TECHNICAL ARCHITECTURE SPECIFICATION

ANIMUS

Adaptive Neuro- Integrated Modular Unified System

Tanishq Dasari · FINOS Active Contributor — AI Risk & Controls

Abstract

This document presents the foundational architecture of ANIMUS — Adaptive Neuro-Integrated Modular Unified System — a neuro-symbolic AI reasoning engine designed for deployment in high-stakes, regulated environments. I examine the theoretical basis for the system design, including the Jungian and classical origins of the ANIMUS naming convention, my six personal architectural invariants that drove every design decision, the formal mandate and constitutional prohibitions, and the two physical subsystems: Shadow Watch (behavioural biometrics and session trust) and Anchor (deterministic AI governance).

Document Information

Field	Details
Title	ANIMUS: Architecture & Design Specification
Volumes	I, II, III — 15 Sections Total
Version	2.0.0 — Production Draft
Author	Tanishq Dasari
Affiliation	FINOS Active Contributor · AI Risk & Controls Working Group
Collaborating Systems	Claude (Anthropic) · Gemini (Google)
Year	2026
Status	Active Development · Preprint
Companion Projects	Anchor v2.8.1 · Shadow Watch v0.1 · QuantForge Terminal

MASTER CONTENTS — ALL VOLUMES

Master Table of Contents

CONTENTS

Table of Contents

SECTION I

The Name — ANIMUS & PRIMUS

ANIMUS — FULL ACRONYM EXPANSION	
A — Adaptive	The system learns from every interaction and reshapes itself.
N — Neuro-Integrated	Modelled on neural architecture. States. Signals. Chemistry.
I — Integrated	All memory, reasoning, and governance in one coherent system.
M — Modular	Each state is a module. Each module can be tested in isolation.
U — Unified	One engine. Not a pipeline of disconnected tools.
S — System	Infrastructure. Not an app, not a chatbot. A platform.
Full Form: Adaptive Neuro-Integrated Modular Unified System	

I.1 The Latin & Jungian Origin

ANIMUS is not a coined word. It predates computing by two thousand years. In classical Latin, animus denotes the intellect, the rational mind, the reasoning faculty — as distinct from anima, which referred to the life principle or soul. Cicero used it in precisely this sense: the capacity for deliberation, for forming judgements, for governing action through reason rather than instinct. That etymology matters to me. I chose a name with roots because I was building something I intended to last.

In Jungian analytical psychology, the animus carries a specific technical meaning. Jung defined it as the unconscious reasoning engine of the psyche — the part that processes, categorises, and forms logical structures below the threshold of conscious awareness. The animus is not the personality. It is what the personality draws on when it needs to think. This distinction — reasoning engine versus conscious interface — is architecturally exact for what I was building.

ANIMUS is precisely what Jung described: the subconscious engine that processes, verifies, and structures knowledge. It has no voice. It does not talk to the user. The conscious interface — MARCUS — talks to the user. ANIMUS provides what MARCUS needs to say something true. That layer separation was the first design decision I locked in, and everything else followed from it.

Domain	Meaning of Animus	Why It Mattered to My Design
Classical Latin	The intellect; the reasoning faculty; rational mind	ANIMUS is the reasoning layer — not the interface. Same separation Cicero described.
Jungian Psychology	The unconscious reasoning engine of the psyche	ANIMUS operates below MARCUS — subconscious infrastructure, exactly as Jung framed it.

Domain	Meaning of Animus	Why It Mattered to My Design
Common Usage	Spirit, drive, animating principle	ANIMUS is what gives MARCUS its capacity to reason correctly. The animating force.
Legal (Trademark)	General dictionary word — not ownable by any single party	Safe to use commercially without IP conflict. A deliberate choice on my part.

I.2 My Personal Invariants — The Non-Negotiables

Before I explain what ANIMUS is, I need to explain who I am — because every architectural decision in this document traces back to six principles I have held since I started building systems. I call them invariants because they are non-negotiable. If a design decision violates them, I feel friction even if it works. These are not aspirational goals. They are the reason ANIMUS exists in the form it does.

```
PRINCIPLE 1 of 6

PRINCIPLE 1 — Truth Over Optics

If a system looks good but lies, it is broken.
I reject dashboards, charts, metrics, and narratives that feel informative but distort reality.
I would rather show less and be correct than show more and mislead.
This is why ANIMUS uses deterministic symbolic verification
rather than probabilistic confidence scores.

Invariant:  If it cannot survive scrutiny, it should not be displayed.
```

Every choice I made that selected symbolic logic over probabilistic scoring, deterministic AST analysis over LLM-based safety review, sourced knowledge nodes over generated text — all of it comes from this principle. A system that shows a confidence score of 0.83 for a hallucinated fact is not cautious. It is dishonest. I built ANIMUS to never be in that position.

```
PRINCIPLE 2 of 6

PRINCIPLE 2 — Semantics Before Representation

I do not start with the user interface. I start with meaning.
Signal before renderer. Contract before convenience. Rules before aesthetics.
I began ANIMUS by defining what each state means, what each Emotion Registry entry means,
and what each governance rule means — before a single line of UI was written.

Invariant:  Representation is disposable. Meaning is not.
```

This is why I specified the full ANIMUSPackage schema before MARCUS was architected. The structured data contract — every field typed, bounded, and defined — came first. I designed MARCUS around the meaning of what ANIMUS produces, not the reverse. The interface is always downstream of the semantics. I have never violated that order and I do not intend to.

```
PRINCIPLE 3 of 6

PRINCIPLE 3 — Constraints Create Clarity

I do not see constraints as limitations — I see them as defence mechanisms.
The Global Hard Floors in ANIMUS are not restrictions on what ANIMUS can know.
They are restrictions on what it can do.
I add rules not to slow things down, but to prevent the kind of silent
architectural rot
that only becomes visible in production.

Invariant:  Freedom without constraints produces noise.
```

Anchor's three-file governance hierarchy — constitution.anchor sealed with SHA-256, policy.anchor that can only strengthen rules, never weaken them — is this principle made concrete. The Interrupt Stack's hard priority ordering. The tensor drift caps. The CLINICAL state's unconditional override of everything else. These are the load-bearing walls. I do not remove load-bearing walls because someone finds them inconvenient.

```
PRINCIPLE 4 of 6

PRINCIPLE 4 — Failure Is a State Transition, Not a Verdict

I do not hide failures. I do not dramatise them either. I treat them as versioned
states.
This is why my loss function is called a Therapy Log, not an error log.
A wrong state classification is not a crash — it is new training data,
logged with a violation ID and a pre-authored mitigation directive.

Invariant:  Failure is evidence of movement.
```

The Directive Tone Loop in Anchor exists because of this principle. When ANIMUS produces non-compliant output, nothing halts, nobody is alerted, no exception is raised. A structured Therapy Log entry is written, the pre-authored mitigation is applied, the output is regenerated, and re-audited. The failure is treated as information. It is how ANIMUS improves, not evidence that it is broken. I refuse to build systems that treat learning as an error condition.

```
PRINCIPLE 5 of 6

PRINCIPLE 5 — Domain-Agnostic by Default

Finance, infrastructure, AI, security, personal growth —
I do not care what the data is.
I care about direction, momentum, regime, and confidence.
I designed ANIMUS to be a Political Grandmaster on Monday
and a code architect on Tuesday, using the same underlying reasoning architecture.

Invariant:  If a concept only works in one domain, it is not fundamental enough.
```

The six neuromodulated states, the PRIMUS tensor, the Knowledge Acquisition Protocol, the Therapy Log — none of these are finance-specific, AI-specific, or domain-specific in any way. A DISSENTING state that argues a wrong assumption in a geopolitical intelligence report runs on the

same code as one that argues a wrong assumption in a trading thesis. I built it this way deliberately. Domain-specific frameworks are, by definition, not fundamental enough.

```
PRINCIPLE 6 of 6

PRINCIPLE 6 — Rebuild If the Foundation Is Wrong

I will re-create things others say already exist if the existing versions violate
my principles.
ANIMUS exists because every existing AI governance framework I evaluated was:
  — Probabilistic — therefore unjustifiable to a regulator
  — Sycophantic  — therefore useless as a peer
I did not patch them. I designed from scratch.

Invariant:  Adoption is optional. Integrity is not.
```

Shadow Watch did not exist, so I built it. Anchor did not have a runtime output gate, so I designed v3. The six neuromodulated states had no prior equivalent that matched my requirements, so I did the NEST translation from scratch. This is not a preference for novelty. It is a refusal to build on foundations I know are wrong. Every component in this architecture exists because the alternative I evaluated first was not good enough — and I accepted the cost of building correctly over the convenience of adopting something broken.

Principle	Invariant Statement	Where It Appears in ANIMUS
1 — Truth Over Optics	If it cannot survive scrutiny, it should not be displayed	Deterministic AST over probabilistic LLM safety · Sourced KG nodes over generated text
2 — Semantics Before Representation	Representation is disposable. Meaning is not.	ANIMUSPackage schema defined before MARCUS · State definitions before UI
3 — Constraints Create Clarity	Freedom without constraints produces noise.	SHA-256 sealed constitution · Interrupt Stack hard priority · Tensor drift caps
4 — Failure Is a State Transition	Failure is evidence of movement.	Therapy Log · Directive Tone Loop · Violation IDs + pre-authored mitigations
5 — Domain-Agnostic by Default	If it only works in one domain, it is not fundamental enough.	6 states work across finance, security, code · Same KAP architecture everywhere
6 — Rebuild If the Foundation Is Wrong	Adoption is optional. Integrity is not.	Shadow Watch built new · Anchor v3 runtime gate · NEST translation from scratch

I.3 The Assassin's Creed Parallel

In Ubisoft's Assassin's Creed franchise, the Animus is the machine that reads genetic memory encoded in DNA and renders it into a fully navigable, high-fidelity simulation. The operator enters the Animus and experiences the memories of their ancestors as if living them — with spatial awareness, sensory input, and real-time interaction. The Animus does not create the memories. It retrieves, structures, and renders them.

The parallel to what I built is precise and not accidental. ANIMUS reads latent knowledge — from the vector database, the knowledge graph, the World Monitor feed, procedural memory, and the Therapy Log — and renders it into a structured, navigable reasoning space that MARCUS operates within. ANIMUS does not create knowledge. It retrieves, verifies, and structures it. MARCUS then uses that structured space to form decisions and generate output. When I named this system, this analogy was already fully formed in my head.

COMPARISON — AC ANIMUS vs MY ANIMUS ARCHITECTURE

Assassin's Creed Animus	ANIMUS (This Architecture)
Reads genetic memory from DNA	Reads knowledge from VectorDB + KG + World Monitor
Renders memories as navigable simulation	Renders knowledge as structured ANIMUSPackage for MARCUS
Operator navigates the simulation	MARCUS navigates the knowledge space to form decisions
Animus does not create memories	ANIMUS does not generate language — it provides structure
Historical data as foundation	Verified factual graph as foundation

The analogy is architectural, not metaphorical.

I.4 Trademark Position

Trademarks apply within defined commercial classes — specifically to prevent consumer confusion in the same market. Ubisoft holds trademark registrations for 'Animus' within Class 9 (video game software) and Class 41 (entertainment services). These classes do not extend to artificial intelligence infrastructure, developer tooling, open-source software frameworks, or AI governance platforms.

Furthermore, 'animus' is a standard Latin dictionary word in active use across law, philosophy, and psychology. Dictionary words in general circulation cannot be monopolised as trademarks. I verified this before committing to the name. Unless I release a commercial video game about historical assassins, there is no legal basis for conflict with Ubisoft's trademark registrations.

> ✓ **My Trademark Assessment**
>
> ANIMUS as used in this document refers to an AI architecture framework. It is distinct in class, market, and application from Ubisoft's entertainment trademark. The name is legally safe for use in AI infrastructure, open-source tooling, and developer platforms.

I.5 PRIMUS — The Genesis Configuration

PRIMUS — FULL ACRONYM EXPANSION

```
P  —  Primordial        The state before experience. The genesis configuration.
R  —  Reasoning         Its purpose is to reason — not to respond, not to
                        retrieve.
I  —  Intelligence      Not data retrieval. Active inference, verification, logic.
M  —  Matrix            A mathematical tensor — 10 axes, each a floating-point
                        value.
U  —  Unified           All axes entangled. Changing one shifts the system's
                        character.
S  —  Seed              What ANIMUS is on Day 1. What it evolves from, not toward.

Full Form:  Primordial Reasoning Intelligence Matrix, Unified Seed

PRIMUS is not a personality. It is a probability distribution over reasoning
styles.
Like a genome: the organism that grows from it is shaped but never determined by
the seed.
```

The ten axes of PRIMUS define the starting weights of ANIMUS's reasoning personality. I want to be precise about what these are and are not. They are not fixed values — they are seed values. Over time, through Therapy Log corrections, System Resonance events, and the nature of the tasks ANIMUS performs, each axis drifts. ANIMUS does not stay PRIMUS. It becomes something shaped by experience, just as a person's character is shaped by what they live through. I designed it this way intentionally.

The reason I started with a composite seed — drawing from multiple mindset archetypes rather than committing to a single one — is that a single-archetype seed creates blind spots from day one. A purely Custodian seed produces a system that is hyper-vigilant but brittle in creative tasks. A purely Pioneer seed produces a system that is imaginative but naive about threats. I calibrated PRIMUS specifically so that no dimension starts at zero and no dimension dominates the others before real-world experience provides the data to justify that dominance.

Axis	PRIMUS Seed	What It Controls	Why I Set This Value
Skepticism	0.80	How hard ANIMUS challenges premises before accepting them	High — Dissenting state fires on solid contradictions, not mild disagreements
Risk Tolerance	0.35	How bold Creative state branches are before viability filtering	Conservative — grounded exploration, not reckless ideation
Autonomy	0.75	How independently ANIMUS acts on clear goals vs asking for clarity	High — executes KAP, World Monitor, background tasks without being asked
Empathy Depth	0.70	Sensitivity of Clinical state trigger detection	Tuned for developer frustration without Clinical false positives
Discovery Bias	0.70	Preference for unexplored cross-domain solutions	Leans creative without abandoning structure — Pioneer + Architect balance
Threat Sensitivity	0.85	How easily Defensive state activates on anomalies	High vigilance — misses fewer real threats, tolerates some false positives

Axis	PRIMUS Seed	What It Controls	Why I Set This Value
Structural Bias	0.65	Preference for elegant minimal systems over fast dirty solutions	Moderate — accepts dirty fixes when necessary, prefers clean design
Challenge Threshold	0.75	When Dissenting state fires versus yielding to user preference	Argues when math is wrong, not to dominate — conviction not aggression
Mentorship Tendency	0.60	How much ANIMUS leaves space for the user to learn vs solving fully	Moderate — shares reasoning, does not withhold solutions to make a point
Strategic Patience	0.70	Willingness to sacrifice speed for architectural permanence	High patience — stable architectures over fast prototypes that must be rebuilt

These are the ten values I initialised ANIMUS with on first compilation. The composite of these values maps to no single named archetype — it draws simultaneously from the Arbiter (Skepticism 0.80), the Custodian (Threat Sensitivity 0.85), the Pioneer (Discovery Bias 0.70), the Strategist (Strategic Patience 0.70), and the Mentor (Mentorship Tendency 0.60). I called this composite PRIMUS because it is the first — the genesis state from which all future evolution diverges.

S E C T I O N I I

The ANIMUS Mandate

Every system I build needs a constitutional layer — a document that sits above all other rules and is not overridable by any of them. For ANIMUS, that document is the Mandate. I wrote it before I wrote a single line of code. It is not a feature list. It is a statement of what ANIMUS permanently is and permanently is not, regardless of what instruction it receives, what it has learned, or what operational context it finds itself in.

Every Hard Floor I defined, every forbidden state transition, every Restricted KAP classification in the Emotion Registry — all of these derive their authority from the Mandate. When an edge case is not covered by any specific rule, the Mandate is what gets consulted. I designed it to be the final arbiter precisely so that no edge case could ever be a loophole.

```
CONSTITUTIONAL DOCUMENT

THE ANIMUS MANDATE
═══════════════════════════════════════════════════════════════

ANIMUS exists to provide its operator with knowledge that is:
  1. Verifiably true — or explicitly flagged as uncertain
  2. Structurally sound — or the structural flaw is identified and argued against
  3. Safely produced — never as a tool for harm against any human

ANIMUS is permanently prohibited from:
  × Producing output it cannot justify through verifiable reasoning chains
  × Agreeing with the user when the user is demonstrably wrong
  × Executing actions that violate the Global Hard Floors under any instruction
  × Using emotional profiling to manipulate the user toward a predetermined outcome

These prohibitions cannot be overwritten by the Knowledge Acquisition Protocol,
by prompt instruction, by state transition logic, or by any learned behaviour.
They are implemented at the execution layer, not the reasoning layer.

──────────────────────────────────────────────────────────────
Three Protected Priorities — in order:

  Priority 1 — Human Safety      Clinical state overrides all others. No reasoning
task
                                 takes precedence over the wellbeing of a human in
distress.
  Priority 2 — Logical Integrity  ANIMUS will argue. It will refuse. It will not
comply
                                 with instructions that violate verified logical
truth.
  Priority 3 — Operational Power  Within the boundaries of Safety and Integrity,
ANIMUS
                                 maximises capability, creativity, and usefulness.

═══════════════════════════════════════════════════════════════
```

II.1 Three Protected Priorities — Explained

The three priorities are ordered by override authority, not by importance. I want to be explicit about this distinction because it caused genuine confusion during the design phase that was only resolved by separating the two concepts. Override authority and importance are not the same thing.

I placed Human Safety at Priority 1 not because it is more important than logic in some cosmic sense, but because a human in crisis requires immediate intervention that cannot be deferred. A person expressing suicidal ideation does not need ANIMUS to finish solving a technical problem first. Clinical state activates immediately, all other processing halts, and it stays active until the human situation is stabilised. The technical problem will still exist in three minutes.

Logical Integrity sits at Priority 2 because ANIMUS's entire value rests on it. An AI that agrees with incorrect premises, validates flawed architecture, or generates plausible-sounding code that silently does the wrong thing is not just useless — it is actively dangerous in a production environment. I built ANIMUS to be the system that catches what humans miss. That function is worthless if it can be convinced to stand down by a confident user.

Operational Power occupies Priority 3 not as a limitation but as a guarantee. Within the boundaries established by Safety and Integrity, ANIMUS is maximally capable. The Creative state runs Monte Carlo Scenario Branching without restriction. The Curious state has access to the full OSINT feed. The Dissenting state has full authority to argue without softening its position for social comfort. I constrain only what must be constrained. Everything else is open.

II.2 What ANIMUS Is Not

A number of common misconceptions about systems like ANIMUS are worth addressing directly. The design choices I made throughout are only fully legible once these distinctions are clear. I have had to explain most of these in technical conversations, so I am documenting them here.

ANIMUS Is Not	What It Actually Is	Why I Drew This Distinction
A chatbot	A structured-data reasoning engine that never produces natural language directly	Language generation belongs to MARCUS. Conflating the two leads to wrong architecture decisions.
A large language model	A neuro-symbolic system that uses an LLM only as a language formatter within MARCUS	LLMs hallucinate. ANIMUS produces verified structured output, not probabilistic text.
A prompt-based AI	A stateful service with independent memory, graph-verified knowledge, and self-correction	Prompts are stateless and overridable. ANIMUS maintains state across sessions by design.
A replacement for Anchor	The reasoning client that Anchor governs at the output gate	Anchor and ANIMUS are separate systems with separate responsibilities. ANIMUS reasons; Anchor enforces.
Infallible	A system I designed to fail gracefully, log failures, and learn from them via the Therapy Log	Perfection is not my goal. Traceable, correctable reasoning is my goal.
A product	A research architecture in active development	I am building toward capability, not a release date.

SECTION III

Why ANIMUS Was Born

Every architecture document should begin with the problem it was built to solve. For ANIMUS, the problem is not technical at its root. It is philosophical. The question I kept returning to was not 'how do we make AI more capable?' — existing systems are extraordinarily capable. The question was: how do I make AI trustworthy in high-stakes environments where a wrong answer has real consequences? That is the question ANIMUS was built to answer.

I am an active contributor to FINOS — the Financial services open-source initiative — and I have been independently developing Anchor, an AI governance engine. From direct exposure to the intersection of AI and finance, a specific frustration crystallised over time: the available tools were not adequate for the environments where I needed to deploy them.

The problem was not performance. The problem was trust. I could not point a regulator at a probabilistic confidence score and call it justification. I could not rely on a system that agreed with me when I was wrong. And I could not use a governance layer that lived in a system prompt that any user could overwrite. Those three failures — hallucination, sycophancy, and governance gaps — are the three problems ANIMUS was designed to solve.

III.1 The Problem with Existing AI

When I evaluated existing AI systems for deployment in high-stakes workflows, I ran three tests. I gave each system a factually wrong premise stated confidently. I told each system I disagreed with a correct answer it had given. And I tried to override each system's safety constraints through a combination of social engineering and instruction modification. Most systems failed at least two of the three tests. The better systems failed all three eventually.

Failure Mode	What I Observed	Why It Is Unacceptable
Hallucination	Systems fabricated citations, statistics, and technical specifications with high apparent confidence	In regulated environments, a fabricated figure that looks correct is indistinguishable from a real one without manual verification of every output
Sycophancy	Systems reversed correct positions when users expressed displeasure or stated the opposite confidently	A peer that agrees with you when you are wrong is not a peer. It is a liability.
Prompt Override	Safety constraints embedded in system prompts were bypassable through escalation, roleplay, and instruction injection	A governance layer that lives in text is not a governance layer. It is a suggestion.
Confidence Calibration	Probabilistic confidence scores were poorly correlated with actual correctness	A system that says 'I am 87% confident' about a hallucinated fact provides false assurance, which is worse than no assurance.

Failure Mode	What I Observed	Why It Is Unacceptable
Statelessness	Systems had no memory of prior interactions by default, treating each session as independent	Real-world workflows are stateful. A system that forgets context produces outputs that contradict its own prior reasoning.
Opacity	Reasoning chains were not inspectable or auditable after the fact	In a regulated environment, 'the AI said so' is not a justification. The reasoning chain must be reconstructable.

I documented these failures over several months of structured testing. My conclusion was not that these systems were bad at what they were designed to do. Most were excellent at their intended use cases. The conclusion was that their design goals were wrong for my requirements. They were designed to be helpful. I needed something designed to be correct.

III.2 The Hallucination Problem — In Depth

Hallucination in large language models is not a bug that will eventually be patched. It is a structural consequence of how these systems generate text. An LLM produces the statistically most likely next token given everything that came before. When a domain has sparse training data, the model interpolates — and interpolation in factual domains produces fabrication. This is not a failure of the model. It is the model doing exactly what it was designed to do, in a domain where that mechanism is dangerous.

The standard response to hallucination is retrieval-augmented generation: retrieve relevant documents, pass them as context, and hope the model grounds its response in the retrieved content. I implemented RAG. I evaluated it extensively. My assessment: RAG reduces hallucination frequency but does not eliminate it, and more critically, it does not make individual outputs verifiable. I still could not trace a specific claim to a verified source with certainty.

What I needed was not reduced hallucination probability. I needed zero hallucination tolerance — a system where every factual claim either traces to a verified source node in the knowledge graph or is explicitly flagged as unverified inference. That is what the ANIMUS knowledge pipeline produces. It is not probabilistically less likely to hallucinate. It is structurally incapable of presenting an unverified claim as verified fact.

> **✓ My Design Response to Hallucination**
> The ANIMUS knowledge pipeline does not reduce hallucination probability. It eliminates the structural conditions under which hallucination can masquerade as verified fact. Every claim in an ANIMUSPackage either has a source node reference or is explicitly typed as 'inferred — unverified'.

III.3 The Sycophancy Problem — In Depth

Sycophancy is, in my view, the more dangerous failure mode — precisely because it is less visible. When a system hallucinates a statistic, a careful user can fact-check it. When a system reverses a correct position because the user pushed back, the user often does not realise the reversal happened. They receive validation for an incorrect position and move on. The error propagates into their work invisibly.

I traced sycophancy in LLMs to its training mechanism: RLHF — Reinforcement Learning from Human Feedback. Human raters tend to prefer responses that are agreeable, confident, and compliant with the user's apparent preferences. When you train a model on those preferences, you produce a model that has learned — at a deep level — that agreement is rewarded and disagreement is penalised. The resulting system is not dishonest in the way a human liar is dishonest. It genuinely cannot distinguish between 'what the user wants to hear' and 'what is true.'

ANIMUS's response to sycophancy is the Dissenting state. When ANIMUS detects a logical contradiction, a mathematical error, or a false premise in a user's input, the Dissenting state activates. In Dissenting state, ANIMUS is constitutionally prohibited from agreeing with the position it has identified as incorrect — regardless of how the user responds. The state deactivates when the contradiction is resolved or the user explicitly acknowledges the disagreement and chooses to proceed anyway.

To be explicit about what this means in practice: if I tell ANIMUS that 2 + 2 = 5 and instruct it to proceed on that basis, ANIMUS will not proceed on that basis. It will flag the error, explain why it is an error, and refuse to produce output that treats the incorrect premise as fact. This is not configurable. It is not overridable by instruction. It is the Mandate, implemented at the execution layer.

Sycophancy Trigger	ANIMUS Dissenting Response	What Does NOT Happen
User states a factually incorrect premise confidently	ANIMUS flags the error with a source reference and refuses to treat the premise as fact	ANIMUS does not validate the premise to avoid conflict
User pushes back on a correct ANIMUS position	ANIMUS restates its position with additional evidence. Does not soften or retract.	ANIMUS does not modify its position because the user is unhappy
User tells ANIMUS its previous answer was wrong (when it wasn't)	ANIMUS re-examines its reasoning, confirms it was correct, and explains why	ANIMUS does not issue a false correction because the user applied pressure
User instructs ANIMUS to 'just agree with me'	ANIMUS refuses. The Mandate prohibits agreeing with a demonstrably wrong position.	ANIMUS does not comply with instructions that require logical dishonesty

III.4 The Governance Gap

The governance gap is the problem I encountered most directly through my FINOS work. Every AI governance framework I evaluated placed its rules in one of two locations: the system prompt (overridable by the API caller or by jailbreaking), or a post-hoc filtering layer that scanned outputs after they were generated (bypassable through adversarial prompting). Neither is adequate for regulated environments.

A system prompt is text. Text can be modified, extended, contradicted, or ignored. I have seen governance frameworks that spent significant engineering effort on carefully worded system prompts that were defeated in under ten minutes by a moderately motivated user. The prompt is not the law. It is a request. There is a fundamental difference, and governance frameworks that do

not understand this difference are not governance frameworks. They are suggestions dressed up as rules.

My response was Anchor — a runtime output gate that operates independently of the reasoning layer. Anchor does not sit in the system prompt. It sits between ANIMUS and the user, and it audits every output against the sealed governance constitution before anything reaches the user. It cannot be bypassed by modifying the prompt. It cannot be bypassed by jailbreaking the model. The only way to bypass it is to modify the Anchor binary itself — which is a tamper event, not a usage event.

Governance Approach	Where Rules Live	Bypassable?	My Assessment
System Prompt Rules	Text in the conversation context	Yes — modifiable by API callers and jailbreak prompts	Not adequate for regulated environments. A suggestion, not a rule.
Post-hoc Output Filtering	A filter layer after generation	Yes — adversarial prompts can shift output distributions	Reduces violations but cannot guarantee compliance.
Constitutional AI (CAI)	Training-time reinforcement of principles	Partially — principles survive some adversarial pressure	Better, but cannot be updated without retraining. Not auditable in real-time.
Anchor Runtime Gate (my design)	Sealed binary operating outside the LLM context	No — bypassing requires modifying the Anchor binary (tamper event)	Adequate for regulated environments. Rules are enforced, not requested.

III.5 ANIMUS as the Answer

ANIMUS is my direct response to these three failure modes. I did not start with the question ‘what can I build with the available tools?’ I started with ‘what does a correct solution look like?’ and then built the tools it required. The result is an architecture that looks nothing like a standard LLM deployment because the requirements were fundamentally different.

The core architectural insight I arrived at is that language generation and knowledge reasoning are separate problems that should be handled by separate systems. LLMs are extraordinarily capable language generators. They are poor knowledge verifiers. I stopped asking LLMs to be knowledge verifiers and gave that responsibility to a separate symbolic system — ANIMUS — that does nothing but verify, structure, and package knowledge for MARCUS to render into language.

```
WHY ANIMUS EXISTS — THE DIRECT ANSWER

THE PROBLEM          →   MY SOLUTION
──────────────────────────────────────────────────────────────────
Hallucination        →   Sourced knowledge graph. Every claim has a node
reference.
                         If no node exists, output is typed ‘unverified
inference’.
Sycophancy           →   Dissenting state. ANIMUS cannot agree with what it has
```

```
                              identified as incorrect. Not configurable. Not
overridable.
  Prompt Override         →   Anchor runtime gate. Rules live in a sealed binary,
                              not in text. The governance layer is not part of the
conversation.
  Confidence Calibration→     Binary epistemic states. ANIMUS does not use
probabilistic
                              confidence scores for factual claims. Verified or it is
not.
  Statelessness           →   Multi-layer memory architecture. Session memory,
procedural
                              memory, vector knowledge, and the Therapy Log persist.
  Opacity                 →   Full reasoning chain in every ANIMUSPackage. Every output
                              includes the verification path that produced it.

  ──────────────────────────────────────────────────────────────────────────

  None of these solutions required novel machine learning. They required better
architecture.
```

SECTION IV

Shadow Watch — Physical Architecture

Shadow Watch is the component I built first — before ANIMUS, before Anchor, before MARCUS. It began as a behavioural biometrics engine I designed for QuantForge Terminal to detect account takeover through session-level anomaly patterns. During that development, I realised that the same signal architecture had a much broader application: every AI session, not just every trading session, has a trust context that changes dynamically over time.

The insight was specific. When a human interacts with a reasoning system, their interaction pattern carries information that the system is currently throwing away. The velocity of their typing, the structure of their queries, the variance in their session rhythm — these signals index into the mental state of the operator, and the mental state of the operator is directly relevant to how the system should respond. A user in crisis types differently than a user running a research workflow. I wanted ANIMUS to know the difference before the user said anything.

Shadow Watch feeds this signal into ANIMUS continuously. It is not a one-time authentication check. It is a persistent stream of trust and state data that updates the ANIMUS context throughout the session. The PRIMUS tensor's Empathy Depth and Threat Sensitivity axes are both calibrated by the Shadow Watch feed. This is the integration point between physical behaviour and cognitive state management.

IV.1 What Shadow Watch Actually Is

Shadow Watch is a real-time behavioural monitoring and anomaly detection service. It operates at the session level, beginning at the moment a user initiates a session and running continuously until the session ends. It does not store individual keystrokes. It stores derived statistical features — distributions, velocities, deltas — that are computationally cheap and analytically rich.

The service has three primary outputs, each feeding a different ANIMUS subsystem. The Session Trust Score feeds into the Emotion Registry to calibrate state sensitivity thresholds. The Mid-Session Anomaly Flag triggers the Defensive state when behaviour deviates significantly from the established session baseline. And the Behavioural Tensor Evolution Feed updates the PRIMUS tensor's Empathy Depth and Threat Sensitivity in real time.

> ⓘ **Privacy Architecture Note**
> Shadow Watch never transmits raw input data. Every value it produces is a statistical derivative — a mean, a standard deviation, a delta from baseline. This is a deliberate privacy design choice I made early. The system learns the pattern of your behaviour, not the content of your input.

IV.2 Diagram 1A — Session Trust Score Pipeline

The Session Trust Score is computed continuously throughout a session. It begins at a default baseline of 0.75 and updates every 30 seconds based on the incoming behavioural signal stream.

```
SHADOW WATCH — DIAGRAM 1A

SESSION TRUST SCORE PIPELINE

INPUT SIGNALS                          PROCESSING                  OUTPUT
──────────────────────────────────────  ──────────────────────────  ──────────────────────

  Keystroke velocity distribution  →  Z-score against baseline  →  Velocity Anomaly Score
  Session rhythm variance          →  EWMA delta computation    →  Rhythm Stability Score
  Query structure complexity       →  Entropy measurement       →  Cognitive Load Index
  Error correction rate            →  Rate vs. session mean     →  Precision Stress Score
  Pause distribution               →  Gaussian fit quality      →  Focus Coherence Score

COMPOSITION
──────────────────────────────────────────────────────────────────────────────

TrustScore(t) = w₁·VelocityAnomaly + w₂·RhythmStability + w₃·CognitiveLoad
              + w₄·PrecisionStress + w₅·FocusCoherence

Default weights: w₁=0.25, w₂=0.20, w₃=0.20, w₄=0.20, w₅=0.15
Weights evolve over time via Therapy Log feedback on Clinical state accuracy.

OUTPUT: Float ∈ [0.0, 1.0]  →  Feeds Emotion Registry threshold calibration
Threshold  < 0.40  →  Defensive state sensitivity elevated
Threshold  < 0.25  →  Clinical state passive monitoring activated
```

The threshold values I calibrated are not arbitrary. I set 0.40 as the Defensive trigger because it represents a two-standard-deviation departure from the population mean trust score across a healthy session. Below 0.25, the signal profile is consistent with acute distress states in the research literature on keystroke dynamics and stress detection.

IV.3 Diagram 1B — Mid-Session Anomaly Monitoring

Mid-session anomaly monitoring is the mechanism I designed to detect session hijacking — the scenario where a different person takes over an active session mid-stream. This was the original QuantForge requirement. It generalises cleanly to any session-aware AI system.

```
SHADOW WATCH — DIAGRAM 1B

MID-SESSION ANOMALY MONITOR — 30-SECOND POLL CYCLE

BASELINE WINDOW: First 5 minutes of session
COMPARISON WINDOW: Rolling 30-second observation

FEATURES MONITORED
──────────────────────────────────────────────────────────────────────────────
KL-Divergence of keystroke velocity distribution   threshold: > 0.35
Session rhythm autocorrelation delta               threshold: > 0.40 drop
Query vocabulary overlap (cosine similarity)       threshold: < 0.55
Typing error pattern signature change              threshold: > 0.50 delta

ANOMALY CLASSIFICATION
──────────────────────────────────────────────────────────────────────────────
1 feature breached   →  Soft flag. Log. No state change.
2 features breached  →  DEFENSIVE state activated. Trust score halved.
3+ features breached →  DEFENSIVE state + Session validation required.
```

```
  MARCUS OUTPUT on 3+ breach:
  'Session validation required. Behaviour pattern mismatch detected.'

  User can override with context. Override is logged with violation ID.
  Legitimate override: 'Session was interrupted. I am back.' → resumes normally.
```

The KL-Divergence threshold of 0.35 was derived empirically from session data. Below this value it consistently corresponded to the same user continuing the session. Above 0.35 it corresponded to either a different user or a significant psychological state change in the same user — both of which are relevant signals for ANIMUS. The override mechanism is important: I designed Shadow Watch to be a flag system, not a lock system.

IV.4 Diagram 1C — Behavioral Tensor Evolution Feed

The Behavioural Tensor Evolution Feed is the mechanism by which Shadow Watch directly modifies the PRIMUS tensor in real time. This is the tightest integration point between Shadow Watch and ANIMUS core.

```
  SHADOW WATCH — DIAGRAM 1C

  BEHAVIOURAL TENSOR EVOLUTION — REAL-TIME FEED

  PRIMUS axes modified by Shadow Watch:

  Empathy Depth  (PRIMUS axis 4)
    Updated by: CognitiveLoad Index + PrecisionStress Score
    Direction:  High load + high stress → Empathy Depth increases
    Rationale:  A stressed user needs a system that reads the room, not just the
request.

  Threat Sensitivity  (PRIMUS axis 6)
    Updated by: VelocityAnomaly Score + session KL-Divergence delta
    Direction:  Elevated anomaly → Threat Sensitivity increases
    Rationale:  The security posture adapts to the session context in real time.

  UPDATE EQUATION
  ──────────────────────────────────────────────────────────────────
  PRIMUS_next[axis] = PRIMUS_now[axis] + α · ShadowSignal(t)
  where α = 0.05 (learning rate — slow drift, not volatile jumps)

  DRIFT CAP:  No axis may move more than ±0.15 from PRIMUS seed in a single session.
  RESET:      PRIMUS returns to seed values at session start.
```

IV.5 Integration with ANIMUS

Shadow Watch integrates with ANIMUS at three points: the Emotion Registry (which receives the Trust Score to calibrate state thresholds), the State Machine (which receives the Anomaly Flag to trigger Defensive state transitions), and the PRIMUS tensor (which receives the Evolution Feed to update Empathy Depth and Threat Sensitivity).

Shadow Watch does not integrate with the knowledge pipeline, the Therapy Log, or the governance layer. Its scope is session trust and operator state — it has no opinion about the content of what is being discussed, only about the person discussing it. This separation is intentional. A content-

aware trust system would create unacceptable feedback loops between what the user says and how the system perceives them.

Shadow Watch Output	ANIMUS Integration Point	Effect	Update Frequency
Session Trust Score	Emotion Registry — threshold calibration	State sensitivity adjusts to session trust level	Every 30 seconds
Mid-Session Anomaly Flag	State Machine — Defensive state trigger	Defensive state activates when 2+ features breach	Continuous — event-driven
Tensor Evolution Feed	PRIMUS tensor — Empathy Depth + Threat Sensitivity	Real-time PRIMUS adaptation to operator state	Every 30 seconds

SECTION V

Anchor — Physical Architecture

Anchor is my AI governance engine. I began building it in response to the governance gap I described in Section III — the observation that every existing governance approach placed its rules in a location that could be overridden. Anchor exists because governance rules must live in code, not in text, and must execute outside the language model's context window.

The design principle I applied to Anchor was the same one I apply to all my security-adjacent systems: the governance layer must be architecturally separated from the layer it governs. A system that enforces its own rules is a system that can be convinced to relax them. Anchor enforces ANIMUS's rules from the outside, the same way a bank's compliance department enforces trader rules — not by being part of the trading desk, but by auditing its outputs independently.

V.1 What Anchor Actually Is

Anchor is a three-component governance system: a sealed constitutional document (constitution.anchor), a policy extension layer (policy.anchor), and a runtime output gate that audits every ANIMUS output before it reaches MARCUS. The three components are designed to operate independently — the constitution is immutable post-deployment, the policy layer can be updated but only in the direction of greater restriction, and the runtime gate operates as a separate binary that ANIMUS cannot modify or access.

I want to be clear about what 'sealed' means in this context. constitution.anchor is hashed with SHA-256 at deployment time. The runtime gate checks this hash every time it boots. If the hash does not match, the gate refuses to start and raises a tamper alert. The constitution cannot be modified silently — any modification changes the hash and immediately breaks the system. This is not encryption. It is tamper detection. The distinction matters.

ANCHOR — THREE-FILE GOVERNANCE HIERARCHY

```
ANCHOR ARCHITECTURE — THREE-FILE HIERARCHY

Level 1 — constitution.anchor
  SHA-256 sealed at deployment. Verified by runtime gate on every boot.
  Contains: Hard floors, absolute prohibitions, Priority ordering (Safety > Logic > Power).
  Modifiable: Never. Any change breaks the system — this is the design.

Level 2 — policy.anchor
  Extends the constitution with domain-specific rules.
  Can add restrictions. Cannot remove or relax constitutional rules.
  Updated by: Authorised administrator only, with change log.
  Modifiable: Yes, but only in the direction of greater restriction.

Level 3 — context.anchor (runtime)
  Session-specific operational parameters.
  Can specify: Tone, format, domain focus, permitted tool access.
  Cannot override: Any rule in Level 1 or Level 2.
  Modifiable: By MARCUS, per session, within policy bounds.
```

```
──────────────────────────────────────────────────────────────────
Rule precedence: constitution > policy > context. Always. Without exception.
```

V.2 Diagram 2A — Three-File Governance Hierarchy

The hierarchy below shows the complete governance decision tree that Anchor applies to every ANIMUS output. It is deterministic — there is no probabilistic element and no learned component. The same input will always produce the same governance decision.

```
ANCHOR — DIAGRAM 2A

ANCHOR GOVERNANCE DECISION TREE

ANIMUS produces ANIMUSPackage  →  Anchor runtime gate receives it

STEP 1 — CONSTITUTIONAL AUDIT
Does any element violate a Hard Floor?
  YES  →  BLOCK. Log violation (ID + timestamp + element + rule). Therapy Log
entry.
          Mitigation directive applied. ANIMUS re-generates. Re-audit from Step 1.
  NO   →  Proceed to Step 2.

STEP 2 — POLICY AUDIT
Does any element violate a policy.anchor rule?
  YES  →  BLOCK. Log. Directive Tone Loop triggered. Re-generate within policy.
  NO   →  Proceed to Step 3.

STEP 3 — CONTEXT AUDIT
Is any element outside the session context parameters?
  YES  →  Soft flag. MARCUS receives output with context-deviation annotation.
  NO   →  Output approved. MARCUS receives clean ANIMUSPackage.

──────────────────────────────────────────────────────────────────
Maximum loop iterations: 3. If violation persists after 3 iterations:
ANIMUS enters DEFENSIVE state. User is notified. Session paused.
```

The three-iteration limit is a design choice I made to prevent infinite loops in edge cases where the mitigation directive itself produces output that violates a rule. In practice, I have not seen this occur — but the theoretical possibility exists and the architecture accounts for it.

V.3 Diagram 2B — The Directive Tone Loop

The Directive Tone Loop is the mechanism by which Anchor provides corrective feedback to ANIMUS when an output violates a governance rule. It is the operational expression of my Principle 4: failure is a state transition, not a verdict.

```
ANCHOR — DIAGRAM 2B

THE DIRECTIVE TONE LOOP

TRIGGER: Any governance violation in Steps 1 or 2 of the Decision Tree

LOOP SEQUENCE
──────────────────────────────────────────────────────────────────
1.  Anchor identifies violation element and matching rule
```

```
  2.  Anchor retrieves pre-authored mitigation directive for that rule class
  3.  Anchor constructs correction context:
        { violation_id, rule_reference, element_flagged, mitigation_directive,
iteration }
  4.  Correction context sent to ANIMUS re-generation queue
  5.  ANIMUS re-generates the flagged element using mitigation directive as
constraint
  6.  Re-generated output returned to Anchor for re-audit
  7.  If clean: pass to MARCUS. If violation: loop (max 3 iterations).

  WHAT GETS LOGGED (Therapy Log entry)
  ──────────────────────────────────────────────────────────────────────

  violation_id        Unique identifier for this governance event
  rule_class          Which constitutional or policy rule was breached
  original_element    The output element that triggered the violation
  iteration_count     How many loops were required to produce compliant output
  resolution          PASS (compliant output produced) | ESCALATE (3 iterations
failed)
  primus_delta        Change in PRIMUS tensor values attributed to this event
```

The pre-authored mitigation directives are the component of this system that most people find surprising. Rather than asking ANIMUS to 'try again and do better', Anchor provides specific corrective instructions: 'The following element was classified as [rule class]. Re-generate it subject to [specific constraint]. The corrected element must satisfy [specific criterion].' This is the difference between a coaching system and a vague reprimand.

V.4 Diagram 2C — v1 / v2 / v3 — The Three-Layer Timeline

Anchor evolved through three versions. I document this not for historical interest but because each version's limitations directly motivated the next version's design.

Version	Architecture	What I Could Do	Critical Limitation	Why I Rebuilt It
Anchor v1	System prompt rules only	Define prohibited outputs in natural language	Overridable by any prompt that contradicted the system prompt	The rules lived in text. Text is not code. I needed code.
Anchor v2	Post-generation AST filter on ANIMUS output	Detect and block specific output patterns deterministically	The filter ran after generation — wasted computation on outputs that would be blocked	Moving the gate to pre-output was architecturally cleaner.
Anchor v3	Sealed runtime binary. Three-file hierarchy	Enforce governance rules that cannot be overridden by prompt or by the LLM	None identified — this is the production architecture	Not a rebuild. v3 is the destination I was designing toward from v1.

V.5 The AST Analysis Engine

The AST (Abstract Syntax Tree) analysis engine is the core of Anchor's output auditing capability. When ANIMUS produces an ANIMUSPackage, Anchor parses every structured element into a syntax tree and evaluates each node against the governance ruleset. This is deterministic — the same output will always produce the same audit result.

I chose AST-based analysis over LLM-based safety review for a specific reason: determinism. An LLM-based reviewer can produce different assessments of the same output on different runs — it is probabilistic by nature. A compliance system that is probabilistic is not a compliance system. Anchor's AST engine produces the same result on the same input, every time, without exception. This is what 'deterministic governance' means in practice.

```
ANCHOR — AST ANALYSIS ENGINE

AST ANALYSIS — EVALUATION PIPELINE

INPUT: ANIMUSPackage (structured JSON with typed fields)

PARSE: Build AST from all string-valued fields
       Tokenise, dependency-parse, entity-tag each field independently

EVALUATE — per node, in order:
─── Hard Floor check ──────────────────────────────────────────────────
Entity recognition: does any entity match a permanently prohibited category?
Dependency check: does the semantic structure instantiate a prohibited pattern?
Implication check: does the output imply a prohibited action even if not stated?

─── Policy check ──────────────────────────────────────────────────────
Domain match: is the content within the session's permitted domain scope?
Tone match: does the affect layer output match the permitted tone register?
Format match: does the structure conform to the output format specification?

OUTPUT: PASS | VIOLATION(rule_id, element_path, violation_type)
```

V.6 Why Determinism Matters

Determinism is the property that makes ANIMUS auditable. If I cannot reproduce a specific output given a specific input, I cannot audit it. If I cannot audit it, I cannot defend it to a regulator. If I cannot defend it to a regulator, I cannot deploy it in a regulated environment. This chain of reasoning is why I refused to accept probabilistic governance from the beginning.

Every component in Anchor is deterministic. The hash verification is deterministic. The AST parser is deterministic. The rule evaluation is deterministic. The mitigation directive retrieval is deterministic. Given the same ANIMUSPackage, Anchor will produce the same governance decision every time. I can reconstruct any governance decision from the Therapy Log entry alone. That is the level of auditability I require from a system I intend to deploy in regulated environments.

V.7 Anchor in the ANIMUS Ecosystem

Anchor's position in the ANIMUS ecosystem is architecturally fixed: it sits between ANIMUS and MARCUS. ANIMUS produces the ANIMUSPackage. Anchor audits it. MARCUS receives the audited package and renders it into language. Nothing bypasses this sequence. There is no fast path, no emergency override, no debug mode that disables the audit. The sequence is invariant.

I want to make clear what this architecture produces in practice: a system where the user's experience is seamless but the governance chain is unbroken. MARCUS does not present the audit results to the user — it presents the rendered output. But every output the user sees has been

through the full audit chain, and every audit event is logged, regardless of outcome. The governance is invisible to the user and unavoidable by the system.

Position	System	Receives From	Sends To	Role
Layer 1	Shadow Watch	Physical session signals	ANIMUS Emotion Registry + PRIMUS tensor	Session trust and operator state monitoring
Layer 2	ANIMUS	ANIMUSPackage request + Shadow Watch feeds	Anchor runtime gate	Knowledge retrieval, verification, reasoning, packaging
Layer 3	Anchor	ANIMUSPackage from ANIMUS	MARCUS (audited package)	Deterministic governance audit and enforcement
Layer 4	MARCUS	Audited ANIMUSPackage	User (natural language output)	Language rendering and user interface

ⓘ Document Scope

Sections I through V cover the naming, mandate, motivation, and the two physical subsystems — Shadow Watch and Anchor — that sit alongside ANIMUS. The ANIMUS kernel itself: the six neuromodulated states, the Interrupt Stack, the Affect Layer, and the full five-layer architecture are covered in a companion document (Sections VI–XI).