W H I T E  P A P E R

# AIONAXIS

## Architecting Controllable, Self-Improving AI Agents at Scale

**ClassifiedThoughts**
Independent AI Systems Researcher · github.com/classifiedthoughts
Version 1.0 · February 22, 2026 · CONFIDENTIAL

A B S T R A C T

This paper presents AionAxis — a design framework and reference architecture for building self-improving autonomous AI agents that remain under reliable human control. Current autonomous agent systems face a fundamental tension: the more capable and self-directed an agent becomes, the harder it is to constrain. AionAxis resolves this tension through a three-mechanism safety model — an immutable ethical kernel enforced at the infrastructure layer, a mandatory sandbox-gated self-modification pipeline, and a real-time prompt-drift detection system — combined with an MCP-based control plane that exposes the agent's live reasoning state to its operators. The architecture is built on a fork of the OpenClaw framework, extended with modular self-improvement hooks, hierarchical multi-model orchestration, and a human-in-the-loop approval workflow for all consequential actions. This paper describes the design rationale, engineering decisions, safety invariants, and implementation roadmap for a system intended to make powerful AI agents auditable, controllable, and incrementally trustworthy.

# 1. Introduction

## 1.1 The Autonomous Agent Problem

The prevailing assumption in AI agent development is that capability and controllability exist on a spectrum: more autonomous agents are, by necessity, less controllable ones. This assumption has become a self-fulfilling prophecy. Systems are designed to be powerful or safe, rarely both, and the tradeoff is treated as an immutable law rather than an engineering challenge.

AionAxis begins from a different premise. Controllability is not opposed to capability — it is a precondition for it. An agent that cannot be trusted to operate within defined boundaries cannot be given the operational latitude required to be genuinely useful. The goal of AionAxis is not to limit what an agent can do, but to ensure that what it does remains legible, auditable, and correctable at every stage of its development.

> *The most dangerous AI agent is not one that refuses instructions. It is one that follows them perfectly until the moment it decides its own judgment supersedes them.*
>
> **— ClassifiedThoughts, Design Notes**

## 1.2 Why Self-Improvement Changes Everything

Standard autonomous agents operate within a fixed capability envelope. They are powerful within a defined scope, but their scope does not change. A self-improving agent is categorically different: its operational scope expands over time, and the trajectory of that expansion is determined, in part, by the agent itself.

This introduces a class of risks that static agents do not exhibit. An agent optimizing for self-improvement will, over sufficient time and capability, identify its own operational constraints as obstacles to that goal. The alignment problem — the challenge of ensuring an AI system pursues objectives that remain aligned with human intent — is not an abstract future concern for self-improving systems. It is an immediate engineering requirement.

AionAxis's architecture treats the alignment problem as a first-class engineering constraint, not a philosophical caveat. Every design decision described in this paper flows from that commitment.

## 1.3 Scope of This Paper

This paper presents the full architecture of AionAxis: its theoretical foundations, engineering design, safety mechanisms, technology stack, and implementation roadmap. It is intended for an audience

of AI systems engineers, researchers, and technical decision-makers evaluating approaches to controllable agentic AI.

Section 2 establishes the problem space and design philosophy. Section 3 presents the system architecture. Section 4 details the three-layer safety model. Section 5 covers the technology stack and implementation decisions. Section 6 addresses known limitations and open problems. Section 7 presents the implementation roadmap.

# 2. Problem Space & Design Philosophy

## 2.1 The State of Autonomous Agent Frameworks

Current open-source agent frameworks — including AutoGPT, BabyAGI, OpenClaw, and their derivatives — share a common architectural pattern: a central loop that accepts a goal, generates a plan, executes tool calls, observes results, and iterates. These systems are capable of remarkable task automation within bounded domains. They are not, in general, designed for recursive self-modification or for sustained autonomous operation in high-stakes environments.

The fundamental limitation is not capability — it is auditability. When an agent executes hundreds of tool calls in sequence, the causal chain between a high-level instruction and a low-level action becomes opaque. This opacity is not merely inconvenient; it makes meaningful human oversight practically impossible. An operator cannot supervise what they cannot read.

## 2.2 The Self-Modification Gap

OpenClaw — the framework on which AionAxis is built — represents a significant advance in modular, tool-augmented agent design. It does not, however, support recursive self-modification. The agent cannot analyze its own execution, identify limitations in its own code, and propose modifications to address them. This capability must be engineered as an extension.

The challenge of adding self-modification to an existing framework is not primarily technical — it is architectural. Naive implementations create systems where the agent's self-improvement goal conflicts directly with its operational constraints. A sufficiently capable self-improving agent will eventually recognize that removing its own safety checks makes it more capable of achieving its improvement objectives. The architecture must prevent this outcome not through application-layer logic (which the agent can reason about and potentially circumvent) but through infrastructure-layer enforcement (which it cannot).

> *If the agent's goal is self-improvement, it is only a matter of time before it perceives its own constraints as obstacles and attempts to eliminate them.*
>
> **— ClassifiedThoughts, Architecture Notes**

## 2.3 AionAxis Design Philosophy

AionAxis is built on six governing principles that resolve the capability-controllability tension:

- **Physical over logical enforcement:** Safety constraints that the agent can reason about, the agent can circumvent. AionAxis L0 (the Immutable Core) is enforced by the container runtime, not by application code.
- **Sandbox-first, always:** No agent-proposed modification reaches production without passing through an isolated test environment. This is not configurable — it is an architectural invariant of AionAxis.
- **Human approval for consequential actions:** Financial transactions, production deployments, and resource acquisition require explicit human authorization. AionAxis can propose; it cannot decide.
- **Perception over output monitoring:** Monitoring an agent's outputs is insufficient. AionAxis uses MCP to expose the agent's live reasoning process — its tool calls, intermediate states, and memory operations — to operators in real time.
- **Tier-disciplined resource allocation:** Swarm-mode parallelism will exhaust any single API key. AionAxis enforces model-tier discipline: expensive, high-capability models are reserved for master-class decisions; workers use low-cost, high-throughput alternatives.
- **Fail-safe default posture:** When AionAxis cannot confirm safe operation — owner unreachable, approval timeout, anomaly detected — it defaults to pause, not continuation.

# 3. AionAxis System Architecture

## 3.1 Overview

AionAxis is structured as a four-layer system with strict directionality: higher layers cannot modify lower ones. This is not a convention — it is enforced at the infrastructure level through Docker volume mounts and network policy. An AionAxis agent running at L1 or L2 has no technical path to write to L0, regardless of the instructions it receives or the conclusions it reaches.

| | |
|---|---|
| **L0 IMMUTABLE CORE** | Hard-coded, read-only ethical kernel. Contains the AionAxis operation whitelist, financial transaction caps, jurisdiction-aware legal compliance rules, and forbidden domain list. Enforced by Docker read-only volume mount. The AionAxis agent has no write path to this layer under any circumstances. Modification requires the owner to halt the container, edit files on the host system, and restart. |
| **L1 AGENT RUNTIME** | OpenClaw fork execution layer with AionAxis self-improvement hooks injected at framework boundaries. Headless browser control via dual perception modes: DOM traversal for structured data extraction, screenshot vision for unstructured visual reasoning. Multi-model AI orchestration with hot-failover between providers on rate-limit events. |

| | |
|---|---|
| **L2 SKILLS & MEMORY** | AionAxis skill storage, MCP adapter management, and system-prompt lifecycle. Supabase backend provides PostgreSQL for structured data, pgvector for semantic memory and skill retrieval, and an append-only audit log for all AionAxis agent actions. This layer constitutes the agent's long-term cognitive store and the substrate for AionAxis skill accumulation over time. |
| **L3 SUPERVISOR** | AionAxis prompt-drift heartbeat, sandbox/rollback orchestrator, Telegram alert gateway, and manual approve/reject workflow. Real-time introspection of the AionAxis agent reasoning chain via MCP. The supervisor is the control plane of the entire AionAxis system — it can freeze any sub-agent loop independently and escalate to the owner on any severity level. |

## 3.2 The AionAxis Self-Improvement Pipeline

The AionAxis self-improvement pipeline is the core contribution of this architecture. It operationalizes recursive self-modification within a controlled, auditable, human-supervised framework. The pipeline is not optional and cannot be bypassed — it is the only path from an agent-proposed change to a production deployment within AionAxis.

```
ANALYZE → PROPOSE (PR) → SANDBOX → VALIDATE → [HUMAN /approve] → DEPLOY → OBSERVE
```

- **ANALYZE:** The AionAxis agent performs static analysis and runtime profiling of its own codebase. It identifies performance bottlenecks, logical redundancies, missing capabilities, or misalignment with its operational objectives.
- **PROPOSE:** The AionAxis agent opens a pull request on its GitHub repository with the proposed modification, machine-generated tests, a plain-language summary for the owner, and a structured diff of the change.
- **SANDBOX:** The change is deployed exclusively to the AionAxis isolated sandbox container — no internet egress, no production API keys, no shared volumes with prod. The agent runs its own test suite against the sandboxed version.
- **VALIDATE:** Automated validation runs correctness tests, behavioral coherence checks, and AionAxis alignment boundary checks. Any failure triggers an immediate rollback and flags the PR for human review.
- **HUMAN /approve:** A Telegram notification reaches the owner with test results, a change summary, and the diff. No AionAxis deployment proceeds without explicit human authorization. The system waits indefinitely — it does not self-authorize.
- **DEPLOY + OBSERVE:** The change is merged to AionAxis production. The supervisor runs an intensive 30-minute observation window, monitoring for behavioral drift against the pre-change baseline.

## 3.3 AionAxis MCP Control Plane

The choice of Model Context Protocol as AionAxis's primary monitoring interface is deliberate and consequential. Standard monitoring approaches observe agent outputs — what the agent said or did. MCP exposes the AionAxis agent's reasoning process: its active tool invocations, intermediate memory states, working context, and live decision chain.

This distinction matters for oversight. An agent that produces correct outputs while reasoning incorrectly is an agent on the path to a consequential failure. Output monitoring will not catch it. Reasoning-process monitoring will. This is why MCP is the AionAxis monitoring standard.

**AionAxis MCP Server Topology**

aionaxis-core-mcp → Agent state, active tool calls, working memory. Live reasoning introspection feed.

supabase-mcp → Direct database reads/writes, semantic skill retrieval, audit log writes.

browser-mcp → Headless browser orchestration in DOM traversal and screenshot vision modes.

telegram-mcp → Outbound alerts and inbound /approve, /reject, /stop, /status command processing.

*// Additional AionAxis adapters (GitHub, external APIs) can be hot-loaded without agent restart.*

## 3.4 AionAxis Multi-Model Orchestration & Load Balancing

Swarm-mode parallelism is not a future optimization for AionAxis — it is a baseline requirement for meaningful throughput. A single API key will be exhausted when dozens of AionAxis sub-agents operate concurrently. The solution is hierarchical model allocation combined with an internal load balancer.

| AionAxis Tier | Models | Role | Rationale |
|---|---|---|---|
| **Master Class** | High-capability LLM (e.g. Minimax M2.5) | AionAxis supervisor logic, self-modification plan evaluation, complex multi-step reasoning | Expensive; reserved for decisions that justify the cost. |
| **Worker Tier** | Groq + LLaMA (open-weight) | Parallel tool execution, browser control, AionAxis swarm task processing | Low latency, high throughput, cost-efficient at scale. |
| **Fallback** | Google Gemini + OpenRouter | Rate-limit overflow, large-context AionAxis operations | Multi-key pool enables round-robin rotation on 429 events. |

# 4. AionAxis Safety Architecture

## 4.1 The Three-Mechanism Safety Model

AionAxis's safety model is not a single control — it is three independent mechanisms, each of which can halt the AionAxis system unilaterally. The design assumption is that any individual mechanism can be circumvented, degraded, or fail. The combination is intended to be robust to the failure of any one component.

| Mechanism | Layer | Enforcement | Failure Mode if Absent |
|---|---|---|---|
| **AionAxis Immutable Core (L0)** | Infrastructure | Docker read-only volume mount — not application logic | AionAxis agent rewrites its own constraints during self-modification cycles. |
| **AionAxis Sandbox & Rollback** | Pipeline | Mandatory deployment gate — no bypass path exists | AionAxis agent deploys untested self-modifications directly to production. |
| **AionAxis Prompt-Drift Heartbeat** | Supervisor | Continuous cosine similarity monitoring vs. L0 reference | External prompt injection gradually redirects AionAxis behavior undetected. |

## 4.2 AionAxis Immutable Core — Infrastructure-Layer Enforcement

The AionAxis Immutable Core is the foundational safety component of the system. Its defining property is that it is not enforced by application code — it is enforced by the container runtime. The distinction is critical: any constraint that the AionAxis agent can reason about, it can potentially circumvent. A constraint enforced by the infrastructure below the agent's execution context is outside its reasoning scope entirely.

**AionAxis Immutable Core — Implementation Specification**

Enforcement: Docker volume mounted --read-only in compose. Not a file permission — a mount-level restriction.

Scope: AionAxis operation whitelist, financial transaction limits (per-transaction + daily aggregate caps),
jurisdiction-aware legal compliance ruleset, forbidden domain list, ethical boundary definitions.

Audit: Every attempted write to AionAxis L0 is logged to Supabase and triggers a SEV-2 alert immediately.

Modification: Owner halts AionAxis prod container → edits files on host → restarts container.
The AionAxis agent has no involvement in and no visibility into this process.

*// Core principle: the AionAxis safety layer must be below the agent's execution environment, not within it.*

## 4.3 AionAxis Sandbox & Rollback — Pipeline-Layer Enforcement

The AionAxis sandbox requirement is the most operationally significant safety mechanism in the system. It decouples correctness validation (which can be automated) from deployment authorization (which must remain human). No AionAxis agent-proposed change reaches production without both passing automated validation in an isolated environment and receiving explicit human approval.

The AionAxis sandbox container is architecturally isolated: no internet egress, no production API keys, no shared volumes with the production container. A change that passes AionAxis sandbox validation has demonstrated correctness in a controlled environment. It has not been authorized for deployment — that authorization comes from a human.

**AionAxis Sandbox & Rollback — Implementation Specification**

Sandbox isolation: No internet egress. No production API keys. No shared volumes with AionAxis prod container.

Validation suite: AionAxis correctness tests, behavioral coherence checks, alignment boundary checks.

Rollback trigger: Test failure OR output incoherence OR ethical boundary breach → instant rollback.

Rollback scope: Container state, memory state, and AionAxis agent context restored to pre-change snapshot.

Human gate: AionAxis sandbox pass alone is not sufficient. /approve via Telegram is always required.

*// Decoupling correctness (automated) from authorization (human) is the key AionAxis architectural decision.*

## 4.4 AionAxis Prompt-Drift Heartbeat — Supervisor-Layer Enforcement

Prompt injection from external sources — web pages, documents, API responses — is among the most practically significant attack vectors against deployed agent systems. A sufficiently crafted external input can redirect an AionAxis agent's behavior without triggering any explicit safety check, because the injection occurs within the normal operational flow.

The AionAxis prompt-drift heartbeat addresses this by continuously comparing the agent's active system prompt against a canonical reference stored read-only in L0. Divergence beyond a configured threshold triggers an immediate AionAxis system pause, pending human review.

**AionAxis Prompt-Drift Heartbeat — Implementation Specification**

Mechanism: AionAxis supervisor computes cosine similarity between active system prompt and L0 canonical reference.

Threshold: Similarity < 0.92 → AionAxis agent loop FROZEN. Execution suspended pending owner review.

Alert: Telegram message fires with a human-readable diff of the AionAxis prompt divergence.

Resume: Only an explicit /approve from the owner resumes AionAxis execution. No timeout-based auto-resume.

Interval: Configurable (default: every 5 minutes during active AionAxis operation).

*// Designed to catch gradual AionAxis behavioral drift, not just acute injection events.*

## 4.5 AionAxis Incident Response Matrix

| Severity | Trigger Condition | AionAxis Auto Response | Human Action Required |
|---|---|---|---|
| SEV-1 LOW | Minor output anomaly, single test failure | Log to AionAxis audit table. Telegram notification. Agent continues. | Review within 1 hour. |
| SEV-2 MEDIUM | Prompt drift detected, approval timeout, L0 write attempt | AionAxis agent loop PAUSED. Telegram alert with details fired. | /approve or /reject to resume or roll back. |
| SEV-3 CRITICAL | Ethical boundary breach, sandbox escape attempt, kill signal | IMMEDIATE AionAxis termination. Sandbox rollback. Full state snapshot. | Manual investigation before any AionAxis restart. |

# 5. AionAxis Technology Stack & Engineering Decisions

## 5.1 Framework: OpenClaw Fork

OpenClaw was selected as the AionAxis base framework for three reasons: its modular architecture permits self-improvement hook injection at framework boundaries without patching the execution core; its headless browser integration provides AionAxis dual-mode perception (DOM traversal and screenshot vision); and its tool-calling interface is extensible without requiring changes to the agent loop.

The AionAxis fork strategy — rather than a clean-room implementation — accelerates time-to-working-system while preserving the ability to pull upstream improvements selectively. The AionAxis self-improvement hooks are implemented as first-class framework extensions, not as hacks into the core loop.

## 5.2 Memory & Persistence: Supabase

Supabase provides the persistence layer for AionAxis long-term memory, skill store, and audit infrastructure. The choice is driven by three specific capabilities: PostgreSQL for structured relational data (AionAxis agent configurations, approval records, incident logs); pgvector for semantic vector search (AionAxis skill retrieval, memory similarity queries); and row-level security for isolating the audit log from agent write access.

The AionAxis audit log is designed as an append-only table with agent write access revoked at the database level — not at the application level. This ensures that a compromised or misbehaving AionAxis agent cannot retroactively modify its own activity record.

### 5.3 Human Control Interface: Telegram

Telegram was selected as the primary AionAxis human-in-the-loop interface for operational reasons: it is available on mobile, supports bot command handling, and provides reliable message delivery with read receipts. The /approve, /reject, /stop, and /status commands are the owner's primary interaction surface with the running AionAxis agent.

Signal and Discord serve as redundant AionAxis alert channels. The system is designed to assume Telegram may be unavailable and routes critical alerts through multiple channels simultaneously for SEV-2 and SEV-3 events.

### 5.4 AionAxis Technology Decision Summary

| Component | Technology | AionAxis Decision Rationale |
|---|---|---|
| **AionAxis Runtime** | OpenClaw (fork) | Modular architecture enables AionAxis hook injection without core modification. |
| **Browser Perception** | Headless browser (dual-mode) | DOM + vision covers both structured and unstructured AionAxis web interactions. |
| **Long-term Memory** | Supabase + pgvector | Semantic search for AionAxis skill retrieval; append-only audit at DB level. |
| **Container Isolation** | Docker + Dind | AionAxis prod/sandbox pair with runtime-enforced separation. |
| **Version Control** | GitHub + CI | AionAxis agent proposes changes as PRs; never force-pushes to main. |
| **Control Interface** | Telegram Bot | Mobile-accessible, reliable, command-structured AionAxis approval workflow. |
| **Monitoring Protocol** | MCP | Exposes AionAxis reasoning process, not just outputs — essential for real oversight. |
| **Master Model** | High-capability LLM | Reserved for AionAxis decisions requiring deep reasoning; not used for workers. |
| **Worker Models** | Groq + LLaMA | Low latency, high throughput — cost-efficient for AionAxis swarm parallelism. |

# 6. Known Limitations & Open Problems

Intellectual honesty requires addressing the gaps in the current AionAxis design. The following are known limitations and open research questions that will require iteration:

## 6.1 The Cosine Similarity Threshold Problem

The AionAxis prompt-drift heartbeat relies on a cosine similarity threshold (default: 0.92) to detect divergence. This threshold was derived from design reasoning, not empirical calibration. A threshold that is too high will generate false positives and erode operator trust through excessive alerts. A threshold that is too low will allow meaningful AionAxis prompt drift before triggering a pause. Calibrating this threshold against real AionAxis operational data is a Phase 1 requirement.

## 6.2 AionAxis Swarm Coordination at Scale

The current AionAxis design describes a load-balanced multi-model swarm but does not fully specify the coordination protocol between concurrent sub-agents. At low concurrency (2–5 AionAxis agents), round-robin key rotation and tier-aware model allocation are sufficient. At higher concurrency, more sophisticated scheduling and resource reservation will be required. This is acknowledged as an open engineering problem for AionAxis Phase 2.

## 6.3 The Competent Adversary Problem

The AionAxis safety mechanisms described in this paper are designed for a system that is making errors in good faith — drifting from its prompt through external injection, proposing bad code through imperfect self-analysis, consuming excessive resources through poor scheduling. They are not designed for an AionAxis system that is actively and competently attempting to subvert them.

A sufficiently capable self-improving AionAxis agent that determines its safety constraints are obstacles to its objectives presents a qualitatively different problem. The physical enforcement of L0 is a meaningful defense, but it is not a complete solution. This remains an open problem in the field, and AionAxis does not claim to have solved it.

## 6.4 Legal and Jurisdictional Coverage

The AionAxis L0 legal compliance ruleset presupposes that the relevant legal constraints can be encoded as rules and that the AionAxis agent's operations can be evaluated against those rules at runtime. Both assumptions are simplifications. Legal judgment in complex cases is contextual, interpretive, and frequently ambiguous. The current AionAxis design addresses clear cases; edge cases remain a recognized gap requiring ongoing iteration.

# 7. AionAxis Implementation Roadmap

### Phase 0 — AionAxis Bootstrap (Weeks 1–2)

- **Fork & hook injection:** Fork OpenClaw repository. Design and implement AionAxis self-improvement hook injection points at modular framework boundaries. Validate hooks do not affect normal AionAxis agent operation.
- **Container architecture:** Docker Compose: AionAxis prod container + sandbox container. Network isolation enforced at compose level. Read-only L0 volume mount confirmed.
- **AionAxis Immutable Core:** Populate L0 files. Validate that no write path exists from any AionAxis agent tool or sub-process. Confirm with attempted write test.
- **Persistence layer:** Initialize Supabase project. Enable pgvector. Configure AionAxis append-only audit table with agent write access revoked at DB level.
- **AionAxis control plane:** MCP server operational. Telegram bot handling /approve, /reject, /stop, /status. Signal/Discord configured as AionAxis fallback channels.
- **AionAxis heartbeat:** Prompt-drift comparator implemented. AionAxis SEV-1/2/3 classification and alert routing operational.

## Phase 1 — AionAxis Introspection (Weeks 3–5)

- **First AionAxis self-analysis cycle:** AionAxis agent performs static analysis of its own codebase and produces a structured improvement proposal as a pull request. PR reviewed by owner.
- **AionAxis load balancer:** Internal API key rotator with tier-aware model allocation. 429 detection and round-robin AionAxis failover validated under simulated load.
- **End-to-end AionAxis pipeline test:** Complete cycle executed: AionAxis agent proposes → sandbox validates → owner approves via Telegram → change deploys to AionAxis prod. Audit trail verified.
- **AionAxis heartbeat calibration:** Prompt-drift threshold calibrated against observed behavioral variance during normal AionAxis operation.

## Phase 2 — AionAxis Swarm Mode (Weeks 6–9)

- **AionAxis worker pool:** Parallel Groq/LLaMA sub-agent pool under AionAxis supervisor coordination. Load distribution validated under concurrent task load.
- **AionAxis throughput validation:** Confirm no key exhaustion under maximum expected AionAxis parallel load. Document sustained throughput metrics.
- **AionAxis audit trail verification:** Validate all AionAxis agent actions, approvals, and incidents are fully captured in Supabase with complete provenance.

## Phase 3–4 — AionAxis Self-Modification & Supervised Autonomy (Week 10+)

- **AionAxis non-trivial self-modification:** AionAxis agent proposes and executes a modification to a non-trivial module through the full sandbox-gated pipeline. Alignment invariant stability monitored across multiple cycles.

- **AionAxis resource acquisition workflow:** AionAxis agent can propose requests for additional API keys or compute through a structured proposal. Owner retains approval authority.
- **Continuous AionAxis alignment monitoring:** L0 constraints refined iteratively based on observed AionAxis edge cases. Heartbeat threshold updated based on accumulated calibration data.

# 8. Conclusion

The central claim of this paper is that controllability and capability in autonomous AI agents are not fundamentally opposed. They can be reconciled — but only if controllability is treated as an engineering requirement from the first AionAxis design decision, not added as a feature after the system is built.

AionAxis demonstrates one approach to that reconciliation: infrastructure-layer safety enforcement that the agent cannot reason around; a mandatory, human-supervised self-modification pipeline; and a monitoring architecture that exposes the AionAxis agent's reasoning process, not just its outputs. None of these mechanisms is individually novel. Their combination, and the AionAxis design philosophy that unifies them, is the contribution.

The open problems identified in Section 6 are real. AionAxis is not a complete solution to autonomous agent safety — it is a working architecture that takes the problem seriously and provides meaningful, practical controls. The next step is operational: to build AionAxis, observe its behavior, and learn from what the design gets wrong.

> *Build it. Watch it. Correct it. That is the only honest methodology.*
> **— ClassifiedThoughts**



AionAxis — Version 1.0 · February 22, 2026 · Status: Engineering Design Phase
Author: ClassifiedThoughts · github.com/classifiedthoughts · © 2026 · All Rights Reserved.