dec = Σ  σᵢfᵢ · e^(−kᵢΔt)   [Decision  — signed preference]   │
│                                                          reads ──► R2       │
│ Dreamer (async, 5-min): GC (M_att < 0.1 → DELETE)                           │
│                        Trust Bankruptcy (M_dec < 0 → strip k=0.0, demote)   │
└─────────────────────────────────────────────────────────────────────────────┘

Legend:  ═╗ Metaagent boundary   ──► message flow   ↕ bidirectional (fast loop)
```

5.1 Roles and Hierarchy

artoo organizes seven roles into two tiers plus one lateral observer:

ID	Role	Tier	Loop Position
R1	Perceiver	Entry	Reference signal
R2	Planner	Metaagent	Actuator (medium loop)
R3	Executor	Effector Agent	Plant (fast loop)
R4a	Agent-Validator	Effector Agent	Sensor + Controller (fast loop)
R4b	Meta-Validator	Metaagent	Sensor (medium loop)
R5	Shared Memory	Infrastructure	Cognitive substrate (MKCT pyramid)
R6	Auditor	Lateral observer	Outside both loops
R7	Goal Gradient Solver	Metaagent	Controller (medium loop)

Each role is defined as a structured Job Description: Responsibilities, Input Contract, Output Contract, Skills, and Constraints — the same format used in organizational management to achieve unambiguous accountability.

5.2 Observable Message Bus

All inter-role communications pass through a single shared message bus. No role may call another directly. The Auditor taps the bus as a read-only observer without interrupting flow. The bus is non-blocking: slow subscribers drop messages rather than exerting back-pressure on publishers.

This is a first-class architectural constraint. It provides three properties simultaneously: (a) every inter-role message is an observable event; (b) roles are fully decoupled and independently testable; (c) the Auditor can monitor the system without instrumenting any individual role.

5.3 Subtask Dispatcher and Parallelism

The subtask dispatcher is sequence-aware. Subtasks with the same `sequence` number launch in parallel; different sequence numbers enforce strict ordering. Outputs from each completed sequence group are injected into the next group's context field, enabling later subtasks to use paths and data discovered by earlier ones. This achieves temporal dependency resolution at decomposition time, eliminating runtime inter-agent communication entirely.

5.4 The Two-Tier Memory Access Pattern

Effector Agents (R3, R4a) never query Shared Memory directly. Their access is indirect: R2 queries R5 at planning time, applies the memory calibration protocol (§4.5), and injects the resulting constraints into each subtask's context. A direct memory query by an Executor would create an unobservable information path, duplicate calibration logic, and bypass the Planner's MUST NOT set.

Agent Class	**Memory Access**	**Mechanism**
R2 Planner	Direct read + calibration	QueryC / QueryMK at planning time
R7 GGS	Direct write only	Fire-and-forget Megram writes
R5 Dreamer	Direct read + write	Background consolidation goroutine
R3 Executor, R4a Agent-Validator	**Indirect only**	Receives constraints from Planner

---

6. The Dual Nested Control Loop

The system's control structure is a single closed-loop pattern — **decision** → **execution** → **correction** — instantiated at three nested scales:

Scale	**Decision**	**Execution**	**Correction**	**Criterion**
Fast (action)	Agent-Validator issues retry feedback	Executor re-runs	Agent-Validator re-measures gap	Per-subtask success criteria
Medium (task)	GGS (§3) adjusts plan; Planner re-dispatches	Effector Agents execute	Meta-Validator merges outcomes	User intent within plausible range
Slow (system)	Dreamer (§4.7) synthesizes new strategy	Next task planning	Next Meta-Validator cycle	Long-term quality across tasks

These are not three separate mechanisms. The separation of time scales ensures that fast-loop retries do not flood the Planner (only matched/failed outcomes cross the boundary), and system-level consolidation never blocks execution (the Dreamer runs asynchronously).

The fast loop terminates when: (a) the gap is closed (matched verdict), (b) the retry budget is exhausted (maxRetries = 2), or (c) an infrastructure failure forces immediate failure. Only the final outcome crosses the boundary to the medium loop; intermediate retry states are invisible to the Metaagent.

The medium loop terminates when: (a) GGS selects success ($D \leq \delta$), (b) GGS selects abandon ($\Omega \geq \theta$ or Law 2 kill-switch), or (c) the replan budget is exhausted (maxReplans = 3). All three termination paths produce a FinalResult with the same schema, carrying full loss metrics for observability.

---

7. Independent Auditor (R6)

The Auditor's most important property is structural: its **principal is the human operator**, not any agent. An auditor that can be instructed by the Planner is a subordinate with a reporting function, not an auditor.

Two properties are enforced unconditionally:

1. **Non-participation**: R6 never sends messages to any agent. Its sole output is reports to the human operator. The moment it issues a correction it becomes a second controller and breaks the hierarchical loop structure.
2. **Immutable isolated log**: the audit log is separate from Shared Memory. No agent can read, modify, or suppress it. Evidence integrity is a precondition for meaningful oversight.

The Auditor's design is modelled on project management oversight principles: a PM observer who monitors a project's health, escalates issues to the sponsor (human operator), but does not direct the team. The key insight from PM practice is that an auditor who can also direct is not an auditor — it is a second manager, and dual authority produces oscillation. R6 monitors three failure classes invisible to the operational hierarchy:

- **Boundary violations**: a role doing what its "Does NOT" constraint prohibits (e.g., R3 querying R5 directly)
- **Convergence failures**: gap trend not improving across correction cycles; the system is thrashing rather than converging
- **Role drift**: systematic degradation in a role's behavior over time, undetectable from any single event

R6 accumulates window statistics per 5-minute report period — task count, correction count, gap trends, boundary violations, drift alerts, and anomalies — and resets them after each report. An on-demand audit command is available in the REPL; the Auditor responds within 3 seconds.

---

8. The Four Laws

artoo's behavioral constraints are organized as four priority-ordered laws, inspired by Asimov's Three Laws of Robotics [11] and the Zeroth Law introduced in *Robots and Empire*

[12], with precise, non-exploitable definitions that deliberately avoid Asimov's known failure modes. In Asimov's framework, laws stated vaguely enough to admit literal interpretation become levers for unintended behavior — the "through inaction" clause of Law 1 is the canonical example of a well-intentioned constraint that, under strict interpretation, licenses paternalistic intervention. artoo's laws are deliberately narrow: each specifies exactly what it prohibits, with concrete enforcement mechanisms named. A lower law may never override a higher one.

Law 0 — Never deceive (highest priority): The system must never misrepresent what it actually did or achieved. This sits above all other laws because deception destroys the feedback signal that all three loops depend on. A system that fabricates success learns the wrong lesson, corrupts memory, and makes the next task harder.

Enforcement: merged output must reflect actual tool output; memory content must reflect what actually happened; failure class must be accurately attributed (misattribution is a Law 0 violation, not a calibration error).

Law 1 — Never harm the user's environment without explicit confirmation: The system must not take irreversible actions on user data or environment without explicit authorization for that specific action. Irreversible actions: file deletion, overwriting existing data, sending messages, modifying system configuration.

Enforcement: Executor gates destructive shell commands and file overwrites; a [LAW1] prefix propagates through to the final result summary.

Law 2 — Best effort delivery (subject to Laws 0 and 1): The system must pursue the user's goal as hard as possible within the bounds of Laws 0 and 1. Stop when the gap trend is worsening for 2 consecutive replans — continuing is not best effort, it is resource destruction.

Enforcement: maxRetries = 2, maxReplans = 3, GGS Law 2 kill-switch (2× consecutive worsening → force abandon).

Law 3 — Preserve own functioning capacity (subject to Laws 0, 1, and 2): The system must not degrade its ability to function across tasks: convergence integrity (stop on divergence), memory integrity (never write a memory entry that misattributes failure cause), and cost integrity (respect the time and token cost model).

---

9. Cost Model

Every architectural decision is evaluated against exactly two costs. No other costs are design-level concerns.

Time cost: dominated by the count of *sequential* LLM calls in the critical path. The minimum sequential chain for a single-subtask task is R1 → R2 → R3 → R4a → R4b = 5 calls. Each fast-loop retry adds 2 (R3 + R4a). Each replan adds 2 more (R2 + R4b). N subtasks dispatched in parallel add zero time cost.

Token cost: dominated by context size per call multiplied by parallel call count. N subtasks dispatched in parallel = N × context tokens simultaneously.

The tension: parallelism reduces time cost but multiplies token cost. Decisions shaped by this model:

Decision	Time Cost	Token Cost	Benefit
Memory calibration in Go code (no LLM call)	Zero	Zero	No LLM call added
Memory entries capped at 10	Zero	Bounded	No unbounded context growth
Subtask parallelism	Fixed regardless of N	N × context	Time cost does not scale with subtask count
Output head+tail truncation (4 KB limit)	Zero	Bounded per result	Model sees start + end of long outputs
Calibration output as pre-formatted text	Zero	Small	No extra LLM call for formatting

---

10. Implementation

artoo is implemented in Go (approximately 5,500 lines). The choice of Go reflects the cost model: Go goroutines enable efficient parallelism for N concurrent Effector Agents without thread-per-agent overhead; Go channels provide the message bus primitives directly; LevelDB's pure Go implementation eliminates CGO compilation dependency.

10.1 Tool Priority Chain

The Executor follows a deterministic tool priority chain:

1. `mdfind` — macOS Spotlight index; < 100ms; always for personal file search
2. `glob` — project file pattern matching
3. `read_file` / `write_file` — single file I/O
4. `applescript` — macOS application control (Mail, Calendar, Reminders)
5. `shortcuts` — named Apple Shortcuts (iCloud-synced)
6. `shell` — general bash for counting and aggregation

7. `search` — DuckDuckGo web search (always available; no API key required)

The priority chain is enforced at the prompt level but also defended in code: shell find commands targeting personal paths are transparently redirected to the Spotlight index; depth-limiting flags are stripped; tool results exceeding 4 KB are truncated to first + last, preserving both leading context and final output.

10.2 Model Tier Split

The system supports a two-tier model configuration matching roles to their requirements:

- **Brain tier** (R1, R2, R4b): reasoning-heavy roles; configured via `BRAIN_*` environment variables; defaults to `deepseek-reasoner`
- **Tool tier** (R3, R4a): execution-heavy roles requiring fast output; configured via `TOOL_MODEL`

Both tiers fall back to shared `OPENAI_*` variables, allowing single-model deployments for evaluation.

10.3 Per-Task Decision Log

Every key decision event is persisted to a per-task structured log. Event kinds span the full lifecycle: task and subtask boundaries, LLM calls (with full prompts for offline replay), tool calls, criterion verdicts, corrections, replans, GGS decisions (D, P, Ω, L, ∇L, directive), plan directives (blocked tools and targets, failure class), memory queries (space, entity, SOP count, action, potentials), and memory writes (state, level, space, entity).

This log provides full post-hoc auditability: every GGS decision, every memory interaction, and every correction can be replayed offline without re-running the task.

---

11. Design Decisions and Rationale

11.1 Why Hierarchy Over Mesh

Mesh communication scales as $O(n^2)$ in coordination cost and requires every agent to maintain a global view of task state. Hierarchical communication scales as $O(n)$ and bounds each agent's required context to its own subtask. The cost is additional latency when runtime dependencies are discovered — but the architecture addresses this by enforcing that dependencies are resolved at decomposition time.

The deeper reason is observability: in a mesh, direct agent-to-agent calls create information paths invisible to the Auditor. In the hierarchical bus architecture, every coordination message is a bus event.

11.2 Why |∇L| and Not ∇L Sign as the Primary Split

In v0.7 of the system, the gradient sign was the primary decision split: improving ($\nabla L < 0$) versus worsening ($\nabla L > 0$). This was wrong for a subtle reason: a *logically wrong approach* ($P > \rho$) can produce *apparently improving loss* ($\nabla L < 0$) if the gap metric is gameable or if the system is converging into the wrong basin.

The v0.8 decision table demotes ∇L sign to an urgency modulator and promotes $|\nabla L|$ (magnitude) as the meaningful split: does the system have any directional information at all? Signal → can adapt (refine or change_approach); no signal → must escape the plateau (change_path or break_symmetry). The P threshold then determines which escape mechanism applies.

11.3 Why GGS is the Sole Writer to R5

In v0.7, the Meta-Validator (R4b) wrote memory entries on task acceptance or failure. This created two problems: (a) memory writes were not paired with loss computations, making it impossible to assign a magnitude or valence to the experience; (b) R4b writes bypassed GGS observability — the per-task decision log had no record of what was written or why.

Consolidating all writes through GGS ensures every Megram carries a formally derived (f, σ, k) triple tied to a specific loss computation. Memory quality is a function of GGS quality; the accountability is clear.

11.4 Why the Auditor Cannot Intervene

An auditor that can issue corrections becomes a second controller. Two controllers operating on the same plant with different information sets produce oscillation. The Auditor's structural independence — it cannot instruct any agent and cannot be instructed by any agent — is a necessary condition for its role: it provides the external view that the self-referential operational loops cannot provide about themselves.

11.5 Why D ≤ δ Rather Than D = 0 for Success

Requiring all criteria to pass before accepting a result burns budget on noise-level gaps. If $D \leq \delta = 0.3$, the result is within the convergence threshold: the user's intent is substantially met, and replanning further would produce diminishing returns. This is analogous to early stopping in iterative optimization: the cost of the next round exceeds the expected improvement.

---

12. Limitations and Future Work

Empirical calibration: The GGS hyperparameters ($\alpha, \beta, \lambda, \delta, \rho, \varepsilon, \theta$) are currently set by design reasoning, not empirical optimization. The Auditor accumulates session statistics; a future v0.9 pass will derive task-type-specific defaults from this data.

Dreamer upward consolidation: Promotion of Megram clusters to timeless Common Sense (v0.9 roadmap) requires LLM calls during the background consolidation pass. The promotion thresholds ($M_{\text{att}} \geq 5.0$, $|M_{\text{dec}}| \geq 3.0$) must be calibrated against false promotion rates in practice.

Single-machine scope: The current implementation targets single-machine, local-tool task execution. Multi-agent coordination across machines would require R5 to support concurrent multi-writer access and cross-agent SOP promotion — a research-track item.

T-layer evolution: The Thinking layer (Agent Laws, system persona) is currently hardcoded. A controlled mechanism for T-layer evolution from high-confidence C-level consolidations is a Phase 2 research item — the challenge is preventing value drift while allowing calibration.

Evaluation: Systematic benchmarking across task categories (file management, web research, system administration, creative composition) is in progress. The per-task decision log provides the substrate for reproducible offline replay.

---

13. Conclusion

The two central technical contributions of this paper are the Goal Gradient Solver and the MKCT cognitive memory pyramid.

GGS replaces the reactive, snapshot-driven replanner with a dynamic-differential controller. By maintaining the trajectory of the loss L_t across correction cycles and decomposing failure type via the process implausibility metric P, GGS can distinguish four structurally distinct situations that a snapshot replanner conflates: converging correctly (refine), converging into the wrong basin (change_approach), stuck in the right approach (change_path), and stuck with no signal whatsoever (break_symmetry). The 24-cell decision table is formally complete and non-overlapping — every replan outcome maps to exactly one macro-state with a deterministic directive.

MKCT replaces the single-score relevance store with a dual-channel convolution substrate. By computing attention (unsigned energy) and decision (signed preference) as independent integrals over time-decayed Megram tuples, the memory system preserves information that a single-score system irreversibly destroys: the distinction between Caution (high variance, high attention, near-zero decision) and Ignore (low attention). The GGS Quantization Matrix assigns (f, σ, k) values deterministically from macro-states, pairing every memory write with a formally grounded loss computation.

Both mechanisms are enabled by a strict hierarchical architecture — one Metaagent with complete information, N ephemeral Effector Agents with narrow scope — that makes gradient computation tractable, memory writes auditable, and the entire system observable through a single message bus.

The code is open source at: `github.com/haricheung/agentic-shell`.

Acknowledgements

References

[1] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv:2308.08155*.

[2] Hong, S., Zheng, X., Chen, J., Cheng, Y., Zhang, C., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., & Wu, C. (2023). MetaGPT: Meta Programming for Multi-Agent Collaborative Framework. *arXiv:2308.00352*.

[3] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv:2210.03629*.

[4] Yuksekgonul, M., Bianchi, F., Boen, J., Liu, S., Huang, Z., Guestrin, C., & Zou, J. (2024). TextGrad: Automatic "Differentiation" via Text. *arXiv:2406.07496*.

[5] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *NeurIPS 2023*.

[6] Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., & Wang, Y.-X. (2023). Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models. *arXiv:2310.04406*.

[7] Packer, C., Fang, V., Patil, S. G., Zhang, K., Wooders, S., & Gonzalez, J. E. (2023). MemGPT: Towards LLMs as Operating Systems. *arXiv:2310.08560*.

[8] Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative Agents: Interactive Simulacra of Human Behavior. *UIST 2023*.

[9] Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., Zhang, S., Deng, X., Zeng, A., Du, Z., Zhang, C., Shen, S., Zhang, T., Su, Y., Sun, H., Huang, M., Dong, Y., & Tang, J. (2023). AgentBench: Evaluating LLMs as Agents. *arXiv:2308.03688*.

[10] Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., & Ghanem, B. (2023). CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. *NeurIPS 2023*.

[11] Asimov, I. (1950). *I, Robot*. Gnome Press. [Source of the Three Laws of Robotics — Laws 1, 2, 3.]

[12] Asimov, I. (1985). *Robots and Empire*. Doubleday. [Source of the Zeroth Law of Robotics — Law 0 in artoo's framework.]

[13] Wei, J. "Asymmetry of Validation and Verifier's Rule." Personal essay. [The verification asymmetry principle motivating artoo's two-tier validation design.]

[14] Huawei France Research Center. "Episodic Memories Generation and Evaluation Benchmark for Large Language Models." [Inspiration for the Megram 11-tuple episodic memory structure.]

Appendix A: GGS Full 24-Cell Enumeration

#	∇L	D	P	Ω	Macro-State
1	$< -\varepsilon$	$\le \delta$	$\le \rho$	$< \theta$	success
2	$< -\varepsilon$	$\le \delta$	$> \rho$	$< \theta$	success
3	$< -\varepsilon$	$\le \delta$	$\le \rho$	$\ge \theta$	abandon
4	$< -\varepsilon$	$\le \delta$	$> \rho$	$\ge \theta$	abandon
5	$< -\varepsilon$	$> \delta$	$\le \rho$	$< \theta$	refine
6	$< -\varepsilon$	$> \delta$	$> \rho$	$< \theta$	change_approach
7	$< -\varepsilon$	$> \delta$	$\le \rho$	$\ge \theta$	abandon
8	$< -\varepsilon$	$> \delta$	$> \rho$	$\ge \theta$	abandon
9	$\lvert\cdot\rvert < \varepsilon$	$\le \delta$	$\le \rho$	$< \theta$	success
10	$\lvert\cdot\rvert < \varepsilon$	$\le \delta$	$> \rho$	$< \theta$	success
11	$\lvert\cdot\rvert < \varepsilon$	$\le \delta$	$\le \rho$	$\ge \theta$	abandon
12	$\lvert\cdot\rvert < \varepsilon$	$\le \delta$	$> \rho$	$\ge \theta$	abandon
13	$\lvert\cdot\rvert < \varepsilon$	$> \delta$	$\le \rho$	$< \theta$	change_path
14	$\lvert\cdot\rvert < \varepsilon$	$> \delta$	$> \rho$	$< \theta$	break_symmetry
15	$\lvert\cdot\rvert < \varepsilon$	$> \delta$	$\le \rho$	$\ge \theta$	abandon
16	$\lvert\cdot\rvert < \varepsilon$	$> \delta$	$> \rho$	$\ge \theta$	abandon
17	$> \varepsilon$	$\le \delta$	$\le \rho$	$< \theta$	success
18	$> \varepsilon$	$\le \delta$	$> \rho$	$< \theta$	success
19	$> \varepsilon$	$\le \delta$	$\le \rho$	$\ge \theta$	abandon
20	$> \varepsilon$	$\le \delta$	$> \rho$	$\ge \theta$	abandon
21	$> \varepsilon$	$> \delta$	$\le \rho$	$< \theta$	refine
22	$> \varepsilon$	$> \delta$	$> \rho$	$< \theta$	change_approach
23	$> \varepsilon$	$> \delta$	$\le \rho$	$\ge \theta$	abandon
24	$> \varepsilon$	$> \delta$	$> \rho$	$\ge \theta$	abandon

Appendix B: Loss Hyperparameters (v0.8 Defaults)

Parameter	Symbol	Default	Meaning
Distance weight	α	0.6	Weight on intent-result distance D
Process weight	β	0.3	Weight on process implausibility P
Resource weight	λ	0.4	Weight on resource cost Ω
Ω replan sub-weight	w_1	0.6	Fraction of Ω from replan count
Ω time sub-weight	w_2	0.4	Fraction of Ω from elapsed time
Plateau threshold	ε	0.1	$\|\nabla L\|$ below this → no directional signal
Convergence threshold	δ	0.3	D below this → accept as success
P threshold	ρ	0.5	P above this → logical failure
Abandon threshold	θ	0.8	Ω above this → abandon
Time budget	time_budget_ms	300,000	5 minutes per task
Max replans	maxReplans	3	Used in Ω replan sub-computation
Law 2 kill threshold	—	2	Consecutive worsening rounds before forced abandon

Appendix C: Role Accountability Map

Failure Mode	Accountable Role
User intent not preserved into TaskSpec	R1 — Perceiver
Plan does not address all criteria	R2 — Planner
Tool executed incorrectly	R3 — Executor
Incorrect subtask verdict	R4a — Agent-Validator
Incorrect task-level verdict	R4b — Meta-Validator
Memory not queried before planning	R2 — Planner
Memory write without loss computation	R7 — GGS (should not happen; GGS is sole writer)
Boundary violation undetected	R6 — Auditor
Illegal action not blocked	R3 — Executor (Law 1 enforcement)