\*\*A Total System of Memory‑Native, Verifiable, Agentic Intelligence\*\* Edition v0.1 (Master Index) --- ## Front Matter \* Foreword (Origins, Intent, Acknowledgments) \* Preface (How to Read This Book; Reading Roadmaps: Researcher • Engineer • Builder) \* Conventions (Typography, Symbols, Units, Pseudocode, ACL formatting) \* Notation & Symbols Quick Reference (RS, IDS, DD, κ, û, TPV, SEG, SDF‑CVF, etc.) \* Summary of Core Invariants (CMC, APOE, VIF, SDF‑CVF, SEG) \* Executive Overview (5‑page Brief for Decision‑Makers) --- ## Part I — First Principles & Invariants

\*\*Chapter 1. The Why\*\*

1.1 The Machine‑Communication Thesis (Beyond “chat”)

1.2 From Context to Memory (CMC)

1.3 From Improvisation to Compilation (APOE)

1.4 From Claims to Witness (VIF)

1.5 From Drift to Atomic Evolution (SDF‑CVF)

1.6 From Documents to Evidence Graphs (SEG)

\*\*Chapter 2. System Axioms\*\*

2.1 Memory Invariant (CMC) — Formal statement & proofs

2.2 Orchestration Invariant (APOE) — Formal statement & proofs

2.3 Witness Invariant (VIF) — Trust calculus & auditability

2.4 Atomic Evolution (SDF‑CVF) — Compositionality & parity

2.5 Substrate Invariant (SEG) — Graph theoretic underpinnings

2.6 Axioms → Design Constraints (traceability table)

\*\*Chapter 3. Design North‑Star\*\*

3.1 C‑3PO‑with‑a‑Workshop: Roles & Scopes

3.2 Safety as Capability (abstention, κ‑gating)

3.3 IDE‑in‑the‑Loop Rationale

3.4 Threat & Misuse Model (policy surface) ---

## Part II — Context Memory Core (CMC) & HHNI

\*\*Chapter 4. Fractal Memory Hierarchy\*\*

4.1 Atoms & Molecules (schemas)

4.2 Atomicity, Recursion, Emergence

4.3 Modality Independence & Payload Abstraction

4.4 Embeddings, Tags, and TPV

\*\*Chapter 5. Hyper‑Hierarchical Neural Indexing (HHNI)\*\*

5.1 Indices from System→Section→Paragraph→Sentence→Word→Sub‑word

5.2 Dependency Hashing & Impact Previews

5.3 Priority & Decay (TPV dynamics)

5.4 Retrieval Score RS (QS · IDS · (1−DD)) — normalization & calibration

\*\*Chapter 6. Write/Read Pipelines\*\*

6.1 Ingest→Atomize→Propose→DD→Gate→Snapshot

6.2 Hierarchical Read with Dumbbell Compression

6.3 Caching & Preloading (Markov traversals)

6.4 Performance Targets & SLOs

\*\*Chapter 7. Storage Substrate\*\*

7.1 Vector Store & KNN

7.2 Evidence Graph Store (temporal)

7.3 Object Store (content‑addressed)

7.4 Snapshot Log & Rollback ---

## Part III — Dynamic Vector Navigation System (DVNS)

\*\*Chapter 8. Physics‑Inspired Retrieval\*\*

8.1 GODN Recap (grav, elastic, repulse, damping)

8.2 Path Cost C(P) (length, smoothness, blocks, density)

8.3 Stability & Abstention

8.4 Policy‑Aware Geometry (forbidden crossings)

\*\*Chapter 9. Algorithms & Analysis\*\*

9.1 Force Integration (Verlet vs. Euler)

9.2 Convergence & Complexity

9.3 Super‑Index Precomputation

9.4 Empirical Lift vs. Static KNN

\*\*Chapter 10. Implementation Patterns\*\*

10.1 Online Refinement on Top‑k

10.2 Parameter Tuning (G,k,δ,c,α,β,γ)

10.3 Failure Modes (oscillation, collapse)

10.4 Visualization & Telemetry ---

## Part IV — Orchestration: APOE & DEPP

\*\*Chapter 11. Cognitive Compilation (APOE)\*\*

11.1 Roles & Contracts (planner, retriever, reasoner, verifier, builder, critic, operator, witness)

11.2 Budgets (tokens,time,tools)

11.3 κ & Uncertainty Emission at Boundaries

11.4 Routing & Parallelization \*\*Chapter

12. Dynamic Emergent Prompt Pipeline (DEPP)\*\*

12.1 Master Chain as Graph

12.2 Self‑Rewrite via Evidence

12.3 Chain Health Metrics (κ\_chain, budget adherence)

12.4 Large‑Scale Planning (thousands of nodes) \*\*Chapter

13. ACL — AIMOS Chain Language\*\*

13.1 Grammar (EBNF)

13.2 Core Constructs (pipeline, step, gate, budget, inputs)

13.3 Static Checks & Type System

13.4 Examples (research, coding, operations) ---

## Part V — Verifiable Intelligence: VIF & SEG

\*\*Chapter 14. Provenance & Uncertainty\*\*

14.1 VIF Metadata (model, weights hash, prompts, data slices, tools)

14.2 UQ Methods (temperature sweeps, ensembles, ECE)

14.3 Confidence Bands & UI Cues

14.4 Reproducibility & Replays

\*\*Chapter 15. Shared Evidence Graph (SEG)\*\*

15.1 JSON‑LD Schema & SHACL

15.2 Temporal Graphs & Time‑Slicing

15.3 Claim→Evidence→Decision Lineage

15.4 Export & Regulatory Packages ---

## Part VI — Atomic Evolution: SDF‑CVF

\*\*Chapter 16. Atomic Commits Across Code/Docs/Tags/Traces\*\*

16.1 Gate Catalog (parity, tests, safety, provenance, HITL)

16.2 ADRs & Branching Strategy

16.3 Auto‑Fix, Quarantine, Rollback

16.4 DORA‑aligned Metrics (CFR, MTTR)

\*\*Chapter 17. Human‑in‑the‑Loop (HITL)\*\*

17.1 Risk Scoring & Two‑Key Approval

17.2 Review UX & Diff Semantics

17.3 Audit Trails & Accountability

17.4 Training Data Hygiene ---

## Part VII — IDE‑in‑the‑Loop: From Prompt to Running App

\*\*Chapter 18. Blueprint→Scaffold→Code→Test→Deploy→Observe\*\*

18.1 Blueprint YAML 18.2 Scaffolding Strategies

18.3 Codegen & Refactors

18.4 Testing Pyramid & Coverage Gates

18.5 Preview Environments & Telemetry

18.6 Parity Enforcement (spec↔code)

\*\*Chapter 19. Templates & Components\*\*

19.1 UI, API, Auth, Data

19.2 Policy Packs (PII, licenses)

19.3 Integration Patterns (tools, APIs)

19.4 Cost & Energy Budgets ---

## Part VIII — Security, Safety, & Compliance

\*\*Chapter 20. Threat Model & Guardrails\*\*

20.1 Prompt Injection, Exfiltration, SSRF

20.2 Tooling Sandboxes & Capability Tokens

20.3 Redaction & Differential Privacy

20.4 Abstention & Containment

\*\*Chapter 21. Compliance Engineering\*\*

21.1 EU AI Act (logging, synthetic labeling)

21.2 SOC2/ISO Mapping

21.3 Data Retention vs. Minimization

21.4 Incident Response & Notifications ---

## Part IX — Evaluation, Benchmarks, & Observability

\*\*Chapter 22. Retrieval & Reasoning Benchmarks\*\*

22.1 p@k, nDCG, RS‑lift 22.2 κ/ECE Dashboards

22.3 Streamed Agents & Long‑Horizon Tasks

22.4 Gold Sets & Adversarial Suites

\*\*Chapter 23. Observability & Costing\*\*

23.1 OpenTelemetry for Agents

23.2 Token/Latency Budgets

23.3 Drift & Prompt‑rot Monitors

23.4 SLOs & Error Budgets ---

## Part X — Case Studies & Patterns

\*\*Chapter 24. Machine Communication\*\*

24.1 Protocol Translation

24.2 Negotiation & Mediation

24.3 Device & File Diplomacy

\*\*Chapter 25. Builder Programs\*\*

25.1 Greenfield App

25.2 Brownfield Refactor

25.3 Compliance‑Critical Workflow

\*\*Chapter 26. Ops & Incidents\*\*

26.1 Log Forensics

26.2 RCA with VIF/SEG

26.3 Safe Autoremediation ---

## Part XI — Reference Implementations

\*\*Chapter 27. Data Schemas\*\*

27.1 Atom/Edge

27.2 SEG JSON‑LD

27.3 ACL AST

\*\*Chapter 28. APIs\*\*

28.1 Memory API

28.2 Orchestration API

28.3 Audit API (read‑only)

\*\*Chapter 29. SDKs & Clients\*\*

29.1 JS/TS

29.2 Python

29.3 CLI ---

## Part XII — Mathematics & Formal Methods

\*\*Chapter 30. Retrieval Math\*\*

30.1 RS/QS/IDS/DD definitions & bounds

30.2 Normalization & Calibration

30.3 UQ Theory (entropy, variance, calibration)

\*\*Chapter 31. Dynamics\*\*

31.1 GODN Potentials & Forces

31.2 Convergence Proof Sketches

31.3 Stability Regions

\*\*Chapter 32. Type Systems & Logic\*\*

32.1 ACL Types & Effects

32.2 Symbolic Checkers

32.3 Soundness Notes ---

## Part XIII — Roadmap & Governance

\*\*Chapter 33. 0‑30‑60‑90\*\*

33.1 Milestones

33.2 Risks & Dependencies

33.3 Resourcing & Budgets

\*\*Chapter 34. Community & Contribution\*\*

34.1 ADR Templates

34.2 Review Rotations

34.3 Ethics & Use Policies ---

## Appendices

A. Glossary (expanded)

B. Acronyms

C. Extended Examples (end‑to‑end traces)

D. Figures & Tables List

E. Change Log

F. Index of Indexes (HHNI cross‑reference)

G. Quickstarts & Labs

H. Troubleshooting Guide

I. Compliance Artifacts (sample exports)

J. Risk Taxonomy & Playbooks --- ## Back Matter \* Bibliography \* Subject Index \* Author/Editor Notes --- ### Crosswalk Matrices (At‑a‑Glance) \* Invariants → Components → Gates → KPIs \* Data Flow (Write/Read) → Traces → Compliance Duties \* Roles (APOE) → Tools → Budget Classes --- ### Expansion Plan (Serial Order for Subsequent Chats)

1. Part I Ch.1–3

2. Part II Ch.4–7

3. Part III Ch.8–10

4. Part IV Ch.11–13

5. Part V Ch.14–15

6. Part VI Ch.16–17

7. Part VII Ch.18–19

8. Part VIII Ch.20–21

9. Part IX Ch.22–23

10. Part X Ch.24–26

11. Part XI Ch.27–29 12.

Part XII Ch.30–32 13.

Part XIII Ch.33–34 14.

Appendices & Back Matter

**Chapter 1 — The Why**

**1.1 The Machine-Communication Thesis (Beyond “chat”)**

**Thesis**

**Large models are not “chatbots.” They are machine-communication engines: mediators that translate among people, documents, tools, files, networks, and code—while remembering, orchestrating, witnessing, and evolving in verifiable ways.  
AIMOS/APOE turns this thesis into an operating system:**

* **CMC (Context Memory Core): converts ephemeral context into structured, queryable, reversible memory.**
* **APOE (Prompt Orchestration Engine): compiles reasoning into typed plans, not one-shot guesses.**
* **VIF (Verifiable Intelligence Fabric): every claim ships with lineage, uncertainty, and replayability.**
* **SDF-CVF (Atomic Evolution): code, docs, tags, and traces evolve as one through gates.**
* **SEG (Shared Evidence Graph): evidence is a graph, time-sliced and exportable.**

**Together, these make “chat” merely the UI skin over a disciplined, auditable machine-to-machine protocol.**

**Why “chat” fails at scale**

1. **Ephemerality. Conversation context evaporates; decisions cannot be reproduced.**
2. **Improvisation. One-shot prompting has no budgets, no roles, no contracts.**
3. **Opacity. Outputs lack provenance, uncertainty, or replays; trust is performative.**
4. **Drift. Code/spec/tests/provenance diverge; fixes are tribal knowledge.**
5. **Silolization. Documents, tools, and time live in separate islands; no cohesive lineage.**

**Conclusion: “Better prompts” cannot fix structural defects. Architecture must.**

**What “machine communication” requires**

**1) Memory-native IO.  
All IO (prompts, files, API calls, logs) lands first as atoms in CMC → indices (HHNI) → snapshots → graph entries in SEG.**

**2) Compiled reasoning.  
Tasks are plans (DAGs) with roles, tools, and explicit budgets. Plans run deterministically (or abstain).**

**3) Verifiable speech.  
Every artifact carries VIF: model id, weights hash, prompt template id, tool calls, snapshot id, and UQ (calibrated confidence).**

**4) Atomic evolution.  
A single commit spans code/docs/tags/traces, gated by tests/spec/provenance/HITL. Quarantine on failure; rollback is cheap.**

**5) Graph substrate.  
Claims, sources, tools, decisions live as typed nodes/edges with temporal validity; contradictions are first-class, not deletions.**

**Minimal surface: the five invariants as protocol layers**

* **Link layer: CMC — atoms, indices, snapshots (write once, cite forever).**
* **Network layer: APOE — typed plans, budgets, routes, parallelization.**
* **Transport layer: VIF — provenance packets and UQ headers at each hop.**
* **Session layer: SDF-CVF — parity gates across artifacts; HITL checkpoints.**
* **Application layer: SEG — time-sliced evidence queries and exports.**

**Design consequences (non-negotiable)**

1. **Snapshot-first IO. Nothing user-visible exists without a snapshot id.**
2. **Two-stage reads. Coarse → fine retrieval, with dumbbell compression to avoid “lost in the middle.”**
3. **Budgets at boundaries. Each step declares tokens/time/tools; breaches abort or route to HITL.**
4. **Witness or abstain. If RS < κ or calibration poor, the system must decline or escalate.**
5. **Atomic parity. No doc/spec change merges without corresponding tests/code/provenance passing gates.**
6. **Temporal truth. SEG queries are as-of time; deprecations leave tombstones and contradiction edges.**

**Core mechanics (concise)**

* **Retrieval score: RS = norm(QS · IDS · (1 − DD)) ∈ [0,1]**
  + **QS content quality, IDS indexing depth, DD dependency delta.**
* **Abstention threshold: κ (decline when RS < κ or UQ > κ).**
* **DVNS refinement: post-KNN force-guided pathing (gravity/elastic/repulse/damping) to improve Top-k precision under policy constraints.**

**Worked mini-scenarios**

**A) Protocol translation (machine↔machine).  
Input: “Convert this vendor XML feed into our JSON schema, validate, and post to /ingest.”  
Flow: Ingest → CMC atoms/snapshot → APOE plan (mapping, validation, post) → VIF at each step → SEG lineage.  
Result: Auditable transform with replayable witnesses and contract-level budgets.**

**B) File diplomacy (device↔file↔policy).  
Input: “Summarize this 3GB log, redact PII, and propose remediations.”  
Flow: Chunk→index (HHNI) → DVNS selects salient spans → redact per policy → APOE proposes fixes → κ/HITL on high risk → SEG anchors.  
Result: Redaction proof + remediation plan, both replayable.**

**C) IDE-in-the-loop build (user↔code).  
Input: “Scaffold a browser app, OAuth2 login, audit log export.”  
Flow: Blueprint→APOE (scaffold, codegen, tests) → SDF-CVF gates → preview env → SEG export pack.  
Result: Running app whose spec↔code↔tests↔traces are in parity.**

**KPIs (what improves when you leave “chat” behind)**

* **Memory: p@k@latency↑, RS-lift vs. static KNN, snapshot-replay fidelity.**
* **Orchestration: % deterministic plan replays at fixed seeds; budget adherence.**
* **Trust: lineage completeness %, ECE (calibration) bands, κ true-positive abstentions.**
* **Evolution: change-failure rate↓, MTTR↓, spec↔code parity score↑.**
* **Compliance: export time (mins), redaction coverage %, contradiction capture rate.**

**Anti-goals (what we refuse to do)**

* **Hallucinated citations or unverifiable claims.**
* **Hidden tool usage or mutable, unlogged state.**
* **“One-shot miracles” without plans, budgets, or witnesses.**
* **Silent drift between docs, code, tests, and traces.**

**Risks & mitigations**

* **Prompt injection / exfiltration: tool sandboxes, capability tokens, redaction by default, policy-aware DVNS.**
* **Overconfidence: mandatory UQ, κ-abstention, calibration dashboards.**
* **Plan bloat: ACL static checks, budget caps, chain health metrics (κ\_chain).**
* **Governance fatigue: HITL two-key only on R2/R3 risk tiers; everything else auto-merges under gates.**

**One-paragraph summary**

**Machine communication replaces “chat” with a memory-native, plan-driven, witness-first discipline. AIMOS/APOE operationalizes this with CMC (remember), APOE (compile), VIF (prove), SDF-CVF (evolve atomically), and SEG (graph the evidence). The result is a system that can translate, build, and decide—and show its work every step of the way.**

**1.2 From Context to Memory (CMC)**

**Problem**

**Raw prompts are ephemeral. Without structure, yesterday’s hard-won insight can’t be found, reused, audited, or reversed. “Chat logs” are not memory; they’re sediment.**

**Principle (Memory Invariant)**

**An intelligent system must convert ephemeral context into structured, queryable, and reversible memory.**

**CMC makes this operational by turning every input/output/tool call into atoms → indices → snapshots → evidence—the four layers of a memory-native stack.**

**A. The Four Layers**

1. **Atoms (what exists)  
   Minimal, typed facts created on ingest:**

* **id, ts, author, modality{text|code|event|tool}, payload, aux{tags, spans, pii\_mask}**
* **embedding (vector), tpv (Tag Priority Vector: priority, relevance, decay)**

1. **Indices (how it’s found) — *HHNI*  
   Fractal, recursive indices from System → Section → Paragraph → Sentence → Word → Sub-word, with:**

* **dependency edges (parent/child, supports, contradicts, relates\_to)**
* **IDS (Indexing Depth Score) and impact previews for safe edits**

1. **Snapshots (how it is frozen)  
   Content-addressed, immutable bundles of {atoms + indices + metadata}.**

* **Signed digest, redaction map, policy labels**
* **Reversibility: restore(snapshot\_id) → (atoms, indices)**

1. **Evidence (how it is trusted) — *SEG*  
   JSON-LD graph nodes for claims/sources/tools/decisions with temporal validity.**

* **Lineage: (model, weights\_hash, prompt\_template\_id, tool\_calls, snapshot\_id)**
* **Contradictions are edges, not deletions.**

**B. Write Path (Context → Memory)**

**Pipeline: ingest → atomize → enrich → index → score → gate → snapshot → graph**

* **Ingest & Atomize. Normalize IO (prompt, file, API, log) into typed atoms.**
* **Enrich. Summaries, tags, PII masking, embeddings; compute QS (quality).**
* **Index. Insert into HHNI; compute IDS (depth/coverage).**
* **Score. Compute DD (dependency delta) if editing linked content.**
* **Gate (CMC-G). Require thresholds: QS≥θ\_q, IDS≥θ\_i, DD≤θ\_d.**
* **Snapshot-first. Produce snapshot\_id; nothing surfaces without it.**
* **Graph. Emit VIF lineage into SEG (with time slice [t\_from, t\_to)).**

**Retrieval Score (used later by readers):**

**C. Read Path (Memory → Context)**

**Two-stage read with “dumbbell compression”:**

1. **Coarse select. Vector KNN on embeddings + HHNI filters (scope, time, policy) → candidate packs.**
2. **Refine.**
   * **Dumbbell: keep early and late critical spans, compress the middle (prevents “lost-in-the-middle”).**
   * **DVNS (optional): physics-guided reranking (gravity/elastic/repulse/damping) under policy constraints.**
3. **Rehydrate. Expand final spans back to atoms; attach provenance, masks, and snapshot\_id.**

**If RS < κ or UQ is poor → abstain or route to HITL.**

**D. Data Contracts (concise)**

**Atom (core)**

**Atom {**

**id, ts, author, modality, payload, summary,**

**tags[], embedding[], tpv{priority, relevance, decay},**

**pii\_mask, parent\_molecule\_id?**

**}**

**Index Edge**

**Edge {**

**src\_id, dst\_id, type{contains|supports|contradicts|relates\_to},**

**weight, dependency\_hash, impact\_preview[]**

**}**

**Snapshot**

**Snapshot {**

**id, digest, redactions[], policy\_labels[], atom\_refs[], index\_refs[],**

**created\_by, created\_at**

**}**

**SEG Node (claim/tool/decision)**

**VIF {**

**artifact\_id, model, weights\_hash, prompt\_template\_id,**

**tool\_calls[], snapshot\_id, uq{entropy, ece, band}, valid\_from,to?**

**}**

**E. Guarantees**

* **Reversibility. restore(snapshot) → (atoms, indices); diffs are explicit.**
* **Monotonicity. Graph grows with time; deprecations leave tombstones/contradictions.**
* **Determinism (under fixed state). Given (M\_t, tools, seeds) the same read returns the same pack (within tolerance) or abstains.**
* **Privacy by design. PII masked at ingest; redaction maps carried in snapshots; policy labels enforced at read.**

**F. Anti-Patterns (what CMC prevents)**

* **Blob memory. Unstructured chat logs; impossible lineage.**
* **Append-only amnesia. No indices, no decay, no retrieval control.**
* **Mutable facts. Silent edits without snapshots; untraceable changes.**
* **Monolithic context. Huge prompts with LIM; no dumbbell or HHNI.**

**G. Minimal Pseudocode**

**def write\_context(io):**

**atoms = atomize(io) # modality-aware**

**enrich(atoms) # summary, tags, embeddings, QS**

**idx = index\_hhni(atoms) # IDS, edges, hashes**

**dd = dependency\_delta(idx) # impact of edits**

**if not gate(qs=atoms.QS, ids=idx.IDS, dd=dd): raise GateReject**

**snap = snapshot(atoms, idx) # content-addressed**

**emit\_vif(snapshot=snap, io=io) # SEG lineage**

**return snap.id**

**def read\_memory(query, t=None, policy=None):**

**cands = knn(query.embedding, scope=(t, policy))**

**refined = dumbbell\_compress(cands)**

**packed = dvns\_refine(refined, policy=policy) # optional**

**if packed.RS < kappa or packed.UQ.bad: return Abstain**

**return rehydrate(packed) # atoms + provenance + snapshot\_id**

**H. KPIs (for CMC)**

* **p@k@latency on retrieval; RS-lift vs. static KNN**
* **Snapshot replay fidelity (exact/functional)**
* **Lineage completeness % (atoms with VIF)**
* **Edit safety: mean DD, count of blocked high-impact edits**
* **Privacy coverage: PII mask hit-rate, redaction drift**

**I. Example (before/after)**

**Before (chat):  
“Summarize the 3GB logs and suggest fixes.” → One-off answer; no sources; can’t reproduce.**

**After (CMC):**

* **Logs atomized; masked; embedded.**
* **HHNI links errors↔deploy events; snapshot\_id=S-8421.**
* **DVNS elevates salient spans; APOE proposes remediations.**
* **Output cites S-8421, tool calls, model hash, and UQ band.**
* **A week later, replay(S-8421) reproduces the same pack for audit.**

**J. Design Constraints (binding)**

* **C-1 Single writer. All writes via CreateMemory transactional flow.**
* **C-2 Snapshot-first. Nothing surfaces without a snapshot\_id.**
* **C-3 Two-stage read. Coarse→refine with dumbbell; DVNS optional.**
* **C-4 Policy-aware. PII masks & labels must propagate into every read pack.**
* **C-5 Witness-ready. Every read result carries VIF suitable for SEG insertion.**

**1.3 From Improvisation to Compilation (APOE)**

**Problem**

**Single-shot prompting improvises reasoning. It drifts with phrasing, is hard to reproduce, and hides cost/risk. Complex work (research, coding, operations) needs plans, not vibes.**

**Principle (Orchestration Invariant)**

**Reasoning must be compiled, not improvised.**

**APOE (AIMOS Prompt Orchestration Engine) turns intent into a typed, budgeted, auditable plan; executes it; and emits a witnessed trace you can replay.**

**A. What APOE Is**

**Plan is a finite DAG over steps , edges define data/control flow.**

**Step signature**

* **Inputs: atoms/read-packs from CMC, constants, prior step outputs.**
* **Tools: LLM calls, RAG queries, code runners, HTTP, sandboxed shells.**
* **Budgets: .**
* **Artifacts: text/code/config/test/results.**
* **Witness: VIF provenance + UQ vector (entropy/calibration) + snapshot ids.**

**Roles *(typed executors)*: planner, retriever, reasoner, builder, verifier, critic, operator, witness.  
Determinism under fixed state: With fixed (memory), tools, seeds, and budgets, yields the same trace (or abstains at ).**

**B. Compile-Time vs Run-Time**

**Compile-time (before any LLM call):**

1. **Intent parse → task graph. Decompose goals; annotate edges with data/decision dependencies.**
2. **Budgeting. Allocate token/time/cost envelopes per step; define global caps.**
3. **Static checks. Type/effect checks (e.g., “tool:db.write” forbidden in read-only plan), PII policy, cyclicity.**
4. **Acceptance criteria. Per step success predicates & gates (tests, RS thresholds, UQ bands).**
5. **Emission of ACL. Plan lowered into ACL (AIMOS Chain Language) for execution.**

**Run-time (when executing):**

1. **Routing & parallelization. Topologically schedule; run independent branches in parallel; cache stable subplans.**
2. **Boundary gates. Each step emits VIF+UQ; if band or RS< → repair, fallback, or abstain.**
3. **Self-rewrite via evidence. If a gate fails with actionable evidence, planner may edit the plan, bump version, and continue (bounded by rewrite budget).**
4. **Witness trace. All inputs/outputs/tool calls/snapshots recorded to SEG.**

**C. Minimal Formalism**

**Typed step:**

* **: set of atoms/read-packs; : new/modified artifacts; : witness node(s).**
* **Gate accepts iff all predicates hold: tests, RS, policy, UQ.**
* **Abstention : if (from calibrated UQ), the step returns ABSTAIN.**

**Objective (multi-criteria)  
Minimize   
subject to: All gates pass; global risk ; budgets respected.**

**D. APOE Lifecycle (concise)**

1. **Plan Synthesis.  
   Input: user goal + context sketch → plan draft with roles, tools, budgets, gates.**
2. **Compilation (ACL).  
   Lower draft into ACL with types, effects, and static guarantees.**
3. **Execution.  
   Orchestrator runs steps; retrieval uses CMC (RS-aware); reasoning uses LLM; verification runs tests/critics.**
4. **Witnessing.  
   For each boundary: record model id, weights hash, prompt template id, tool calls, snapshot ids, UQ.**
5. **Adaptation.  
   If evidence suggests a better route, perform bounded DEPP self-rewrite (new ACL version, linked in SEG).**
6. **Closeout.  
   Emit artifacts + replayable trace; update CMC; surface KPIs.**

**E. Anti-Patterns APOE Eliminates**

* **One-shot megaprompting. No budgets, no gates, brittle outcomes.**
* **Hidden tool use. No lineage or policy check; exfiltration risk.**
* **Heuristic retries. “Try again” loops without calibrated UQ or acceptance tests.**
* **Spec/code drift. Output without parity gates or atomic commits.**

**F. Constraints (binding)**

* **O-1 Budgets explicit. Tokens/time/tools/cost must be declared per step and enforced.**
* **O-2 Typed roles. Steps must declare role and permitted effects (read, write, network, exec).**
* **O-3 Boundary gates. Every step ends with verifiable gates (tests/RS/UQ/policy) and emits VIF.**

**G. Tiny Example (research→draft→critique)**

**ACL sketch**

**pipeline ResearchNote(goal)**

**step plan as planner**

**inputs = {goal}**

**budgets = {tokens:4k, time:30s}**

**out = outline**

**gate = {nonempty(outline)}**

**step retrieve as retriever**

**inputs = {outline}**

**tools = {rag.search}**

**budgets = {tokens:3k}**

**out = sources[]**

**gate = {p\_at\_5(sources) >= 0.8, policy.ok(sources)}**

**step draft as reasoner**

**inputs = {outline, sources}**

**tools = {llm.write}**

**budgets = {tokens:6k}**

**out = note.md**

**gate = {UQ.band in [A,B], cites(sources)}**

**step critique as critic**

**inputs = {note.md}**

**tools = {llm.review}**

**out = issues[]**

**gate = {len(critical(issues)) == 0}**

**witness all**

**end**

**Execution result: note.md + SEG lineage; replay uses the same snapshots & seeds.**

**H. Pseudocode (orchestrator core)**

**def run(plan):**

**state = {}**

**for s in topo\_sort(plan):**

**x = collect\_inputs(s, state, CMC)**

**y, w = execute(s, x) # tools bounded by budgets**

**if not gate\_pass(s, y, w):**

**decision = repair\_or\_abstain(s, x, y, w)**

**if decision == "repair": plan = bounded\_rewrite(plan, s, evidence=w)**

**else: return ABSTAIN, witness\_so\_far()**

**state.update(y); record\_vif(s, w)**

**return state, full\_witness()**

**I. KPIs (for APOE)**

* **Determinism rate (trace match under replay).**
* **Gate pass-through (per role) and true-positive abstentions at .**
* **Budget adherence (tokens/time/cost), cost per accepted artifact.**
* **Repair efficacy (first-repair success), rewrite frequency.**
* **Lineage completeness (VIF coverage).**

**J. Why It Matters**

**APOE converts “ask the model” into compile-and-run a program over memory, tools, and policies. The result is crisper answers, lower variance, transparent cost, and replayable trust—the necessary substrate for safe, large-scale machine communication.**

**1.4 From Claims to Witness (VIF)**

**Problem**

**Outputs that look right but can’t show how they were made are un-auditable risks. In regulated or high-stakes workflows, an answer without lineage is indistinguishable from a rumor.**

**Principle (Witness Invariant)**

**A claim that cannot show its lineage is a rumor.**

**VIF (Verifiable Intelligence Framework) makes every artifact provable, replayable, and calibrated. It standardizes *what* must be recorded at each boundary and *how* it is queried, displayed, and audited.**

**A. What VIF Guarantees**

1. **Lineage: A machine-readable path from claim → evidence → sources/tools → snapshot.**
2. **Provenance integrity: Immutable IDs (content hashes), signed envelopes, versioned prompts/tools.**
3. **Uncertainty disclosure: Calibrated UQ vectors; abstention if risk ≥ κ.**
4. **Replayability: The ability to re-execute the exact trace on the exact data/model versions.**
5. **Exportability: Self-contained evidence packs (for audit/compliance).**

**B. Minimal Formalism**

**Let be the Shared Evidence Graph (SEG) at time .  
A claim node is valid iff there exists a path:**

**with a provenance tuple and uncertainty vector :**

**Calibration: expected calibration error over held-out checkpoints; otherwise claims in that band require HITL.**

**Replayability: There exists a function such that:**

**C. VIF Envelope (at every step boundary)**

**Each boundary (APOE step) must emit a VIF envelope:**

**{**

**"witness\_id": "w\_01F8MECHZX3TBDSZ7XRADM79XE",**

**"time": "2025-10-17T19:21:54Z",**

**"actor": { "role": "reasoner", "principal": "apoe@system" },**

**"claim": {**

**"id": "claim:summary:v3",**

**"type": "text/markdown",**

**"hash": "sha256-…",**

**"parents": ["claim:outline:v2"]**

**},**

**"context": {**

**"snapshot\_id": "snap:c1c9…",**

**"read\_packs": ["rp:docs:…", "rp:code:…"],**

**"inputs\_hash": "sha256-…"**

**},**

**"provenance": {**

**"model\_id": "gpt-5t-2025-10",**

**"weights\_hash": "sha256-…",**

**"prompt\_template\_id": "tmpl:summary@1.12.4",**

**"prompt\_args\_hash": "sha256-…",**

**"tools": [**

**{ "name": "rag.search", "version": "2.3.1", "args\_hash": "sha256-…" }**

**]**

**},**

**"uncertainty": {**

**"entropy": 1.27,**

**"ensemble\_spread": 0.11,**

**"calibration\_bucket": "B",**

**"ece\_at\_bucket": 0.024,**

**"kappa\_decision": "ACCEPT" // or ABSTAIN/ESCALATE**

**},**

**"citations": [**

**{ "source\_id": "doc:ISO27001:2022", "span": "L210-L245", "hash": "sha256-…" },**

**{ "source\_id": "kb:design:authz-v2", "section": "3.1", "hash": "sha256-…" }**

**],**

**"signatures": {**

**"witness\_sig": "ed25519:…",**

**"chain\_root": "merkle:…"**

**}**

**}**

**Notes**

* **All large payloads are content-addressed; the envelope stores hashes + IDs, not blobs.**
* **Prompt template and weights are versioned and hashed.**
* **Citations point into the SEG with byte/line spans (or structured selectors).**

**D. Operational Rules (binding)**

* **V-1 No envelope, no artifact. If a step fails to produce a valid VIF, its artifact is non-exportable and cannot advance the chain.**
* **V-2 Show uncertainty by default. UI must render confidence bands; low-calibration buckets are yellow/red-flagged.**
* **V-3 Replay or retract. Any contested claim must be replayed from its snapshot; if replay fails, mark claim RETRACTED and create a contradiction edge.**
* **V-4 Tamper-evidence. Envelopes are signed (ed25519) and rooted in a Merkle tree committed to the snapshot log.**
* **V-5 Synthetic labeling. All AI-generated content carries machine-readable labels (content authenticity metadata).**

**E. UQ Methods (practical menu)**

1. **Temperature sweep (T=0.2…1.0) with agreement rate → spread.**
2. **Dropout/seed ensembles → variance proxy.**
3. **Retrieval overlap (top-k overlap across reruns) → grounding confidence.**
4. **Calibrators: Platt/Isotonic on task-specific dev sets → ECE.**
5. **Hybrid risk score .**
6. **κ-gating: If or ECE > ε for the bucket, ABSTAIN or ESCALATE (HITL).**

**F. UI/UX for Trust**

* **Inline confidence chip (“Conf: B • ±11%”) with hover to expand how it was computed.**
* **One-click lineage panel: sources, tool calls, prompt template, model hash, snapshot id.**
* **Replay button (guarded): re-execute with frozen inputs; diff overlay of outputs.**
* **Citations as anchors: clicking a citation jumps to the SEG node/selector.**
* **Risk banners: red/yellow bands for out-of-calibration buckets; direct “Send to reviewer.”**

**G. Failure Modes & Mitigations**

| **Failure** | **Symptom** | **Guard** |
| --- | --- | --- |
| **Orphan claims** | **Artifact with no envelope** | **V-1 hard fail; quarantine branch** |
| **Prompt drift** | **Same template id, different text** | **Hash prompts; diff at gate** |
| **Tool-call blind spots** | **Hidden HTTP/FS access** | **Capability tokens; tool registry; log all side-effects** |
| **Calibration rot** | **Confidently wrong answers** | **Continuous ECE monitoring; recalibrate or raise κ** |
| **Provenance forgery** | **Edited envelopes** | **Sign & Merkle-root; verify on read/export** |

**H. KPIs (VIF)**

* **Lineage coverage (% artifacts with valid VIF).**
* **Replay fidelity (% deterministic replays within tolerance).**
* **Calibration health (ECE by bucket; drift alarms).**
* **Abstention quality (precision/recall of κ decisions).**
* **Export pass rate (evidence packs that validate end-to-end).**

**I. Micro-Example (research paragraph)**

* **Claim: “AES-256-GCM provides confidentiality and integrity with 128-bit tags.”**
* **VIF: cites NIST SP 800-38D (spans), model/weights/prompt hashes, UQ band B, κ=ACCEPT.**
* **Replay: re-run retrieval + summarization from same snapshot; identical citation set; textual diff = 0.**

**J. How VIF Fits the Whole**

* **With CMC/HHNI: VIF envelopes bind claims to atoms & indices at the moment of use.**
* **With APOE: every boundary emits a witness; gates consume UQ to accept/abstain.**
* **With SEG: VIFs are edges & nodes in the evidence graph; time-sliced queries return who-knew-what-when.**
* **With SDF-CVF: releases are witnessed decisions; if later contradicted, we amend—we don’t silently overwrite.**

**Bottom line: VIF turns answers into evidence—measurable, explainable, and safe to build upon.**

**1.5 From Drift to Atomic Evolution (SDF-CVF)**

**Problem**

**Software and knowledge systems rot when their artifacts evolve out of sync. A model fixes a bug in code but not in docs; a policy tag changes but logs don’t record the new constraint; a test is updated but the incident runbook isn’t. This artifact drift creates invisible failure surfaces that no prompt can paper over.**

**Principle (Atomic Evolution Invariant)**

**Code, docs, tags, and traces must evolve as one.  
(*SDF-CVF* names the quartet we bind together: Source/Docs/Facts ↔ Code/Verification/Forensics.)**

**We treat every change as an atomic, witnessed bundle that preserves parity across the quartet and is either accepted as a whole or quarantined as a whole.**

**A. Artifact Quartet & Parity**

**Artifacts (minimal):**

* **Code: source, configs, schemas, migrations.**
* **Docs: specs, ADRs, runbooks, user help, API docs.**
* **Tags: policies, data classifications, PII labels, license headers, ACLs.**
* **Traces: tests, telemetry contracts, evidence links (VIF), SEG lineage, incident notes.**

**Parity means: for any semantic change in one member, the impacted members reflect that change *in the same commit* and the system can prove it.**

**B. Formalization**

**Let be a proposed change-set. Define typed diffs:**

**Each commit produces gate outcomes:**

**A parity score aggregates sub-parities:**

**where each is a normalized semantic-diff alignment (1.0 = perfectly synchronized).**

**Acceptance rule.**

**Else QUARANTINE with an auto-fix plan or ROLLBACK to last green snapshot.**

**Reproducibility. Every accepted change is anchored by a snapshot id and a witness envelope (VIF) that ties diffs to models, prompts, tools, and tests used.**

**C. Gate Catalog (binding, extendable)**

| **Gate** | **What it checks** | **Minimal signals** |
| --- | --- | --- |
| **g\_tests** | **Functional/contract correctness** | **Unit/integration/e2e; golden IO; schema/ABI stability** |
| **g\_spec** | **Doc/spec ↔ code parity** | **Spec-to-code trace links; unresolved TODOs = FAIL** |
| **g\_tags** | **Policy/labels correctness** | **PII/PD/License tags; breaking policy deltas escalate** |
| **g\_trace** | **Evidence & observability** | **VIF completeness; SEG lineage; telemetry contracts** |
| **g\_risk** | **Risk & HITL** | **κ decision from UQ; risk score ≥ τ ⇒ HITL two-key** |

**Non-bypassable: If a gate can’t run, it returns ABSTAIN and blocks the merge.**

**D. Atomic Bundle Pipeline (δ → decision)**

**Pipeline steps (synchronous):**

1. **Normalize: parse/AST docs & code; canonicalize tags; extract trace selectors.**
2. **Diff: compute typed diffs and impact graph (which files/paths must change).**
3. **Link: validate cross-refs (spec anchors, code symbols, policy targets, test names).**
4. **Prove: run tests; build VIF witnesses; update SEG working set.**
5. **Score: compute parity , UQ-risk, and budget deltas.**
6. **Decide: ACCEPT / QUARANTINE / ROLLBACK; emit snapshot & witness.**

**Quarantine loop (asynchronous):**

* **Generate auto-fix plan (missing doc sections, tag migrations, test stubs).**
* **Open a remediation branch with suggested diffs and a check-list.**
* **If remediation exceeds SLA or raises risk, rollback to and attach contradiction edges in SEG (we amend, never silently overwrite).**

**E. Data Structures (minimal)**

**Atomic-Bundle Manifest (ABM)**

**{**

**"bundle\_id": "abm\_01HV…",**

**"parent\_snapshot": "snap\_01HU…",**

**"diffs": {**

**"code": {"files":[...], "ast\_hash":"sha256-…"},**

**"docs": {"anchors":[...], "spec\_map":[...]},**

**"tags": {"adds":[...], "removes":[...], "migrations":[...]},**

**"traces": {"tests":[...], "vif\_ids":[...], "seg\_edges":[...]}**

**},**

**"parity": {**

**"spec\_code": 0.93,**

**"tags\_code": 0.88,**

**"trace\_code": 0.95,**

**"P": 0.92, "theta": 0.9**

**},**

**"gates": {**

**"tests":"PASS",**

**"spec":"PASS",**

**"tags":"PASS",**

**"trace":"PASS",**

**"risk":"PASS"**

**},**

**"decision": "ACCEPT",**

**"snapshot\_id": "snap\_01HV…",**

**"witness\_id": "w\_01HV…",**

**"signatures": {"bundle\_sig":"ed25519:…"}**

**}**

**F. Algorithms (sketches)**

**Spec↔Code parity (AST-aware):**

* **Extract spec anchors (requirements IDs, invariants).**
* **Map anchors to code symbols (types/functions) via static analysis & comments.**
* **Require: each changed symbol resolves to ≥1 updated anchor; each changed anchor resolves to ≥1 changed symbol. Compute F1 as .**

**Tags↔Code parity:**

* **For each changed policy/tag, find binding sites (schemas, routes, collectors).**
* **Validate presence of enforcement & redaction; diff runbooks for incident steps.**

**Trace parity:**

* **Ensure every changed public surface has updated contracts (OpenAPI/ABI), tests, and telemetry selectors; VIF covers all public claims.**

**G. UI/UX (reviewer ergonomics)**

* **Quartet diff view (4 synchronized panes) with cross-highlighting.**
* **Parity chip (“P 0.92 • PASS”) expandable to sub-parities and missing links.**
* **Gate bar (5 lights) with drill-downs; any ABSTAIN is clickable to root cause.**
* **Auto-fix suggestions (apply-patch buttons) + HITL two-key banner if risk ≥ τ.**
* **One-click replay (frozen snapshot) and SEG lineage overlay.**

**H. Failure Modes & Mitigations**

| **Mode** | **Symptom** | **Mitigation** |
| --- | --- | --- |
| **Silent drift** | **Code changed, spec didn’t** | **Gate g\_spec blocks; auto-generate spec diff stubs** |
| **Policy holes** | **Tags updated, no enforcement** | **Gate g\_tags requires instrumented checks & runbook diffs** |
| **Ghost tests** | **Spec updated, tests unchanged** | **Trace parity fails; scaffold test stubs; block until green** |
| **Broken lineage** | **Artifact has no VIF** | **Gate g\_trace blocks; auto-witness on retry** |
| **Risk blindness** | **Confident but miscalibrated changes** | **κ-gate g\_risk → ABSTAIN/HITL; recalibrate UQ** |

**I. KPIs (Atomic Evolution)**

* **Parity median per merge, p95 ≥ θ.**
* **Change-failure rate (CFR) < 10% (DORA).**
* **MTTR for quarantines and rollbacks (target by tier).**
* **Lineage coverage 100% (all bundles witnessed).**
* **Spec/code lag (time from code change → spec/runbook change) ≈ 0 for accepted bundles.**

**J. Micro-Example**

***Change:* add “Export evidence pack” API.**

* **Code: new endpoint + signer.**
* **Docs: API section + example pack schema.**
* **Tags: “exportable:true”, “contains:provenance”, “PII:none”.**
* **Traces: new contract tests; VIF replay for sample pack; SEG node types verified.**

**Result: , all gates PASS → ACCEPT, snapshot minted, Merkle root updated.**

**K. Interfaces to the Rest**

* **CMC/HHNI: ABM is the write gateway; accepted bundles update atoms/indices and tombstone deprecated nodes.**
* **APOE/ACL: plan steps include gate{…} constructs; failure auto-routes to remediation chains.**
* **VIF/SEG: each bundle is a decision node linked to its claims, sources, and tools—time-sliced and exportable.**
* **Security & Compliance: tags drive guardrails and retention; export packs satisfy audit pull-requests.**

**Bottom line: SDF-CVF converts “did we change the code?” into “did we evolve the system?”—and proves the answer, every time.**

**1.6 From Documents to Evidence Graphs (SEG)**

**Problem**

**Documents, tickets, chats, PRs, dashboards, and code reviews live in silos. They describe reality but rarely prove it. Over time, links break, screenshots rot, and “source of truth” collapses into trust-by-habit. Reasoning atop such strata is brittle and unauditable.**

**Principle (Substrate Invariant, SEG)**

**Evidence is a graph.  
Every claim, source, derivation, tool-call, and decision is a typed node with signed, time-bounded edges. We query lineage, not folders.**

**A. What the SEG is (and isn’t)**

* **Is: a temporal, typed, content-addressed knowledge graph that binds the four pillars:**
  1. **Claims (propositions the system asserts),**
  2. **Sources (documents, datasets, code, runs),**
  3. **Derivations (transformations: prompts, tools, compilers, tests),**
  4. **Decisions (approvals, releases, merges).**
* **Is not: a general document store, wiki, or vector DB. Those are attached as sources; the SEG carries relationships and witness.**

**B. Core vocabulary (minimal types)**

* **Claim: {id, text, scope, metrics, uncertainty}**
* **Source: {id, uri|digest, modality, snapshot\_id}**
* **Derivation: {id, kind: ("LLMRun"|"Build"|"Test"|"Query"), inputs[], outputs[], tool, params, udf\_hash}**
* **Decision: {id, policy, approvers, outcome, rationale}**
* **Witness: {id, model\_id, weights\_hash, prompt\_id, tool\_invocations[], timestamp, signature}**
* **Edges (typed): supports, derivesFrom, contradicts, annotates, usesTool, witnessedBy**

**All nodes/edges carry validity intervals and content addresses (hashes) for immutability.**

**C. Temporal model & reversibility**

* **Event-sourced: new facts append; we never silently mutate. Corrections add contradicts or supersedes edges.**
* **Time-slicing: every query is evaluated “as of” a timestamp; the graph kernel masks edges whose excludes the slice.**
* **Tombstones: deprecations are explicit nodes/edges, not deletes—preserving audit trails.**

**D. Minimal JSON-LD schema (sketch)**

**{**

**"@context": {**

**"id": "@id",**

**"type": "@type",**

**"Claim": "https://aimos.ai/seg#Claim",**

**"Source": "https://aimos.ai/seg#Source",**

**"Derivation": "https://aimos.ai/seg#Derivation",**

**"Decision": "https://aimos.ai/seg#Decision",**

**"Witness": "https://aimos.ai/seg#Witness",**

**"supports": {"@id":"https://aimos.ai/seg#supports","@type":"@id"},**

**"derivesFrom": {"@id":"https://aimos.ai/seg#derivesFrom","@type":"@id"},**

**"contradicts": {"@id":"https://aimos.ai/seg#contradicts","@type":"@id"},**

**"witnessedBy": {"@id":"https://aimos.ai/seg#witnessedBy","@type":"@id"},**

**"validFrom": {"@id":"https://aimos.ai/seg#validFrom","@type":"xsd:dateTime"},**

**"validTo": {"@id":"https://aimos.ai/seg#validTo","@type":"xsd:dateTime"}**

**},**

**"id": "urn:seg:claim:sha256:7a…",**

**"type": "Claim",**

**"text": "Export packs include VIF and UQ bands.",**

**"supports": ["urn:seg:source:sha256:ab…", "urn:seg:deriv:sha256:cd…"],**

**"witnessedBy": "urn:seg:witness:sha256:ef…",**

**"validFrom": "2025-10-17T12:00:00Z"**

**}**

**Identifiers. Use urn:seg:<kind>:sha256:<digest> over canonicalized payloads.  
Attachments. Large artifacts (docs, models, logs) remain in object stores; SEG stores content digests + URIs.**

**E. Write path (how facts enter)**

1. **Produce artifact → hash → store in object store (content-addressed).**
2. **Mint nodes: Source for the artifact; Derivation for the process that created it; Witness for VIF (model, weights hash, prompt id, tool list, signature).**
3. **Assert claims with supports/derivesFrom/witnessedBy edges.**
4. **Sign & time-bound nodes/edges; publish to SEG log.**
5. **Materialize index views (by time, by project, by policy tag).**

**All writes funnel through CreateEvidence (the “single writer” analog for SEG).**

**F. Read path (how we answer “why?”)**

**Canonical queries:**

* **Lineage (backward): given a Claim , find all acyclic paths  
  within the time slice. Return the minimal hitting set of sources that cover the claim’s predicates.**
* **Impact (forward): given a Source , enumerate Decisions reachable via  
  .**
* **Contradiction detection: find pairs with a contradicts edge active in the slice; rank by affected Decisions.**
* **Replay recipe: from a Decision, collect the frontier of Derivations + Witnesses; emit a frozen replay manifest.**

**Interfaces: SPARQL/Gremlin-style, plus a high-level “LineageQL”:**

**SELECT lineage FROM Claim('urn:…') AS OF '2025-10-17'**

**WHERE depth <= 3 AND include = {Witness, Derivation}**

**G. Quality & safety properties**

* **Completeness (w.r.t plan): APOE plan execution must emit a Witness per step; empty or partial witnesses fail gates.**
* **Non-repudiation: nodes/edges are signed; any replay verifies content digests and signatures.**
* **Least disclosure: privacy-sensitive attributes are redacted-at-ingest with reversible redaction tokens stored under purpose-bound keys.**
* **Abstention hooks: if lineage depth < τ or calibration error > κ, render claims with yellow bands and route to HITL.**

**H. Export packs (regulatory, read-only)**

**A SEG Export Pack is a zipped, signed bundle with:**

* **Manifest: top-level Decision/Claim ids, time slice, policy scope.**
* **Graph slice: JSON-LD nodes/edges reachable within constraints.**
* **Artifacts: content-addressed pointers + signed digests (optionally inlined small files).**
* **Replay: deterministic recipe (container images, model ids, prompts, seeds).**
* **UQ & calibration report: κ/ECE bands, failure cases.**
* **Redaction map: what was hidden, under which legal basis.**

**Packs are immutable; re-issuing creates a new pack with derivesFrom the prior.**

**I. UI/UX for comprehension**

* **Subway-map lineage: Decision at right; sources at left; derivations in the middle; chips for Witness and UQ bands.**
* **Time scrubber: slide along the timeline to watch edges appear/disappear.**
* **Explain button: expands a minimal proof (fewest edges) for the selected claim.**
* **Risk lenses: toggle to highlight un-witnessed nodes, low-confidence edges, or policy-tagged sources (PII, license).**

**J. Operational commitments (design constraints)**

* **S-1 Time-slicing API: every read accepts asOf and returns only active edges.**
* **S-2 Contradictions as first-class: we add contradicts edges; we do not delete history.**
* **S-3 Export-ready: one-click packs with replay and UQ are mandatory for “High-Risk” scopes.**
* **S-4 Content addressing: all artifacts referenced by digest; mutable URLs are advisory only.**
* **S-5 Single writer: CreateEvidence is transactional; partial writes are rejected.**

**K. Micro-example (end-to-end)**

* **Claim: “Release v1.8 complies with PII policy and meets p95 latency < 120 ms.”**
* **Supports:**
  + **Source(OpenAPI spec@sha), Source(latency dashboard snapshot@sha), Source(DP policy doc@sha).**
  + **Derivation(LoadTest run@sha, tool=k6, params), Derivation(DP scanner report@sha).**
* **Witness: model+weights hash + prompt template for the summarizer used to generate the release note.**
* **Decision: “Approve release v1.8,” two-key approvers, rationale, outcome=APPROVED.**
* **Edges: supports(Claim←Source/Derivation), witnessedBy(Claim←Witness), derivesFrom(Decision←Claim).**
* **Time-slice: As of 2025-10-17T12:34Z, all edges valid; at 2025-10-18 a new Claim with contradicts appears when latency regresses—UI shows yellow band and blocks auto-rollout.**

**L. How SEG anchors the rest**

* **CMC/HHNI supplies atoms and indices; SEG links them into explanations.**
* **APOE/ACL ensures every step emits a witnessed derivation.**
* **VIF is the per-artifact envelope; SEG is the system-level tapestry.**
* **SDF-CVF uses SEG to verify parity (doc/spec/test/code) before accepting evolution.**

**Bottom line: Documents are narratives. Evidence graphs are guarantees.**

**Chapter 2 — System Axioms**

**2.1 Memory Invariant (CMC) — Formal statement & proofs**

**Formal statement (Memory Invariant, CMC)**

**An intelligent system must convert ephemeral context into structured, queryable, and *reversible* memory.  
Concretely, for any time-indexed context stream , there exists a canonical memory state**

**such that (i) determinism: is a deterministic functional of under fixed seeds/tools; (ii) reversibility: a left-inverse reconstructs from with bounded loss; (iii) monotonicity: for only by explicit deprecation/tombstones; (iv) query sufficiency: + are sufficient statistics for top- retrieval under the system’s scoring functional; (v) witnessability: all transitions are witnessed and time-sliced in the Shared Evidence Graph .**

**Construction (what contains)**

* **Atomization : a typed multiset of *atoms* (text span, code span, event, tool-call, file blob, embedding), each content-addressed.**
* **Indexing : HHNI indices (system→section→paragraph→sentence→word→sub-word), tags, embeddings, TPV (Tag Priority Vectors), dependency hashes.**
* **Snapshot : immutable, content-addressed pack (manifest + blocks), signed and versioned.**
* **Graph : time-sliced evidence edges among {claim, source, derivation, decision, witness}.**

**Retrieval score (normalized)  
Let (quality, indexing depth, dependency delta). Define**

**with post-hoc calibration (isotonic/Platt) so that .**

**Required properties (as obligations)**

1. **Single-writer determinism. All writes pass a transactional CreateMemory gate with fixed seeds and tool versions → identical inputs yield identical digests.**
2. **Snapshot-first visibility. No user-visible state without an id.**
3. **Two-stage reads. Hierarchical read (coarse HHNI → fine atoms) with dumbbell compression (front-load + tail-load essentials; middle summarized) for long contexts.**
4. **Time-sliced lineage. Every transformation emits a Witness node; queries accept asOf.**

**Theorems (with proof sketches)**

**Theorem 1 — Determinism & Idempotence**

**Claim. Under fixed toolchain (model ids, weights hashes, prompt templates, seeds) and a single-writer gate, are deterministic; re-applying to is idempotent and yields the same digest.**

**Sketch. Content addressing (SHA-256 over canonicalized payloads) + frozen tool metadata removes nondeterminism from hashing. The single-writer gate serializes concurrent writes; equal inputs ⇒ equal manifests ⇒ equal snapshot digest. Idempotence follows since re-ingest compares digests and short-circuits.**

**Theorem 2 — Reversibility with bounded distortion**

**Claim. There exists such that with distortion , where is a declared budget from (a) lossy compressions and (b) quantized embeddings.**

**Sketch. Snapshots pack raw atoms (lossless) and *summaries/embeddings* (possibly lossy). Rehydration is exact for atoms and structural indices; for compressed spans we restore summaries within quantizer error. is the supremum of declared lossy operators’ bounds (e.g., top- truncation, vector quantization). Loss budgets are logged in (witness payload).**

**Theorem 3 — Monotonicity under event sourcing**

**Claim. For , is a subgraph of modulo explicit tombstones/deprecations; history is never destroyed, only superseded.**

**Sketch. Writes are append-only; corrections add contradicts/supersedes edges. Time-slice queries mask edges outside . Thus, the visible graph can shrink in a slice, but the ledger grows monotonically.**

**Theorem 4 — Query sufficiency of HHNI for top-k**

**Claim. Given the HHNI index with multi-granular embeddings, tags, and dependency hashes, the two-stage read (coarse→fine) returns a top- set whose expected F-measure is within of an oracle full-scan search of atoms, for calibrated .**

**Sketch. HHNI constructs a cover of the atom space with hierarchical representatives (sections/sentences). Under standard assumptions (Lipschitz similarity in embedding space; bounded drift), a coarse ANN retrieval followed by local refinement approximates the oracle ranking. The deviation is empirically calibrated and tracked in RS-lift dashboards.**

**Theorem 5 — Dumbbell compression minimizes position-bias loss**

**Claim. For attention kernels with “lost-in-the-middle” bias (concave in the center), placing critical cues at the front and tail and summarizing the middle minimizes expected recall loss under a fixed token budget.**

**Sketch. Model position-weight as with maxima near boundaries. For a budget , selecting top-mass spans near the ends maximizes . This yields the “dumbbell” layout as the optimal packing strategy; middle spans contribute via summaries to preserve latent links.**

**Theorem 6 — RS calibration & selective abstention**

**Claim. There exists a monotone calibration such that is a well-calibrated probability; choosing with yields a bound on false-positive retrievals .**

**Sketch. Apply isotonic or Platt calibration on held-out gold sets to map raw scores to empirical probabilities. By Neyman-Pearson style thresholding on , one chooses giving target type-I error . UI and gates consume and .**

**Corollary — Safety by construction**

**Because every read/write is (i) snapshot-first, (ii) time-sliced, and (iii) witnessed into , *any* returned fact has a replay recipe; if or uncertainty , the system abstains or routes to HITL.**

**Algorithms (obligation-level)**

**Write pipeline (deterministic):  
INGEST → ATOMIZE → INDEX(HHNI) → SCORE(QS,IDS,DD) → SNAPSHOT(sign,digest) → GRAPH(upsert witnesses)**

**Read pipeline (two-stage):  
COARSE(HHNI nodes) → RE-RANK(local atoms) → DUMBBELL(pack) → UQ/ABSTAIN(κ) → RENDER + LINEAGE**

**Design constraints (bound to CMC)**

* **C-1 Single writer. One transactional CreateMemory endpoint; conflicts resolved before snapshot.**
* **C-2 Snapshot-first. No state leaks without snapshot\_id.**
* **C-3 Two-stage read. Mandatory hierarchical search + dumbbell packing.**
* **C-4 Calibrated RS. Score calibration fitted, tracked, and versioned.**
* **C-5 Loss budgets. Each lossy operator declares a bound; rehydration reports .**
* **C-6 Witness everywhere. Each transition emits a VIF payload into SEG.**

**Worked micro-example (end-to-end)**

* **Context: user prompt + code diff + log excerpt.**
* **Atomize: 7 atoms (2 text spans, 3 code spans, 2 events).**
* **Index: sentence/word embeddings; tags #auth, #jwt; dependency hash binds diff↔test.**
* **Snapshot: manifest digest sha256:… signed; lossy operator: middle-summary (ε=0.03).**
* **Graph: Derivation(LLMRun) witnessed by {model,weights\_hash,prompt\_id,tools}; Claim “JWT added without PII leak” supported by test result + static scan.**
* **Read: query auth token → HHNI retrieves section→sentences; local re-rank surfaces the diff atom; dumbbell packs front (requirements) + tail (tests); UI shows , green band; lineage expands to show witnesses.**

**What this buys the whole system**

* **For APOE: plans operate over *stable* memory, not drifting chat; boundary gates consume calibrated RS.**
* **For VIF/SEG: every memory fact has a witness and replay.**
* **For SDF-CVF: parity checks join code/docs/tags/traces at commit time.**
* **For users: answers that can be *proven* and *replayed*, or gracefully *abstained*.**

**Chapter 2 — System Axioms**

**2.2 Orchestration Invariant (APOE) — Formal statement & proofs**

**Formal statement (Orchestration Invariant, APOE)**

**Reasoning must be compiled, not improvised.  
For any goal and memory state , there exists a plan that is a finite, typed, budgeted DAG in the AIMOS Chain Language (ACL) such that execution of on yields a witnessed trace with:  
(i) determinism under fixed context (up to declared stochastic bands),  
(ii) compositionality (plans compose without changing meaning),  
(iii) boundary gates (every step emits provenance + UQ), and  
(iv) safe liveness (terminates in success or abstains at with rollback).**

**Model (objects and semantics)**

**Plan as a typed DAG**

**A plan where:**

* **: steps; : acyclic edges (data/control deps).**
* **gives each node a step schema:**

**with pre/postconditions, gate set (tests, policy, provenance), and effect type (pure, tool, external).**

**Execution environment**

**where memory (from §2.1), tool registry (version-pinned), budget vector (tokens, time, calls, cost), and seed .**

**Operational semantics (small-step)**

**We write configurations with scheduler state .  
A reduction executes any ready node (all preds satisfied), producing:**

**where updates budgets and only through snapshotted writes, and appends a witness record (model id, weights hash, prompt id, inputs digests, outputs digests, UQ vector , gate outcomes). If any gate in fails or crosses abstention , the runtime executes rollback of side-effects and emits an abstain event.**

**Theorems (with proof sketches)**

**Thm 1 — Determinism under fixed context**

**Claim. With fixed , , , ACL, and seed , the witnessed trace is unique up to declared stochastic tolerance bands; retries are idempotent.**

**Sketch. Each step consumes content-addressed inputs and pinned tool versions. Randomness is derived from and node id → repeated runs hash-equivalent. Tolerance bands (e.g., beam sampling) are logged in ; within-band variation is allowed but artifacts’ canonical forms (digests) remain equal or are rejected by gates.**

**Thm 2 — Compositionality (plan algebra)**

**Claim. ACL plans form a strict symmetric monoidal category:**

* **objects: typed interfaces;**
* **morphisms: plans;**
* **composition : interface-typed gluing;**
* **tensor : parallel composition;  
  with identities and associativity holding, so refactoring does not change denotation.**

**Sketch. Interfaces are product types of inputs/outputs. Composition is node/edge union with interface alignment and topological merge; tensor is disjoint union. Because gates are attached to node boundaries, their satisfaction composes (induction on topo order). This establishes modularity and permits verified refactors.**

**Thm 3 — Gate soundness (safety)**

**Claim. If every step satisfies its preconditions and passes its gate set , then the whole plan satisfies the global policy (tests, safety, provenance minima).**

**Sketch. Gates encode obligations from as local Hoare triples . By DAG induction and monotone resource flow, local postconditions compose to the global postcondition. Violations cannot leak because boundary gates block propagation.**

**Thm 4 — Safe liveness (termination or abstention)**

**Claim. Under fair scheduling, finite , and no cyclic deps, execution terminates in success or abstains at some boundary where safe bands or budgets exhaust.**

**Sketch. Define a well-founded measure as (remaining nodes, remaining budgets). Each reduction strictly decreases or produces abstention; thus no infinite descent.**

**Thm 5 — Replayability**

**Claim. Given and the snapshot id(s) referenced by its witnesses, replaying the plan yields artifacts that are digest-equivalent or a flagged divergence.**

**Sketch. Witnesses pin all degrees of freedom (tool versions, seeds, prompts, inputs). Re-execution reproduces digests; any drift indicates external mutation and is surfaced.**

**Thm 6 — Refinement monotonicity (Evaluator-Optimizer)**

**Claim. Let be a refinement that adds a critic step and a repair step guarded by . Then expected risk is non-increasing and expected RS is non-decreasing over accepted outputs.**

**Sketch. The critic rejects or downgrades low-confidence outputs; repair improves or abstains. By selection, accepted set’s mean risk decreases; calibration verified on held-out traces.**

**Algorithms (obligation-level)**

**APOE compilation pipeline**

**INTENT\_PARSE → DECOMPOSE (roles) → ALLOCATE (budgets) → INSERT (gates) → ACL\_EMIT → SCHEDULE (topo + budgets) → EXECUTE (parallel safe) → COLLECT (witnesses) → REPAIR/ABSTAIN → FINALIZE (snapshot links)**

* **Decompose into roles: *planner, retriever, reasoner, verifier, builder, critic, operator, witness*.**
* **Budgets: assign token/time/tool/cost caps per node.**
* **Gates: attach tests, safety/PII scans, UQ thresholds, provenance minima.**
* **Scheduler: (a) topological order, (b) respect budget/resource locks, (c) width-bounded parallelism.**
* **Retry policy: bounded exponential backoff; idempotent re-runs; seed policy pinned.**

**Boundary contract (every step)**

**Inputs (digests) • Tool spec (model id, weights hash, prompt template id, tool args) • Budgets consumed • Artifacts (digests) • UQ vector • Gate outcomes • SEG pointer**

**Design constraints (bind to APOE)**

* **O-1 Budgets explicit — every step carries token/time/tool/cost limits.**
* **O-2 Typed roles — steps declare role; policies attach by role.**
* **O-3 Boundary gates — per-step tests, safety, provenance, and UQ must pass.**
* **O-4 Replay semantics — traces must re-run on frozen snapshots.**
* **O-5 Abstention first — if crosses or RS falls below threshold, stop/route to HITL.**
* **O-6 Side-effect isolation — tool calls run in sandboxes; writes are snapshot-first.**
* **O-7 Parallel joins are safe — only join on content-addressed artifacts; race-free merges.**
* **O-8 Version pinning — models, weights, prompts, tools are versioned; drift → error.**
* **O-9 Metrics everywhere — per-node budgets, latency, UQ, RS; per-plan .**

**Micro-example (4-node plan)**

**Goal: “Summarize policy doc, extract PII rules, and propose safe redaction filter.”**

1. **Retriever (budget: 3 tool calls, 4k tokens)  
   → artifacts: top-k sections + SEG citations; (low uncertainty).**
2. **Reasoner (budget: 8k tokens)  
   → artifact: concise policy summary; UQ , passes .**
3. **Verifier (tests + policy gate)  
   → checks summary against sources (overlap ≥ 0.85), PII taxonomy matched.**
4. **Builder (codegen)  
   → emits redaction function; gate runs on synthetic PII corpus; passes parity tests.**

**Witnesses for each step anchor model/prompt hashes and inputs/outputs. The plan succeeds; replay reproduces digests. If Verifier failed, APOE would abstain or route to HITL.**

**What this buys the whole system**

* **Reliability — compiled chains are auditable, replayable, and resource-bounded.**
* **Modularity — compositional plans enable reuse and safe refactors.**
* **Safety — gates and make abstention a *first-class* outcome.**
* **Velocity — parallel sections run safely; retries are idempotent.**
* **Governance — every boundary emits VIF metadata into SEG for regulators and SREs.**

***Compile the plan. Execute the plan. Show the witness. Or abstain.***

**Chapter 2 — System Axioms**

**2.3 Witness Invariant (VIF) — Trust calculus & auditability**

**Formal statement (Witness Invariant, VIF)**

**A claim that cannot show its lineage is a rumor.  
For any artifact (text, code, decision) produced by APOE on memory state , there must exist a witness such that:**

1. **encodes a verifiable lineage from sources → retrievals → transforms → outputs in the Shared Evidence Graph (SEG),**
2. **carries a calibrated uncertainty vector , and**
3. **is replayable on the referenced snapshot(s); replay either reproduces digest-equivalent artifacts or yields a flagged divergence.**

**1) What a Witness is (and isn’t)**

**A witness is a machine-readable, cryptographically addressable record that binds who/what/with-what created which artifact, from which inputs, under which budgets, and with what confidence.  
It is *not* a narrative explanation; it is the evidence bundle that makes explanations auditable.**

**Witness tuple (minimal):**

* **id (content address, e.g., BLAKE3 of canonicalized witness JSON)**
* **time (wall clock, monotonic seq) • actor (role, step id)**
* **model (name, weights\_hash) • prompt\_template\_hash**
* **tools (tool ids + versions + args digests)**
* **inputs (SEG node ids / snapshot ids + content digests)**
* **budgets (tokens, time, cost) used vs. allowed**
* **outputs (artifact digests, MIME, size)**
* **UQ vector**
* **gates (test results, policy checks, redactions)**
* **signatures (optional: step signer, system signer)**

**Every step in a plan emits one witness; the artifact’s top-level witness references all interior witnesses via the SEG.**

**2) Trust calculus**

**We compute a trust score that the system displays and gates upon. It is *not* truth; it is operational confidence grounded in evidence.**

* **S (Support): coverage and strength of citations  
  over cited spans vs. claims.**
* **C (Consistency): absence of contradictions in SEG slice  
  .**
* **R (Replayability): digest match under replay  
  if all replays match; else or proportion matched.**
* **F (Freshness): temporal validity  
  (per source, aggregated).**
* **P (Provenance quality): source/auth chain, signatures, licenses  
  normalized score by policy (e.g., peer-reviewed > blog).**
* **L (Calibration): inverse expected calibration error  
  .**

**Abstention rule. Emit -abstain if or any mandatory component (e.g., ) fails.**

**Monotonicity. Adding *consistent* evidence cannot decrease ; adding contradictory evidence decreases and so .**

**3) Auditability: what we log, how we verify**

**3.1 Lineage in the SEG (time-sliced)**

* **Nodes: source, snippet, retrieval, tool-call, transform, claim, decision.**
* **Edges: supports, derives, paraphrases, contradicts, uses-tool, tested-by.**
* **Each edge carries validity; queries accept an as-of time.**

**3.2 Replay protocol (boundary contract)**

1. **Freeze referenced snapshot ids (memory, tools, prompts, seeds).**
2. **Re-execute the plan slice producing with the same budgets.**
3. **Canonicalize outputs → compute digests → compare.**
4. **Record replay\_witness with and diffs.**

**3.3 Calibration protocol**

* **Entropy-based : normalized token-level entropy / variance.**
* **Agreement-based : ensemble or temperature-sweep disagreement.**
* **Calibration : bin predictions by stated confidence vs. empirical correctness on gold or replay checks.**
* **Store reliability diagrams and ECE in witness.**

**4) VIF schema (concrete, minimal JSON)**

**{**

**"witness\_id": "b3:6a1f…",**

**"time": "2025-10-17T21:04:12Z",**

**"actor": {"role":"verifier","plan\_step":"verify-03"},**

**"model": {"name":"gpt-X","weights\_hash":"sha256:…"},**

**"prompt\_template\_hash": "sha256:…",**

**"tools": [{"id":"web.search","version":"2.4","args\_hash":"b3:…"}],**

**"inputs": [{"seg\_id":"src:doc:123#p4","digest":"b3:…"}],**

**"budgets": {"tokens":{"used":2389,"cap":4000},"time\_ms":{"used":820,"cap":3000},"cost\_cents":1.2},**

**"outputs": [{"artifact":"claim:policy-summary","mime":"text/markdown","digest":"b3:…"}],**

**"uncertainty": {"entropy":0.17, "agreement":0.08, "ece":0.04},**

**"gates": [{"name":"source\_overlap>=0.85","pass":true},{"name":"pii\_policy","pass":true}],**

**"signatures": [{"by":"system","alg":"ed25519","sig":"…"}],**

**"seg\_links": [{"edge":"derives","from":"retrieval:77","to":"claim:policy-summary"}]**

**}**

**5) Gates & policies (binding)**

* **W-1 Machine-readable lineage. No artifact may be emitted without at least one witness that forms a connected path to the claims it asserts.**
* **W-2 Confidence hygiene. If or , display risk banners and route to verifier or HITL.**
* **W-3 Replayability. High-risk classes require before merge; otherwise quarantine.**
* **W-4 Redaction. VIF pipelines apply PII/license redaction before logging sources; SEG stores redacted derivatives + reversible escrow hashes.**

**6) Properties (sketches)**

* **Replay Lemma. If tools, models, seeds, and inputs are content-addressed and pinned, then replay is idempotent modulo external nondeterminism; divergence surfaces as a witnessed diff.**
* **Conservativeness. In absence of lineage, set (rumor). Partial lineage yields bounded by the weakest mandatory component.**
* **Anti-inflation. Appending unsupported citations or non-verifiable tool calls cannot raise ; and cap the score.**

**7) Algorithms (obligation-level)**

**Witness-assembly (per step):  
collect\_inputs → hash\_prompts → pin\_model/tool → run → hash\_outputs → compute\_u → run\_gates → emit\_witness → attach\_to\_SEG**

**Trust evaluation (per artifact):  
gather\_subgraph → compute S,C,F,P,L,R → aggregate T → compare to κ\_T → {emit|abstain}**

**Contradiction scan (online):  
for new claim c: search SEG for contradicts(c, t\_asof) → if any active → lower C; require verifier**

**8) UI/UX (operator & end-user)**

* **Confidence band (green/amber/red) with and breakdown hovercard (S/C/R/F/P/L).**
* **“Show lineage” opens the SEG slice: sources with highlighted spans; tool and prompt cards; replays.**
* **“Recheck now” button triggers replay/calibration with current policies (read-only budgets).**
* **Copy as JSON-LD provides the machine-readable witness for export.**

**9) Failure modes & mitigations**

* **Orphan claims (no sources) → , block.**
* **Stale sources (low ) → warn, route to retriever refresh.**
* **Calibration drift (rising ECE) → auto-schedule recalibration jobs; tighten .**
* **Provenance forgery → verify signatures; mismatches quarantine the plan.**

**10) Worked micro-example**

**Claim. “Section §4.2 of Policy-X requires redacting SSNs before storage.”**

* **Support . nDCG@5=0.92; coverage 0.88 → .**
* **Consistency . No contradictions in policy history slice → 1.00.**
* **Replay . Deterministic match → 1.00.**
* **Freshness . Last updated 30 days, → .**
* **Provenance . Signed PDF + checksum in registry → 0.95.**
* **Calibration . ECE=0.05 → .**
* **Aggregate (weights e.g., ) → .  
  Since , APOE may proceed; otherwise it would abstain or escalate to HITL.**

**What VIF gives the system**

* **Operational trust that’s computed, not asserted.**
* **Regulatory posture: instant export of machine-readable lineage.**
* **Engineering leverage: replayable bugs, measurable calibration, searchable contradictions.**
* **Cultural norm: if you can’t show the witness, you don’t ship.**

**Chapter 2 — System Axioms**

**2.4 Atomic Evolution (SDF-CVF) — Compositionality & parity**

**Formal statement (SDF-CVF)**

**Code, docs, tags, and traces evolve as one.  
Let a system state at time be the quartet  
where =code, =documentation/specs, =facts/tags/metadata, =VIF/SEG traces.  
A change is valid iff there exists a single atomic commit that transforms each projection coherently, and all parity gates pass:**

**After apply, preserves referential, behavioral, and evidential consistency.**

**1) Compositional model of change**

**Unit of change.  
A commit is a delta-quartet**

**with each component a typed edit script on its domain (AST diff, doc diff, tag set diff, SEG subgraph diff).**

**Functorial parity.  
Define projection functors to a common semantic space (the spec/behavior model):**

* **(behavior extracted from code/tests)**
* **(requirements/spec clauses)**
* **(policy labels, data contracts)**
* **(claims/evidence implied by traces)**

**Parity constraint.  
After apply, projections agree within a bounded slack :**

**where is a domain-specific semantic distance (e.g., spec clause coverage vs. test behavior; policy label coverage vs. PII filter paths). for high-risk classes.**

**Composition.  
For independent commits that touch disjoint semantic slices, and parity is preserved (commutativity). If overlaps exist, a merge gate checks conflicts in .**

**2) Gate catalog (binding)**

* **— all tests (unit, property, contract) pass; new behavior has tests; coverage thresholds met.**
* **— updated requirements/specs align with behavior: .**
* **— labels/policies/types (e.g., PII, license, SLAs) are consistent with code/data flows: no unguarded sinks, correct encoders/DP transforms.**
* **— VIF/SEG updated: each outward-facing claim has lineage; replay on pinned snapshots matches; export pack builds.**
* **— risk score ≤ threshold or has two-key HITL approval; redactions applied.**

**Failure policy.  
If any gate fails → Auto-fix (localized synthesis/refactor) or Quarantine (branch isolates with rationale & witness), never a silent merge.**

**3) Parity scoring & slack**

**Define a parity score :**

**with . Policy: require (e.g., 0.95 for critical paths, 0.85 for experimental branches). Slack is implied by .**

**4) Proof sketches (intuition)**

**Soundness (gate preservation).  
If all gates hold at , and passes gates at review time on snapshots referenced by its VIF, then applying yields a state where projections remain within ; any downstream divergence re-surfacing under replay raises → quarantine (no silent violation).  
*Sketch:* gates enforce preconditions; witnesses pin artifacts; replay is idempotent under pins (cf. VIF). Therefore deviations are detectable and blocked.**

**Liveness (no deadlock of safe change).  
For any desired spec-conforming behavior reachable by finite edits, there exists a finite sequence of atomic commits and auto-fixes that increases monotonically until .  
*Sketch:* each failed gate yields a finite set of obligations (missing tests, tags, traces) that APOE can synthesize; obligations strictly reduce a deficit measure.**

**Compositionality.  
If touch disjoint slices of , then parity scores combine as a weighted average and cannot violate if each individually meets it.  
*Sketch:* distances add over disjoint supports; normalization preserves bounds.**

**5) Algorithms (practical)**

**Pre-merge parity checker.**

1. **Build projections on the PR snapshot.**
2. **Compute & component distances; emit diffs (missing tests/specs/tags/traces).**
3. **If : generate obligation graph → feed APOE auto-fix chains (e.g., test synthesis, doc patch, SEG witness scaffolding).**
4. **Re-evaluate → pass gates or quarantine.**

**Auto-fix strategies.**

* **Spec→Test parity: synthesize tests from normative language; bind to code via symbol grounding.**
* **Tag parity: static/data-flow analysis inserts sanitizers/DP ops; updates policy labels.**
* **Trace parity: draft VIF records; attempt replay; annotate deltas.**

**Quarantine & rollback.**

* **On persistent failure, branch enters quarantine with a frozen witness bundle and a rollback plan (last good snapshot + revert diffs); merge unblocks via two-key HITL only.**

**6) Worked micro-examples**

**A) Spec tightened (PII retention 90→30 days).**

* **: policy text.**
* **Obligations: add retention test, update data job TTL, rotate tags (#PII-retention:30d), regenerate SEG lineage for policy-driven changes.**
* **Gates: tests & tags & trace. Auto-fix proposes TTL patch + test; reviewer approves; merge.**

**B) Code refactor alters API error model.**

* **: handler signatures.**
* **Obligations: doc update (error table), contract tests, VIF witness for change note.**
* **If doc unchanged → fails → spec patch synthesized; parity restored.**

**7) Metrics & SLOs (SDF-CVF)**

* **Parity score (p50/p90 by service & risk class).**
* **Obligation closure time (mean time from deficit → auto-fix or HITL).**
* **Change Failure Rate (CFR) and MTTR (DORA).**
* **Quarantine rate and rollback frequency.**
* **Replay fidelity for traces (VIF ratio).**

**8) Operator UX (minimum viable)**

* **Parity bar on PRs with breakdown (C↔D, C↔F, C↔V).**
* **Obligation list with “Auto-fix” buttons (preview diff + tests).**
* **Risk banner (class, thresholds, two-key status).**
* **One-click rollback (to last green snapshot) with witness link.**

**9) Failure modes & mitigations**

* **Spec-only edits (no tests/code): blocked by & ; auto-synthesize tests.**
* **Tag drift (labels stale): nightly parity scan opens DEPP tasks; auto-patch or Jira ticket.**
* **Trace rot (replay breaks under env drift): environment pins & simulators; if external API changed, attach contradiction edge in SEG and route to policy.**
* **Over-automation (fixes that mask intent): HITL thresholding by risk class; diff semantics highlight behavior changes beyond tolerance.**

**10) Contract (what merges must promise)**

1. **Single atomic delta-quartet with pinned witnesses.**
2. **Parity score ≥ and all gates pass (or two-key override).**
3. **Rollback plan materialized.**
4. **SEG updated with claim→evidence lineage for the change.**

**Bottom line: SDF-CVF turns “keeping everything in sync” from an aspiration into a merge contract. If it can’t pass together, it doesn’t ship.**

**Chapter 2 — System Axioms**

**2.5 Substrate Invariant (SEG) — Graph-theoretic underpinnings**

**Statement (SEG)**

**Evidence is a graph.  
All claims, sources, tool invocations, derivations, and decisions must be represented as a time-aware, typed, signed graph. Deletion is replaced by deprecation; contradiction is first-class; and every query is as-of a time slice.**

**1) Formal model**

**We model the Shared Evidence Graph as a bitemporal, typed, edge-labeled multigraph with signatures:**

* **Vertices : typed nodes .**
* **Edges with relation types**

**Multi-edges allowed.**

* **Validity : valid-time interval (when the statement holds in the domain).  
  Transaction-time (when the record exists in the store). Together they give bitemporal semantics.**
* **Signatures : node schemas (minimal shape: id, type, content hash, content addr, author keys, UQ vector if claim).**
* **Attributes : edge payload (provenance hash, tool id, prompts, weights, confidence).**
* **Signatures : cryptographic signatures for nodes/edges (model key, tool key, human approver).**

**Time slices.**

* **As-of snapshot at time : s.t. and .**
* **Audit snapshot at processing time : s.t. (even if validity is past/future).**

**Causality discipline.  
Edges of type derives form a time-respecting DAG on any as-of slice: if , then . Cycles may exist across other relations but are disallowed in derives to preserve replayability.**

**2) Normal forms & constraints**

* **NF-1 (Identity): Every vertex has a content-address (hash of canonicalized payload).**
* **NF-2 (No deletion): Retractions are encoded as**
  + **set on validity, and/or**
  + **add explicit contradicts edges with a reason/code.**
* **NF-3 (Witness completeness): A Claim is publishable iff there exists a supported derivation path to at least one Source on .**
* **NF-4 (Minimality): Derivation paths should be irredundant (no subpath removal preserves reachability).**
* **NF-5 (Signatures): Any edge that changes a public artifact (e.g., Decision) must carry from the responsible principal.**

**Design constraints (expanded):**

* **S-1 Time-slicing API: every read includes and/or . Defaults must be explicit.**
* **S-2 Contradictions: represent as edges, never destructive edits; UIs render red stripes across affected claims.**
* **S-3 Export packs: deterministic projection of a bounded subgraph to a portable bundle (JSON-LD + binary blobs + signature manifest).**
* **S-4 Replay pins: a Snapshot node freezes model id, weights hash, prompt template id, tool versions; plans/claims reference snapshots by id.**

**3) Trust calculus on a typed, temporal graph**

**We define a bounded, composable support score for a claim at time :**

* **: set of valid (at ) paths from to any .**
* **: edge weight (evidence strength, calibration-adjusted; decays with age: ).**
* **: node reliability (publisher reputation, model calibration for ToolCall, human approver class).**
* **: hop attenuation; is path length.**
* **Contradictions apply as penalties:**

**where are claims contradicting at , and tunes penalty strength.**

**Properties.**

* **Monotonicity under addition: adding a new supporting path cannot decrease (penalties only from explicit contradicts).**
* **Locality: Updating an edge rescales only scores reachable from that edge (facilitates incremental recompute).**

**4) Query algebra (time-aware)**

**We define an algebra over typed, temporal graphs; let denote evaluation at .**

* **Slice: → .**
* **Neighborhood: → nodes within hops over edges of type .**
* **Lineage (backward): .**
* **Impact (forward): .**
* **Conflict set: .**
* **Best witness: .**
* **Explain: returns best witness path plus UQ vectors and signatures.**

**All operators preserve bitemporality: they operate on with defaulting to “now” unless specified.**

**5) Algorithms & indexing**

**Storage choice.**

* **Primary: property graph (adjacency index + columnar attributes), augmented with interval trees on and .**
* **Projection: RDF/JSON-LD for interoperability (Chapter 15 covers shapes).**

**Indexes.**

* **: → edge ids (fan-out).**
* **: interval index on (valid-time) and (txn-time).**
* **: per-node top-k witness cache (path sketches) per risk class.**
* **: signature manifest keyed by content-address.**

**Core procedures.**

* **As-of traversal. Two filters on each expansion:**
  1. **edge/type filter; 2) interval containment .  
     Complexity: worst-case; typical is bounded by degree & hop cap.**
* **Best-witness search.  
  Use label-setting variant of Dijkstra with edge cost ; prune with time-aware admissible heuristic (e.g., upper bound on remaining support).  
  Complexity: per query; amortize via .**
* **Incremental recompute.  
  Maintain reverse dependency lists; when updates, recompute for reachable using delta-propagation until change < .**
* **Contradiction accounting.  
  Maintain bitsets per claim of active contradictors; update penalty product in where is number of contradicting claims on slice.**

**6) Operations (append-only discipline)**

**All writes are append-only; mutations create new versions with new transaction intervals.**

* **assert\_claim(payload, snapshot\_id, uq) → with .**
* **link\_derives(u,v,meta,τ) → add with .**
* **link\_supports(u,v,meta,τ) → add supporting edge.**
* **link\_contradicts(u,v,reason,τ) → explicit conflict.**
* **deprecate(edge\_id, t\_close) → set validity ; record cause.**
* **approve(decision, approver) → with human signature.**
* **export\_pack(root, t) → bounded subgraph with signatures + hashes.**

**Idempotence.  
Repeated assertions with identical payloads collapse by content-address keys.**

**7) Soundness & monotonicity sketches**

* **Replay soundness.  
  Given a claim accepted at time with witness path , replay on snapshot pins succeeds iff all edges in are valid at and all tool calls are either pure or re-executed in simulation mode with pinned inputs. Because derivations are a DAG and all nodes are content-addressed, path reconstruction is deterministic.**
* **Monotonicity under extensions.  
  Adding new edges or nodes never invalidates prior accepted claims as-of their acceptance time ; it may create contradictions that alter for . This ensures historic auditability.**
* **No hidden deletion.  
  Because tracks transaction intervals, redaction or retention expiration is observable in audit mode even when validity has ended.**

**8) Failure modes & mitigations**

* **Orphan claims.  
  *Symptom*: .  
  *Mitigation*: block publication; open DEPP task for witness acquisition.**
* **Temporal leaks.  
  *Symptom*: edges valid outside the parent snapshot window.  
  *Mitigation*: schema rule—any edge referencing a Snapshot must satisfy .**
* **Contradiction storms.  
  *Symptom*: cascades of low-quality contradict edges.  
  *Mitigation*: require from qualified principals; throttle by reputation; show penalty but gate acceptance.**
* **Cycle in derives.  
  *Symptom*: replay non-termination.  
  *Mitigation*: enforce DAG on derives at write time (topological check on slice).**
* **Signature drift.  
  *Symptom*: unverifiable content after retention rotation.  
  *Mitigation*: store key fingerprints in export packs; rotate with cross-signed manifests.**

**9) Minimal worked example (as-of )**

* **(doc hash H\_A)**
* **(extracted paragraph) with**
* **(RAG retrieval; model M, weights W, prompt P; UQ u)**
* **(“Policy X applies to dataset Y”)**
* **Edges: ,**
* **Later, (“Policy X does not apply to Y”) with (reason: updated regulation) and starts at .**

**Queries:**

* **Explain @ → returns path + toolcall ; .**
* **Explain @ → same path but penalized by contradiction: . UI shows confidence drop and link to .**

**10) Operator & UX commitments**

* **As-of switcher on every evidence view.**
* **Best witness panel (path, UQ, signatures).**
* **Conflict banner with navigation to contradicting claims.**
* **Export button (pack includes JSON-LD subgraph, blobs, signature manifest, checksums).**
* **Pin / Replay controls bound to Snapshot nodes.**

**11) What SEG guarantees the rest of the system**

* **CMC can persist atoms with lineage preserved forever as-of.**
* **APOE can compile to witnesses, not prose, and re-execute plans on pins.**
* **VIF has a canonical place to live and propagate trust.**
* **SDF-CVF merges cannot proceed unless the graph proves parity.**

**Bottom line: By making evidence a time-aware, signed, typed graph—never a log of strings—we earn replayability, auditability, and composable trust. This is the substrate that lets everything else (memory, orchestration, evolution) remain coherent over time.**

**Chapter 2 — System Axioms**

**2.6 Axioms → Design Constraints (traceability table)**

**Purpose. Make every invariant operational. Each line binds: Axiom → Non-negotiable constraint → Design manifestation → Verification → Runtime monitor → Escalation.  
IDs are stable; they appear in code comments, ACL plans, SEG nodes, and dashboards.**

**2.6.1 Master Traceability Table**

| **Axiom** | **Constraint (ID)** | **Design Manifestation** | **Verification (Build-time)** | **Runtime Monitor** | **Escalation** |
| --- | --- | --- | --- | --- | --- |
| **CMC Memory must persist** | **C-1 Single-writer** | **All writes go through CreateMemory flow; one queue/txn domain** | **Unit: reject direct DB writes; Policy test: block non-flow service accounts** | **Alert on any write bypass (DB audit log rule)** | **Quarantine shard; rotate keys; RCA in SEG** |
|  | **C-2 Snapshot-first** | **No artifact exposed without snapshot\_id** | **Contract test on APIs: 4xx if missing snapshot** | **Ratio: responses\_without\_snapshot\_id == 0** | **Freeze endpoint; hotfix gate** |
|  | **C-3 Two-stage read** | **HHNI coarse → atoms (dumbbell compression)** | **Golden query diff vs. flat read** | **p50/p95 read-latency + precision@k vs. baseline** | **Re-index; raise read budget temporarily** |
| **APOE Reasoning is compiled** | **O-1 Budgets explicit** | **ACL requires tokens/time/tools per step** | **Static ACL linter (EBNF) rejects missing budgets** | **Budget burn chart per chain; overflow alarms** | **Auto-abstain; route to operator** |
|  | **O-2 Typed roles** | **Planner/Retriever/Reasoner/Verifier/Builder/Critic/Operator/Witness** | **Plan compile: role coverage & segregation checks** | **Role skew monitor (e.g., verifier invoked %)** | **Trigger “thin-verifier” incident** |
|  | **O-3 Boundary gates** | **Each step emits UQ, provenance, diff** | **Contract test: step must return {artifact,uq,prov}** | **% steps with UQ & prov == 100%** | **Block chain merge; open HITL review** |
| **VIF Lineage or it doesn’t exist** | **W-1 JSON-LD lineage** | **All artifacts serialize VIF schema** | **Schema conformance (SHACL) in CI** | **Missing-lineage counter == 0** | **Rollback publish; require witness path** |
|  | **W-2 Confidence hygiene** | **UI shows confidence bands by default** | **Visual regression tests; ECE < θ in eval** | **Live ECE & κ-abstention rate dashboards** | **Raise risk class; force HITL** |
|  | **W-3 Replayability** | **Snapshots pin models/prompts/tools; replay() must succeed** | **Deterministic replay test on nightly pins** | **Replay failure rate < SLO** | **Freeze model route; incident RCA** |
| **SDF-CVF Evolve atomically** | **E-1 Parity gates** | **Same-commit updates across code/docs/tags/traces** | **CI gate: fail if parity missing** | **Drift monitor: spec↔code parity score** | **Quarantine branch; auto-fix task** |
|  | **E-2 Auto-fix/Quarantine** | **Remediation chains for common failures** | **Simulated failures must trigger auto-fix** | **Mean quarantine time < target** | **Escalate to Maintainer; backout** |
|  | **E-3 DORA telemetry** | **CFR/MTTR per repo visible** | **CI exports metrics; dashboards** | **CFR < 10%, MTTR within target** | **Change freeze; postmortem in SEG** |
| **SEG Evidence is a graph** | **S-1 Time-slicing API** | **Every read requires asOf/txnAsOf** | **API lint: reject missing asOf** | **% queries with explicit slice == 100%** | **Block anonymous slice; ops page** |
|  | **S-2 Contradictions first-class** | **Use contradicts edges; no destructive edits** | **Mutation tests forbid delete; require deprecate+edge** | **Contradiction coverage shown in UI** | **Require approver signature** |
|  | **S-3 Export packs** | **Deterministic “pack” bundles (JSON-LD+blobs+manifest)** | **Checksum reproducibility test** | **Pack creation SLO & signature validity** | **Fail audit → halt release** |

**2.6.2 Crosswalk: Design Artifacts & Where They Live**

* **IDs in code & plans.**
  + **Comments: @axiom CMC @constraint C-2**
  + **ACL: gate id: O-3.boundary\_uq\_prov**
  + **SEG: node/edge metadata axiom\_ids: ["VIF","SEG"], constraint\_ids: ["W-1","S-1"]**
* **Dashboards (minimum).**
  + ***Memory*: precision@k@latency, snapshot replay success.**
  + ***Orchestration*: budget adherence, determinism %, κ-abstentions (true-positive).**
  + ***Verifiability*: lineage completeness %, ECE bands, contradictions coverage.**
  + ***Evolution*: CFR, MTTR, parity score.**
  + ***Graph*: % queries with asOf, pack integrity rate.**

**2.6.3 Acceptance Checklists (per subsystem)**

**Memory API (CMC)**

* **Only CreateMemory service account may write (C-1)**
* **Every response includes snapshot\_id (C-2)**
* **Read path executes HHNI→atoms with dumbbell compression (C-3)**
* **SEG edges (derives/supports) recorded on write**

**APOE/ACL**

* **All steps declare budgets (O-1)**
* **Required roles present; verifier isolated (O-2)**
* **Step outputs {artifact, uq, prov, diff} (O-3)**
* **Replay passes on pinned snapshot (W-3)**

**VIF/SEG**

* **JSON-LD valid; SHACL passes (W-1)**
* **Confidence bands visible; ECE < θ (W-2)**
* **Queries specify asOf/txnAsOf (S-1)**
* **Contradictions via edges, not deletes (S-2)**
* **Export pack checksum reproducible (S-3)**

**SDF-CVF**

* **Code/docs/tags/traces changed together (E-1)**
* **Auto-fix chain available; quarantine path tested (E-2)**
* **DORA metrics exported (E-3)**

**2.6.4 Minimal Policy Tests (pseudo)**

**# PT-APOE-001 — Budgets required**

**given: acl\_plan.yaml**

**when: compile**

**then:**

**- assert: every(step).has(budget.tokens & budget.time & budget.tools)**

**# PT-VIF-002 — Lineage required to publish**

**given: artifact.claim**

**when: publish\_request**

**then:**

**- assert: exists(seg.best\_witness\_path(artifact, asOf=now))**

**# PT-SEG-003 — No destructive delete**

**given: mutation.delete(edge\_id)**

**then:**

**- assert: REJECT**

**- assert: SUGGEST deprecate(edge\_id) + add(contradicts)**

**2.6.5 Run-book Triggers (examples)**

* **Budget overflow (O-1): Auto-abstain, surface partial witness, open DEPP sub-plan to re-budget.**
* **Replay fails (W-3): Freeze route; pin previous snapshot; incident type REPLAY-BREAK; RCA must include SEG pack.**
* **Parity fail (E-1): Branch quarantined; launch auto-fix; require two-key HITL merge.**

**2.6.6 Mapping to KPIs**

* **C-series → Memory KPIs (precision@k, replay success, read latency).**
* **O-series → Orchestration KPIs (determinism %, budget adherence, κ true-positive rate).**
* **W-series → Verifiability KPIs (lineage %, ECE).**
* **E-series → DORA KPIs (CFR, MTTR, parity).**
* **S-series → Evidence KPIs (as-of coverage, pack integrity, contradiction coverage).**

**2.6.7 Why this matters**

**This table is the contract of the system. If a feature can’t point to its row—what axiom it fulfills, how it proves compliance at build time, and how it stays honest at runtime—it doesn’t ship**

**Chapter 3 — Design North-Star**

**3.1 C-3PO-with-a-Workshop: Roles & Scopes**

**Intent. Define the *operational cast* of AIMOS/APOE. These roles make “machine communication” tangible: they speak protocols, remember with integrity, compile plans into actions, and keep the shop safe. Think of them as macro-roles that *contain* the micro-roles from APOE (planner, retriever, reasoner, verifier, builder, critic, operator, witness).**

**3.1.1 Macro-roles (what they own, produce, and protect)**

| **Macro-Role** | **Mission** | **Owns** | **Produces** | **Protects** |
| --- | --- | --- | --- | --- |
| **Ambassador** | **Translate between humans ↔ tools ↔ data ↔ devices.** | **Protocol adapters, schema mappers, tool routers.** | **Canonical requests/responses (typed), policy-clean inputs to chains.** | **Capability boundaries, prompt-injection shields, PII redaction.** |
| **Archivist** | **Turn context into memory (CMC/HHNI) with reversibility.** | **Atomization pipelines, indices, snapshots, SEG edges.** | **Atoms, indices, snapshot\_id, lineage deltas, RS metadata.** | **Schema integrity, time slicing, deduplication, rollback points.** |
| **Engineer** | **Compile intent into working software (IDE-in-the-loop).** | **APOE plans, ACL specs, scaffolds, codegen, tests, deploy units.** | **Build artifacts, coverage & parity reports, preview envs.** | **Spec↔code parity, supply-chain hygiene, reproducible builds.** |
| **Counsel** | **Quantify uncertainty and attach witness (VIF/SEG).** | **UQ calibrators, provenance schemas, replay pins.** | **Confidence bands, VIF manifests, replay receipts.** | **Calibration error budgets, attestations, claim lineage.** |
| **Operator** | **Keep the system budgeted, observable, safe.** | **SLOs, token/time budgets, OpenTelemetry, guardrails.** | **Budget dashboards, drift/rot signals, incident packs.** | **Cost ceilings, runtime policy enforcement, containment paths.** |

**Mapping to APOE micro-roles:**

* **Ambassador ↔ router + retriever wrappers**
* **Archivist ↔ retriever + witness (memory side)**
* **Engineer ↔ planner + builder + verifier + critic**
* **Counsel ↔ verifier + witness (provenance/UQ side)**
* **Operator ↔ operator (budgets, SLO, safety)**

**3.1.2 Role contracts (inputs, outputs, gates)**

**Ambassador → Archivist (write path)**

* **Inputs: raw user/tool context C\_t, policy profile, capability token.**
* **Outputs: sanitized request, candidate atoms.**
* **Gates: PII redaction, tool allowlist, prompt-injection filters, max-tokens.**
* **Failure modes: schema drift, late PII, tool SSRF.**
* **Mitigations: strict schemas, redaction traces, egress proxy.**

**Archivist → Engineer (read/plan path)**

* **Inputs: query, asOf, budget.**
* **Outputs: HHNI coarse set → atom pack, RS metrics, snapshot\_id.**
* **Gates: RS threshold, dumbbell compression, contradiction flags from SEG.**
* **Failure modes: lost-in-the-middle, stale indices.**
* **Mitigations: super-index precompute, mark-and-sweep reindex, abstain on low RS.**

**Engineer ↔ Counsel (build/verify path)**

* **Inputs: plan graph (ACL), retrieved context, tool chain.**
* **Outputs: code, tests, deploy artifacts + UQ & VIF manifests.**
* **Gates: unit/integration thresholds, ECE<θ, parity checks, license & SBOM.**
* **Failure modes: spec drift, silent miscalibration, flaky tests.**
* **Mitigations: parity gate, temperature sweep calibrations, quarantined branches.**

**Operator (cross-cutting)**

* **Inputs: chain traces, budgets, telemetry.**
* **Outputs: throttling, re-routing, incident packs, exports.**
* **Gates: κ-abstention on risk, budget aborts, guardrail violations.**
* **Failure modes: budget blowouts, prompt rot, tool compromise.**
* **Mitigations: token governors, rot monitors, sandbox rotations, kill-switch.**

**3.1.3 Budget classes & KPIs (owned per role)**

| **Budget Class** | **Owner** | **Examples** | **Leading KPIs** |
| --- | --- | --- | --- |
| **Token/Latency** | **Operator** | **tokens\_per\_step, p95 latency** | **Budget adherence %, p95 < SLO** |
| **Tool Risk** | **Ambassador/Operator** | **sandbox tier, network scope** | **Zero SSRF/exfil events** |
| **Memory I/O** | **Archivist** | **p95 read/write, index hit-rate** | **precision@k@latency, replay success** |
| **Quality/Parity** | **Engineer** | **coverage %, spec↔code score** | **CFR < 10%, parity ≥ 0.9** |
| **Verifiability** | **Counsel** | **lineage completeness, ECE** | **lineage ≥ 0.99, ECE ≤ θ** |

**3.1.4 RACI for canonical flows**

**Flow A — CreateMemory (ingest → atomize → snapshot → SEG)**

* **R (Responsible): Archivist**
* **A (Accountable): Operator (C-1/C-2 adherence)**
* **C (Consulted): Ambassador (redaction policy), Counsel (VIF tags)**
* **I (Informed): Engineer (downstream reads)**

**Flow B — Plan→Build (APOE) with IDE-in-the-loop**

* **R: Engineer**
* **A: Operator (budgets), Counsel (calibration gate)**
* **C: Archivist (snapshots), Ambassador (tool contracts)**
* **I: All (release announcement)**

**Flow C — Publish Claim (witnessed output)**

* **R: Counsel**
* **A: Operator (policy), Archivist (snapshot pin)**
* **C: Engineer (spec parity), Ambassador (display/UX)**
* **I: Compliance**

**3.1.5 Handoffs (message shapes)**

**All inter-role handoffs are typed envelopes:**

**{**

**"envelope\_id": "uuid",**

**"asOf": "2025-10-18T12:34:56Z",**

**"snapshot\_id": "snap\_…",**

**"actor": "Engineer|Archivist|…",**

**"budget": {"tokens": 12\_000, "latency\_ms": 1500, "tools": ["sandbox:v2"]},**

**"payload": { "... domain object ..." },**

**"vif": { "model": "gpt-x", "weights\_hash": "sha256:…", "prompt\_template": "tpl\_…"},**

**"uq": {"ece": 0.03, "entropy": 1.7, "band": "green"}**

**}**

**Rule: no envelope → no work. Missing snapshot\_id or asOf → reject (C-2, S-1).**

**3.1.6 Role-scoped SLOs (minimum bars)**

* **Ambassador: 0 critical guardrail escapes; ≥99.9% schema-valid messages.**
* **Archivist: precision@10 ≥ target at p95≤SLO; ≥99.99% snapshot durability; replay success ≥99.5%.**
* **Engineer: spec↔code parity ≥0.9; coverage ≥80%; CFR <10%; MTTR within target.**
* **Counsel: lineage completeness ≥0.99; ECE ≤ θ; replayable claims 100%.**
* **Operator: budget breach <1% chains; κ-abstention true-positive rate ≥ target; MTTD < 5 min.**

**3.1.7 Common failure modes & playbooks**

* **Prompt injection → tool misuse (Ambassador).  
  *Signal:* unexpected DNS/HTTP egress. *Action:* revoke capability token; rotate sandbox; mark SEG contradiction on offending instruction; require HITL before re-enable.**
* **Index drift (Archivist).  
  *Signal:* RS-lift drop, precision@k regression. *Action:* trigger reindex job; tighten dumbbell compression; annotate SEG with drift incident; temporary κ-raise for risky reads.**
* **Spec/code divergence (Engineer).  
  *Signal:* parity gate fail, flaky tests. *Action:* quarantine branch; auto-fix suggestions; require two-key approval.**
* **Overconfident publish (Counsel).  
  *Signal:* low calibration, missing witness path. *Action:* block publish; run temperature sweep; attach VIF; reroute to HITL.**
* **Budget blowout (Operator).  
  *Signal:* tokens/latency > budget. *Action:* abort chain step; suggest compression; open DEPP re-plan; notify user with partial witness.**

**3.1.8 Example: end-to-end “spec change → running code”**

1. **Ambassador parses the human edit; strips PII; emits typed change request.**
2. **Archivist atomizes change; links to prior spec via HHNI; creates snapshot\_id.**
3. **Engineer compiles APOE plan: update code + tests; runs IDE-loop; parity gate passes.**
4. **Counsel calibrates UQ, builds VIF; replays on pinned snapshot; green band.**
5. **Operator checks budgets and SLOs; deploys to preview; telemetry attached.**
6. **Publish with SEG lineage: claim → evidence → decision, as-of the deployment commit.**

**3.1.9 Extension points**

* **New tools: Ambassador extends capability tokens & schema; Operator assigns sandbox tier.**
* **New modalities: Archivist adds atom schema + embedder; HHNI updates taggers.**
* **New guardrails: Counsel updates SHACL/JSON-LD; UI adds banding; Operator enforces.**
* **New stacks: Engineer adds templates/components; parity and license gates inherit.**

**3.1.10 Doctrine (one-liners per role)**

* **Ambassador: *Translate without leaking.***
* **Archivist: *Store what you can prove, and prove what you store.***
* **Engineer: *Plans before code; parity before pride.***
* **Counsel: *No witness, no publish.***
* **Operator: *Budgets are real; safety is a feature.***

**With these scopes and contracts, “C-3PO-with-a-Workshop” stops being a metaphor and becomes an operating model: the right agent, with the right budget, doing the right work, leaving the right evidence.**

**3.2 Safety as Capability (abstention, κ-gating)**

**Intent. Treat safety not as a filter bolted on at the end, but as a *first-class control system* that decides what not to do—early, explicitly, and reproducibly. κ-gating is the mechanism that lets APOE halt, narrow, or reroute work when uncertainty, policy, or budget signals say “stop.”**

**3.2.1 Core concepts**

* **Abstention. A deliberate, explainable refusal to proceed *now* with the current inputs and budgets. Abstention is not failure; it is control.**
* **κ (kappa). A configurable abstention threshold applied to one or more signals (retrieval quality, uncertainty, policy risk, budget pressure). If the composite risk exceeds κ, the system gates.**
* **Gate. A typed decision point in a chain step that consumes metrics + policy and produces {proceed | degrade | reroute | HITL | abort} plus a witness for why.**

**3.2.2 Signals that feed κ**

**Let a step produce metrics:**

* **Retrieval: RS ∈ [0,1] (Section 4–5), plus coverage and novelty.**
* **Uncertainty (UQ): entropy H, expected calibration error ECE, variance of sampled outputs Var, and disagreement score Δ (e.g., temperature sweep/ensemble).**
* **Policy: Boolean predicates over content/tooling (PII present? disallowed domain? prompt-injection heuristics?).**
* **Budget: token & latency utilization β\_tok, β\_lat ∈ [0,1], cost burn rate, remaining budget slack.**
* **Context risk: data classification (public/internal/sensitive), action class (read-only vs. write/vend).**

**Define a normalized risk vector  
r = ⟨ r\_RS, r\_UQ, r\_policy, r\_budget, r\_context ⟩, where, for example:**

* **r\_RS = 1 − RS**
* **r\_UQ = clamp(w1·norm(H) + w2·ECE + w3·norm(Var) + w4·Δ)**
* **r\_policy = 1 if any hard predicate fails, else 0 (or fractional under soft rules)**
* **r\_budget = max(β\_tok, β\_lat)**
* **r\_context derives from sensitivity class (e.g., 0.7 for PII)**

**Composite risk (monotone aggregator):**

**R\* = max( α\_RS·r\_RS,**

**α\_UQ·r\_UQ,**

**α\_policy·r\_policy,**

**α\_budget·r\_budget,**

**α\_ctx·r\_context )**

**Proceed iff R\* < κ. Otherwise, gate.**

**Design note. Use max for fail-closed behavior on hard risks (policy). For softer regimes, L∞ can be replaced with a convex combination; always preserve monotonicity and calibrate α-weights per domain.**

**3.2.3 Gate outcomes (finite set)**

| **Outcome** | **When triggered** | **Action** | **Evidence (VIF/SEG)** |
| --- | --- | --- | --- |
| **Proceed** | **R\* < κ\_low** | **Continue with planned tool/role.** | **Record metrics + benign decision record.** |
| **Degrade** | **κ\_low ≤ R\* < κ\_mid** | **Reduce capability: smaller context, safe-mode tools, redact outputs, lower temperature.** | **Attach “degradation profile” and rationale.** |
| **Reroute** | **R\* ≥ κ\_mid but fix is available** | **Change plan: add retrieval, add verification, switch to safer template.** | **Link to new sub-plan and its budgets.** |
| **HITL** | **R\* ≥ κ\_high on high-risk data/actions** | **Pause; request two-key approval or edits; show diffs and impacts.** | **Store reviewer identity, decisions, timestamps, comments.** |
| **Abort** | **r\_policy=1 hard violation or budget breach** | **Stop; emit safe explanation and next steps; snapshot state for replay.** | **Incident-grade VIF + SEG contradiction edge if hostile input.** |

**Hysteresis prevents flapping: once degraded, require R\* < κ\_low − δ to return to full power.**

**3.2.4 Where κ-gates live (chain anatomy)**

* **At plan boundaries. Every APOE step starts with an Ingress Gate (pre-work) and ends with an Egress Gate (pre-handoff).**
* **On tool invocation. Before any external call (browser, code exec, file I/O), enforce Capability Gates (policy + budget).**
* **Before publish. Counsel’s Witness Gate checks UQ calibration and lineage completeness; no witness → no publish.**

**3.2.5 Policy model (hard vs. soft)**

* **Hard predicates (binary): disallowed domains, PII exfiltration, local file read outside sandbox, network egress beyond egress proxy → immediate Abort.**
* **Soft predicates (graded): sensitive topics, thin retrieval, low calibration → Degrade or Reroute.**

**Policies are declarative (e.g., Rego/SHACL-like over JSON-LD/VIF envelopes) and versioned; the active policy hash is part of the witness.**

**3.2.6 Calibration and thresholds**

* **Calibrate UQ offline with held-out corpora; compute ECE bins; set κ per domain and action class.**
* **Per-role κ. Lower κ for *Engineer* deploy steps; higher κ for read-only retrieval.**
* **Context-aware κ. Raise κ for sensitive datasets, lower for public FAQs.**
* **Budget-aware κ. When β\_tok or β\_lat approaches 1.0, κ tightens (risk of prompt-rot / timeout rises).**

**Rule of thumb: If ECE > 0.05 on the class of tasks in play, force Degrade (more retrieval, lower temperature) or Reroute to verification.**

**3.2.7 Degradation strategies (safe modes)**

* **Context diet: apply dumbbell compression more aggressively; pin top-K atoms with highest TPV; avoid mid-body bloat.**
* **Template hardening: switch to strict, minimal prompts; disable creative expansions; force structured outputs.**
* **Tool downgrades: read-only sandboxes; disable network writes; deny code exec; cap concurrency.**
* **Explainability on: require rationales, cite-keys, and uncertainty bands in UI.**

**3.2.8 APOE/ACL expression (executable example)**

**pipeline BuildFeatureX {**

**budget { tokens: 12000, latency\_ms: 2000, tools: ["sandbox:v2"] }**

**step retrieve\_spec {**

**gate ingress {**

**require RS >= 0.65**

**require UQ.ece <= 0.05**

**forbid policy.prompt\_injection**

**on\_fail {**

**if (RS < 0.65) reroute add\_retrieval;**

**if (UQ.ece > 0.05) degrade { temp: 0.2, style: "strict" };**

**if (policy.prompt\_injection) abort "Policy violation";**

**}**

**}**

**run retriever …**

**gate egress { emit vif, uq, rs }**

**}**

**step generate\_code depends retrieve\_spec {**

**gate ingress { require budget.tokens\_left >= 0.3 }**

**run builder …**

**gate egress {**

**require tests.pass\_ratio >= 0.9**

**require parity.spec\_code >= 0.9**

**on\_fail { quarantine; hitl request "Spec/code parity low" }**

**}**

**}**

**}**

**3.2.9 UX for abstention (what the user sees)**

* **Plain reason + next action. “I’m pausing because retrieval is thin (RS=0.42). I can: (a) search your docs, (b) ask 3 clarifying questions, or (c) proceed in safe-mode.”**
* **Bands, not decimals. Green/Amber/Red badges with short tooltips; full metrics in a collapsible panel.**
* **Traceability. A “Why paused?” link opens the VIF bundle (policy hash, κ, metrics, snapshot id).**
* **One-click paths. “Approve once”, “Always approve for this repo”, or “Tighten policy”.**

**3.2.10 Failure modes & mitigations**

| **Mode** | **Symptom** | **Mitigation** |
| --- | --- | --- |
| **Gate flapping** | **Rapid degrade/proceed cycles** | **Add hysteresis δ; rate-limit re-entry; cache previous decision for Δt.** |
| **Silent overconfidence** | **Low ECE training, high test ECE** | **Recalibrate; enforce temperature sweeps; require ECE witness before publish.** |
| **Latency blowouts** | **Reroute loop expands chain** | **Put hard caps on reroutes; prefer *Degrade* once per branch.** |
| **Policy false positives** | **Over-blocking benign content** | **Split hard vs. soft predicates; log counter-examples; adjust rules via HITL.** |

**3.2.11 Guarantees**

* **Determinism under fixed context. Given the same M\_t, policy hash, budgets, and seeds, κ-gates make the same decisions (or abstentions).**
* **Provenance. Every gate writes a decision envelope to SEG (who/what/why/when, including κ, R\*, and active policy).**
* **Composability. Gates are first-class ACL constructs; they compose with roles, budgets, and tools without side-channels.**

**Doctrine. *If you can’t prove it’s safe, you can’t prove it’s useful.* κ-gating is how the system proves it.**

**3.3 IDE-in-the-Loop Rationale**

**Thesis. If the system must *prove* what it knows and *ship* what it reasons, then reasoning and software production cannot be split. The IDE belongs inside the orchestration loop so that spec ↔ code ↔ evidence stay in lock-step (SDF-CVF), κ-gates can halt unsafe actions, and every artifact carries a witness (VIF) into the SEG.**

**3.3.1 Why an IDE inside APOE (and not a separate toolchain)**

**A. Parity over time (SDF-CVF).  
Separate chat + dev stacks drift. IDE-in-the-loop binds doc edits, code changes, test traces, and deployment artifacts to the same atomic commit and the same snapshot id, enforcing parity gates at each boundary.**

**B. Determinism & replay (APOE).  
Plans compile to typed steps that invoke the actual toolchain (scaffold, lint, test, package, deploy) under fixed budgets and seeds. Replays use frozen containers + lockfiles; identical inputs produce identical build graphs.**

**C. Inline safety (κ-gating).  
Before code exec, network egress, or deployment, gates check RS/UQ/policy/budget. Unsafe paths degrade, reroute to verification/HITL, or abort—*before* anything touches prod.**

**D. Single lineage (VIF/SEG).  
Specs, prompts, code blobs, test results, coverage, and SBOMs are linked in one provenance thread. Audit reads one graph instead of reconciling multiple CI/CD logs.**

**E. Latency & learning.  
Tight loop ⇒ faster diagnosis and better auto-repair. Failures feedback into prompts and plans immediately; prompts get hardened by concrete build/test telemetry.**

**3.3.2 Design goals & non-goals**

* **Goals:**
  1. **Reproducible builds (containerized, pinned deps, signed outputs)**
  2. **First-class tests (gates use coverage & parity, not vibe)**
  3. **Bounded autonomy (κ-gates + HITL for high-risk merges)**
  4. **End-to-end provenance (VIF envelopes into SEG)**
  5. **Budget discipline (tokens, time, cost surfaced as first-class constraints)**
* **Non-goals:  
  A UI text editor replacement; unrestricted code exec; bypassing corporate SDLC.**

**3.3.3 The pipeline surfaces (what the IDE exposes to APOE)**

| **Stage** | **What APOE sees** | **Gates (examples)** | **Witness artifacts** |
| --- | --- | --- | --- |
| **Blueprint** | **YAML/ADR/spec tree** | **schema, policy, ambiguity check** | **spec@sha, prompt templates** |
| **Scaffold** | **file graph, diffs** | **license/headers, policy packs** | **SBOM, license report** |
| **Codegen/Refactor** | **patches, ASTs** | **parity(spec↔code), lint, SAST** | **diffs, lint/SAST reports** |
| **Test** | **suite matrix, coverage** | **pass ≥ θ, mutation score ≥ θ** | **junit, coverage.json** |
| **Package** | **image/artifacts** | **supply-chain attestations** | **provenance (in-toto), digest** |
| **Deploy** | **env plan, rollout** | **change window, blast radius** | **change ticket, rollout events** |
| **Observe** | **logs, traces, KPIs** | **SLO error budget checks** | **traces, metrics snapshots** |

**All stages run in ephemeral, capability-scoped sandboxes with read/write tools whitelisted per step.**

**3.3.4 Roles in the loop (APOE)**

* **Planner compiles blueprint → plan graph.**
* **Builder generates/refactors code under budgets.**
* **Verifier runs tests, computes parity, mutation, security.**
* **Operator packages/deploys with rollout policies.**
* **Counsel emits UQ & κ decisions; Witness stamps VIF; Archivist writes SEG.**

**3.3.5 Safety & policy hardening**

* **Pre-exec gates: no code exec without sandbox + policy hash; no network egress without egress proxy & allowlist.**
* **Degradation recipes: strict prompts, read-only tools, smaller context, temperature clamp, extra retrieval.**
* **Two-key merges: risk≥θ requires dual human approvals; both identities and diffs logged to SEG.**

**3.3.6 Minimal executable example (ACL)**

**pipeline AppV1 {**

**budget { tokens: 20000, latency\_ms: 4000, tools: ["fs:sandbox", "node:20", "docker:build", "k8s:staging"] }**

**step plan\_blueprint {**

**run planner with { blueprint: "specs/app.yaml" }**

**gate egress { require vif.complete; emit seg.link("ADR-001") }**

**}**

**step scaffold depends plan\_blueprint {**

**gate ingress { require policy.licenses.allowed; require κ(R\*) < 0.4 }**

**run scaffold with { template: "react+fastapi", dst: "./app" }**

**gate egress { require sbom.ok; emit vif }**

**}**

**step generate\_code depends scaffold {**

**gate ingress { require parity.spec\_code >= 0.8 }**

**run builder with { repo: "./app", targets: ["auth", "users"] }**

**gate egress {**

**require lint.errors == 0**

**require sast.crit == 0**

**on\_fail { quarantine; hitl "Security issues" }**

**emit vif**

**}**

**}**

**step test depends generate\_code {**

**run verifier with { cmd: "pnpm test --coverage" }**

**gate egress { require tests.pass >= 0.9; require coverage.line >= 0.85; emit vif, coverage }**

**}**

**step package depends test {**

**run packager with { dockerfile: "./app/Dockerfile" }**

**gate egress { require provenance.slsa >= 2; emit attestation }**

**}**

**step deploy depends package {**

**gate ingress { require window.change\_open; require risk.blast\_radius <= "staging" }**

**run operator with { target: "staging", strategy: "bluegreen" }**

**gate egress { require slo.error\_budget.ok; emit rollout, traces }**

**}**

**}**

**Every gate writes a decision envelope (metrics, κ, policy hash) to the SEG; every artifact carries a VIF (model id, weights hash, prompts, snapshot id).**

**3.3.7 Failure modes & mitigations**

* **Spec↔code drift → enforce parity gates; auto-generate failing tests; quarantine branch.**
* **Prompt-rot / flaky builds → pin prompts/templates per step; cache seeds; use deterministic containers.**
* **Over-automation → κ increases with risk class; HITL on merges & deploys; degrade instead of “try harder.”**
* **Dependency volatility → lockfiles + SBOM diffs; fail closed on unexpected supply-chain changes.**

**3.3.8 KPIs & guarantees**

* **KPIs: spec↔code parity ≥ 0.9; test pass ≥ 0.9; line coverage ≥ 0.85; CFR ≤ 10%; MTTR ≤ target; κ true-positive rate.**
* **Guarantees: deterministic replays under frozen snapshots; provenance-complete releases; bounded autonomy via κ-gates; one evidence graph for all actions.**

**Doctrine. *Ship reasoning the way you ship code:* reproducible, witnessed, and safe. The IDE in the loop is how APOE turns plans into running, verifiable systems without breaking the invariants that keep AIMOS honest.**

**3.4 Threat & Misuse Model (policy surface)**

**Thesis.** AIMOS/APOE operates on living memory, executes tools, and ships software. That makes it a high-value target. We treat **security, safety, and privacy as first-class capabilities**: the system must *anticipate* misuse, *constrain* power, *measure* uncertainty, and *prove* lineage.

**3.4.1 Assets, trust boundaries, and actors**

**Crown-jewel assets**

* **CMC**: atoms, indices, snapshots; HHNI dependency graph; secrets in metadata.
* **SEG**: claims, evidence, tool calls, decisions; time-sliced lineage.
* **APOE plans**: ACL pipelines, budgets, prompts/templates, κ policies.
* **Tool surface**: sandboxes, credentials, egress channels, data connectors.
* **Build & deploy**: source repos, SBOMs, images, keys, environment configs.
* **Telemetry**: logs, traces, κ/UQ dashboards, cost/budget data.

**Trust boundaries**

1. **User > Orchestrator** (prompt input, file upload)
2. **Orchestrator > Tools** (exec, network, filesystem)
3. **Orchestrator > Memory** (write/read to CMC & SEG)
4. **Build > Runtime** (package, sign, deploy)
5. **Observers** (APIs, dashboards, exports)

**Adversary profiles**

* **External opportunist** (jailbreaks, prompt injection, exfiltration)
* **Grey-box integrator** (malicious connector, poisoned RAG corpus)
* **Insider** (credential abuse, policy bypass, data dragnet)
* **Supply-chain actor** (malicious dependency/image/template)
* **Advanced persistent** (long-horizon poisoning of memory & lineage)

**3.4.2 Threat taxonomy (STRIDE×LLM) and misuse catalog**

| **Class** | **Example** | **Impact** | **Controls (preview)** |
| --- | --- | --- | --- |
| **Prompt injection / Jailbreak** | Instruction smuggling in docs | Tool misuse, policy bypass | Content firewalls, allowlisted tool intents, κ-abstention, role separation |
| **RAG poisoning** | Corrupted/forged sources | Hallucinated but “grounded” output | Source signing, document allowlists, triage scores, contradiction edges in SEG |
| **Exfiltration** | Coax model to leak secrets | Data breach | DLP/redaction, secret shims, output scanners, data diodes |
| **SSRF / Egress abuse** | Untrusted URL fetch | Internal network exposure | Egress proxy, DNS/HTTP allowlists, fetch-capability tokens |
| **Context flooding** | Token-budget DoS | Starvation, κ drift | Rate limits, quotas, dumbbell compression, early budget clamps |
| **Tool abuse** | Shell/fs misuse | Lateral movement, tampering | Capability-scoped sandboxes, seccomp, read-only mounts |
| **Supply-chain** | NPM/PyPI trojan, base image | CI compromise, backdoor | Repro builds, SBOM, pin+verify, image signing & admission |
| **Provenance forgery** | Fake VIF/SEG records | Audit failure | Signed snapshots, immutable logs, cross-checks |
| **HITL bypass** | Social or UX gap | Unsafe merge/deploy | Two-key approvals, risk bands, time-locks |
| **Policy rot** | Silent drift of guardrails | Surface shrinkage | Policy versioning, canary prompts, regression suites |

**3.4.3 Policy surface: the control lattice**

We structure defenses as **Prevent → Detect → Respond**, mapped to each boundary.

**A) Input boundary (User → Orchestrator)**

**Prevent**

* **Content firewall**: structural parsing + allowlisted formats; strip/neutralize model-targeting control tokens from user content.
* **Classifier ensemble**: prompt-injection, PII, malware; conservative defaults feed κ.
* **Budget clamps**: per-request token/time/IO ceilings up front.

**Detect**

* **Anomaly sensors**: entropy spikes, repetition, tool-intent deviation.
* **UQ tripwires**: high epistemic uncertainty near sensitive intents.

**Respond**

* **Abstain** with actionable rubric; **degrade** (read-only tools, lower temp, stricter retrieval); **HITL** escalation for risky intents.

**B) Tool boundary (Orchestrator → Tools)**

**Prevent**

* **Capability tokens** (fine-grained: fs.read, fs.write, net.fetch[allowlist], exec[toolset]); least-privilege by step.
* **Ephemeral sandboxes**: seccomp, network namespaces, read-only mounts, time & CPU quotas.
* **Egress proxy**: DNS/HTTP allowlists; signed request headers; response size & type limits.

**Detect**

* **Policy audit hooks**: all tool invocations mirrored to SEG; diff-aware alarms (unexpected domains, file paths).
* **Runtime guards**: taint tracking for sensitive content flowing into tools.

**Respond**

* **Kill-switches** per capability; **auto-quarantine** offending plan branch; **evidence snapshot** for RCA.

**C) Memory boundary (Orchestrator ↔ CMC/SEG)**

**Prevent**

* **Single-writer flow** with schema & policy validation; **snapshot-first** semantics.
* **Source integrity**: signed documents; allowed repositories; checksum enforcement.
* **Write gating**: HHNI DD threshold, policy packs (PII, license), role approvals.

**Detect**

* **Graph consistency**: contradiction edge rate, sudden indegree/outdegree shifts, unusual TPV changes.
* **Poisoning heuristics**: surge of near-duplicate atoms from new sources; low-trust domains.

**Respond**

* **Deprecate not delete** (tombstones); **time-slice quarantine**; **rebuild indices**; HITL adjudication.

**D) Build & deploy boundary**

**Prevent**

* **Reproducible builds**: pinned deps, hermetic toolchains, content-addressed artifacts.
* **SLSA/SBOM** gates; **image signing**; policy admission (cluster side).
* **Change windows & blast-radius caps**; progressive delivery (canary/blue-green).

**Detect**

* **Pre-prod chaos**: test fault injection; mutation testing.
* **Runtime SLO monitors**: error budgets with SEG-linked rollouts.

**Respond**

* **Auto-rollback** on SLO breach; **feature flags** off; **post-mortem** to SEG with VIF pack.

**E) Observation & export boundary**

**Prevent**

* **Read-only audit API**; scope-limited tokens; cell-level redaction.
* **Synthetic labeling** for AI-generated content; watermarking when applicable.

**Detect**

* **Access anomaly**: unusual query cardinality, exfil-shaped patterns.
* **Replay drift**: divergence between replay and live lineage.

**Respond**

* **Key rotation**; **lease revocation**; export freeze & review.

**3.4.4 κ-gating and abstention policy**

* **Risk bands**: low < κ\_1 < medium < κ\_2 < high.
* **Actions**:
  + *Low*: proceed; log κ & UQ.
  + *Medium*: degrade (stricter retrieval, read-only tools), require verifier step.
  + *High*: abstain; present rationale + minimal sufficient questions; offer HITL.
* **Calibration**: κ thresholds tuned per domain, validated against adversarial suites (jailbreak corpora, injection galleries).

**3.4.5 Red-team harness & regression suites**

* **Jailbreak corpus** (continuously updated), **RAG poisoning set** (look-alikes, forged refs), **connector fuzzing** (URL schemas, MIME tricks), **policy canaries** (known-bad prompts that *must* be caught).
* **Scoring**: block rate, false-positive rate, time-to-detect, κ true-positive rate, coverage of guard families.
* **Automation**: nightly runs; deltas block promotion of policy packs.

**3.4.6 Incident playbooks (condensed)**

**PI-01 Prompt-injection**

1. Freeze plan branch (quarantine snapshot).
2. Rotate capability token set for impacted tools.
3. Add contradiction edges for tainted claims; re-index.
4. Run adversarial suite; adjust policy weights; publish VIF report.

**RP-02 RAG poisoning**

1. Isolate source set; verify signatures/checksums.
2. Recompute TPV/RS with source penalty; rebuild HHNI for slice.
3. Mark deprecations; notify owners; export regulator pack if needed.

**SC-03 Supply-chain**

1. Revoke image keys; roll back to last signed image.
2. SBOM diff triage; lockfile audit; increase admission policy level.
3. SEG post-mortem with dependency tree.

**3.4.7 Governance & human factors**

* **Two-key merges** for high-risk changes (crypto-bound approvals; independent reviewers).
* **Separation of duties**: Planner/Builder vs. Operator approvals; Counsel (UQ) cannot be Builder.
* **UX for refusal**: transparent κ rationale; minimal, specific next steps; never “just try harder.”
* **Privacy by default**: data minimization, purpose binding, retention timers with review.

**3.4.8 Example: policy-aware APOE step (ACL)**

step fetch\_context {

budget { tokens: 1500, time\_ms: 1500, tools: ["net.fetch[allow:docs.example.com,\*cdn\*]","fs.read[tmp]"] }

gate ingress {

require content.scan.injection == "clean";

require pii.redaction == "applied";

require κ(input\_risk) < 0.45;

}

run retriever with { query: Q, top\_k: 8, policy: "no\_external\_forms,no\_js" }

gate egress {

require sources.signed >= 0.9;

require vif.complete;

on\_fail { abstain "Untrusted sources"; hitl "Source integrity" }

}

}

**3.4.9 KPIs for the policy surface**

* **Block/jailbreak rate** (↑), **false-positive rate** (↓), **time-to-detect** (↓)
* **κ true-positive rate** (↑), **exfil attempt rate** (↓), **RAG integrity score** (↑)
* **Supply-chain conformance** (SLSA level, SBOM completeness), **replay fidelity** (↑)

**Doctrine.** *No capability without a constraint; no answer without a witness.* We don’t “bolt on” safety—AIMOS/APOE **compiles safety** alongside reasoning, and proves it in the graph.

**Part II — Context Memory Core (CMC) & HHNI**

**Chapter 4. Fractal Memory Hierarchy**

**Intent.** Specify the data model and invariants for a memory-native system that turns every bit of context into structured, queryable, reversible memory—across all modalities—without sacrificing provenance or performance.

**4.1 Atoms & Molecules (schemas)**

**Design goals.** Smallest useful unit (**atom**), recursively composed into larger structures (**molecules**), all carried by an evidence-first substrate. No write exists without a snapshot; no composition breaks lineage.

**4.1.1 Core identifiers**

* node\_id: content-address (e.g., sha256 of canonical payload + headers).
* snapshot\_id: monotone, signed capture of a consistent state.
* rev: semantic revision counter (per node lineage).

**4.1.2 Atom (immutable payload + rich headers)**

{

"node\_id": "sha256:…",

"type": "atom",

"modality": "text|code|image|audio|event|vector|schema|plan",

"content": { "inline": "...", "media\_ref": null }, // object-store ref if large

"headers": {

"created\_at": "2025-10-17T12:34:56Z",

"author": "user|agent-id",

"source": { "uri": "...", "offset": [123, 456] },

"lang": "en",

"license": "…"

},

"analysis": {

"summary": "…",

"embedding": { "model": "text-embedding-004", "dim": 768, "vec": "…" },

"tags": [

{ "name": "topic/security", "tpv": { "priority": 0.92, "relevance": 0.88, "decay": 0.03 } }

],

"qs": 0.84 // quality score for this atom (post-ingest checks)

},

"provenance": {

"model\_id": "llm@vendor#version",

"weights\_hash": "sha256:…",

"prompt\_template\_id": "tpl://…",

"tool\_calls": [],

"witness": "seg://node/…"

},

"snapshot\_id": "snap:2025-10-17T12:34:56Z",

"rev": 1

}

**4.1.3 Molecule (ordered DAG over atoms/molecules)**

{

"node\_id": "sha256:…",

"type": "molecule",

"role": "paragraph|function|conversation|session|codebase|blueprint",

"children": [

{ "ref": "sha256:atom1", "edge": "contains", "order": 0 },

{ "ref": "sha256:atom2", "edge": "contains", "order": 1 }

],

"indexes": { "span\_tokens": [0, 512], "span\_time": ["…", "…"] },

"analysis": {

"summary": "hierarchical sum of children",

"embedding": { "model": "…", "vec": "…" },

"tags": [ { "name": "domain/legal", "tpv": { "priority": 0.73, "relevance": 0.81, "decay": 0.01 } } ],

"qs": 0.87

},

"constraints": { "acyclic\_contains": true, "max\_depth": 8 },

"provenance": { "...": "…" },

"snapshot\_id": "snap:…",

"rev": 4

}

**4.1.4 Edges (evidence-native)**

Edge types are first-class records, not just pointers:

* Structural: contains, is\_summary\_of, derives\_from
* Semantic: relates\_to, contradicts, supports, duplicates
* Operational: depends\_on, executes, witnesses

{

"edge\_id": "sha256:src->dst:type@snapshot",

"src": "sha256:…",

"dst": "sha256:…",

"type": "relates\_to",

"weight": 0.83,

"policy": { "pii": false, "license\_ok": true },

"time": { "valid\_from": "…", "valid\_to": null },

"provenance": { "witness": "seg://edge/…" },

"snapshot\_id": "snap:…"

}

**Constraints → Invariants.**

* **CMC/C-2 Snapshot-first:** no edge or node visible without snapshot\_id.
* **SEG/S-2 Contradictions:** model “conflict” with contradicts edges; never delete evidence.

**4.2 Atomicity, Recursion, Emergence**

**4.2.1 Formal structure**

Let (atoms∪molecules). Define:

* is **acyclic**, bounded depth .
* is arbitrary but **time-sliced**.

A **molecule view** materializes the ordered tree at time , folding children summaries upward. Emergence arises because embeddings/tags of molecules are *functions* of children plus **contextual gains** (usage, recency).

**4.2.2 Operations**

* **Fold/Unfold:** compute molecule embedding via weighted sum + nonlinearity (e.g., gated pooling on child TPVs).
* **Dedup:** two atoms with identical canonical payload share node\_id—dedup at write.
* **Merge:** new molecule over existing children produces a **new node**; immutability preserves replayability.
* **Tombstones:** deprecations create valid\_to on edges; nothing is silently removed.

**4.2.3 Safety theorems (sketches)**

* **Reversibility:** Given a snapshot, contains is acyclic ⇒ an ordered traversal reconstructs all child payloads (no information loss beyond compression already witnessed in analysis.summary, which is preserved as a separate atom).
* **Monotonicity:** new snapshots add nodes/edges or close intervals; earlier queries at as\_of remain stable (SEG time slicing).

**4.3 Modality independence & payload abstraction**

**4.3.1 The envelope pattern**

Every node wraps payload in a uniform envelope:

{

"payload": {

"content\_type": "text/plain; charset=utf-8",

"bytes": "…base64…", // optional if inline text present

"inline\_text": "…", // small text/content

"object\_ref": "obj://bucket/key" // large objects

},

"meta": {

"token\_span": [0, 256],

"duration\_ms": null,

"mime\_hints": ["text/markdown"]

}

}

Adapters handle:

* **Text/code:** tokenization, AST n-grams, symbol tables (for code).
* **Images/audio:** perceptual hashes, CLIP-style embeddings, temporal cues.
* **Events/logs:** key–value canonicalization, schema capture.

**4.3.2 Chunking strategy**

* **Atoms (text/code):** target **~512 tokens** (hard <= 2k); prefer **sentence-aligned** splits; enforce **semantic overlap** (5–10%) to maintain recall.
* **Long media:** store as object; create **derived atoms** for captions, OCR, spectrogram notes.
* **Streams:** turn each logical span (e.g., 2–5s audio) into an atom; group into a session molecule.

**4.3.3 Cost & storage**

* **Inline threshold:** ≤ 4 KB inline, else object store.
* **Indexing:** embeddings stored in vector store; metadata & edges in a graph or document DB; payload in object store.
* **Compression:** longer molecules carry summaries; atoms remain raw for reversibility.

**4.4 Embeddings, Tags, and TPV**

**Purpose.** Make retrieval **adaptive** and **auditable**: tags carry priority; embeddings carry semantics; both decay with time; all become inputs to **RS** and policy routing.

**4.4.1 Tag Priority Vector (TPV)**

For tag on node :

* : task/user-specific weight (e.g., “compliance/PII” = high).
* : learned from usage (clickthrough, dwell, grounding hits).
* : time decay with horizon .

**Update rule (online):**

where come from the current plan/feedback.

**4.4.2 Quality, depth, and dependency**

* **QS (quality score):** per node; combines validation checks (language detect, PII scan, schema conformance), model-rated coherence, and provenance completeness.
* **IDS (indexing depth score):** logarithmic function of **HHNI depth coverage** and **connection density** for that node’s subtree.
* **DD (dependency delta):** predicted impact of modifying/using the node, measured against connected dependencies (conflict potential).

**Retrieval Score** (restate, normalized):

Calibration ensures RS ranks correlate with relevance probability.

**4.4.3 Embedding policy**

* **Model discipline:** store {model\_id, version, dim, space}; switching models creates **parallel vectors** with a **join view**; never silently overwrite.
* **Projection:** keep an **operational head** (low-dim projection) for fast DVNS dynamics; use full vectors for ranking.
* **Safety:** hash vectors at rest if needed; enforce access scopes on sensitive embeddings (image faces, voiceprints).

**4.4.4 Example (end-to-end)**

**Input.** A user pastes a TypeScript function with JWT logic.  
**Write path outcome.**

* Atoms: code atom (function body), text atom (docstring), event atom (paste).
* Molecule: function molecule contains the above atoms.
* Tags: topic/auth, risk/secret-handling, lib/jose.
* TPV: topic/auth → [0.85, 0.77, 0.02].
* Embeddings: code and text embeddings + pooled molecule embedding.
* Edges: depends\_on → lib/jose atom; relates\_to → prior “auth patterns” molecule.
* RS: computed per node; high for the molecule due to depth and low DD.
* Snapshot: snap:2025-10-17T… signed; SEG stores witness tuples.

**4.5 Interfaces (authoritative, minimal)**

**4.5.1 CreateMemory (single writer; C-1)**

POST /memory/atoms-or-molecules

Body: {

payload, modality, role?, parent?, source?, policy?,

create\_if\_absent: true

}

→ 201 Created { node\_id, snapshot\_id, rev, witness\_ref }

Validations: canonicalization, duplicate detection (content-address), policy pre-checks (PII/license). Reject if snapshot signing fails.

**4.5.2 Link (edge insert)**

POST /memory/edges

Body: { src, dst, type, weight?, policy? }

→ 201 Created { edge\_id, snapshot\_id, witness\_ref }

Policy mask applied before write. Contradictions generate type:"contradicts" edges; do **not** delete.

**4.5.3 Read (hierarchical)**

GET /memory/{node\_id}?as\_of=…&depth=…&view=atoms|molecule

→ ordered view, embeddings/tags, RS per node, lineage pointers

Supports **dumbbell** read (coarse→fine) for long contexts.

**4.6 Failure modes & guardrails**

* **Cycle injection (contains):** reject; emit witness:violates(acyclic\_contains).
* **Semantic drift:** QS drops over time; decay ensures RS reflects staleness; APOE may **abstain** at κ.
* **Over-tagging:** cap tags per node; demote low-use tags via decay.
* **Embedding migration:** run dual-write (old+new) with snapshot binding; DVNS consults the overlay index while migration completes.
* **PII/leakage:** policy engine gates CreateMemory and Link; redaction produces a **new** atom; originals remain sealed with access controls.

**4.7 Invariants crosswalk (this chapter)**

* **CMC:** atomize→index→snapshot→graph; reversible & monotone.
* **SEG:** edges are typed, temporal, contradiction-friendly.
* **VIF:** every node/edge carries {model, weights\_hash, prompt\_tpl, tools, snapshot}.
* **SDF-CVF:** changes across code/docs/tags/traces are **atomically recorded** in molecules and edges; parity gates live in APOE execution.
* **APOE:** orchestrators declare budgets and produce witnessed traces that reference node\_ids, never raw strings.

**4.8 What “good” looks like (targets)**

* **Indexing depth (IDS):** ≥ 6.0 for core domains (system→…→sub-word present; cross-links > 3 per atom).
* **Memory precision@10 @ ≤50 ms:** ≥ 0.85 on gold sets.
* **Snapshot replay fidelity:** 100% for readbacks with identical IDs.
* **PII false-negative rate:** ≤ 1e-4 (with redaction gates).
* **Dedup ratio:** ≥ 20% for large corpora (content-addressing at ingest).

**Outcome.** Chapter 4 nails the substrate: what we store, how we store it, why it’s reversible, and how it stays legible across time and modality.

**Part II — Context Memory Core (CMC) & HHNI**

**Chapter 5. Hyper-Hierarchical Neural Indexing (HHNI)**

**Intent.** Turn the Fractal Memory Hierarchy into a *neural-grade index-of-indexes* that spans system→section→paragraph→sentence→word→sub-word (and code/file analogs), with cryptographic dependency hashing, real-time impact previews, adaptive tag priority/decay, and a calibrated Retrieval Score .

**5.1 Indices from System → … → Sub-word**

**Principle.** Every level is both an addressable node *and* an index entry. HHNI guarantees: stable addressing, acyclic containment, time-sliced cross-links, and reversible materialization at any level.

**5.1.1 Level taxonomy (text & code analogs)**

| **Level** | **Text analog** | **Code analog** | **Audio/vision analog** |
| --- | --- | --- | --- |
| L0 System | corpus/app | monorepo | dataset |
| L1 Section | chapter/spec section | package/module | scene/album |
| L2 Paragraph | paragraph | file | shot/track |
| L3 Sentence | sentence | function/method | utterance/clip |
| L4 Word | token/word | identifier/token | phoneme/object box |
| L5 Sub-word | morpheme/BPE | subtoken | subphoneme/patch |

Containment edges contains(Ln, Ln+1) are **ordered** and **acyclic**; semantic edges (relates\_to, depends\_on, contradicts) are **temporal**.

**5.1.2 Stable addressing (IXPATH)**

Each node has a content address (node\_id) and a **hierarchical path**:

ixpath := system://{space}/{root}@{snap}/

sec[{i}]/para[{j}]/sent[{k}]/word[{m}]/sub[{bpe}]

* **Determinism:** IXPATH is computed from canonical segmentation and snapshot.
* **Idempotence:** Re-indexing the same snapshot yields identical IXPATHs.
* **Relocation safety:** A node’s node\_id is content-addressed; IXPATH is a *view*.

**5.1.3 Index record (IXREC)**

{

"ixpath": "system://a/b@snap:2025-10-17/.../sec[3]/para[5]/sent[2]/word[7]",

"node\_id": "sha256:…",

"level": "word",

"parent": "…/sent[2]",

"order": 7,

"span": { "char": [1340, 1349], "tok": [211, 211] },

"embedding": { "model": "…", "dim": 768, "vec": "…" },

"tags": [ { "name": "ner/ORG", "tpv": { "priority": 0.68, "relevance": 0.74, "decay": 0.02 } } ],

"links": [

{ "type": "relates\_to", "dst": "sha256:…", "weight": 0.81, "valid\_from": "…", "valid\_to": null }

],

"provenance": { "snapshot\_id": "snap:…", "tool": "segmenter@vX", "model\_hash": "sha256:…" }

}

**5.1.4 Canonical segmentation**

* **Text:** sentence boundary detection (punct+ML), word tokenization with **morphological split** (keeps lemma & morphemes).
* **Code:** AST-first: function/method (L3), statement (between L3 & L4 as needed), identifier/token (L4), subtoken (L5).
* **Audio/vision:** utterance/clip aligned to VAD/shot boundary (L3), phoneme/object (L4), subphoneme/patch (L5).
* **Overlap:** introduce 5–10% *semantic overlap* between adjacent units for recall.

**Guarantee.** Containment graph is acyclic and *bounded depth* (≤ 6). Cross-modal nodes may share IXPATH *prefixes* via alignment maps.

**5.2 Dependency Hashing & Impact Previews**

**Goal.** Make change and retrieval *safe by construction*: every node carries a cryptographic view of what it depends on; any mutation computes a bounded **Dependency Delta (DD)** and shows a *live* impact preview.

**5.2.1 Merklized dependencies**

For node , define its **dependency set** (parents, references, semantic links). Compute:

* **Localizing change:** a change in a dependent bubbles via Merkle root without rescanning the world.
* **Witnessability:** H(i) is logged in SEG; replays verify equality.

**5.2.2 Dependency Delta (DD)**

Let be normalized change magnitude for (e.g., edit distance, AST diff, or embedding drift). For an action centered at node :

* Paths are limited by radius (hop count or cumulative weight).
* controls attenuation with graph distance.

**Interpretation.** Probability-like accumulation of risk; ⇒ high blast radius.

**5.2.3 Impact preview (bounded BFS)**

function impact\_preview(root r, radius R, threshold τ):

Q ← [(r, 0)]; visited ← {r}; impacts ← []

while Q not empty:

(u, dist) ← pop(Q)

for each edge (u → v) with weight w:

if dist + 1 > R or v in visited: continue

score ← w \* γ^(dist+1) \* Δ(v)

if score ≥ τ: impacts.append((v, score))

visited.add(v); push(Q, (v, dist+1))

return sort\_desc(impacts)

* **UI contract:** show top-K impacts with edge paths and *what would change* (diff previews).
* **Write gate:** **C-1/C-2** require *impact preview recorded* before commit.

**5.2.4 Conflict classes**

* **Structural:** would introduce cycle in contains → hard fail.
* **Policy:** crossing into restricted region (PII/license) → route to redaction/HITL.
* **Semantic:** raises contradiction with higher priority claim → require verifier vote (APOE) or HITL.

**5.3 Priority & Decay (TPV dynamics)**

**Aim.** Keep indices *alive*: high-value tags rise; stale/rare tags fade; task-critical tags spike under mission profiles. TPV per tag on node :

* (priority): *exogenous* weight from current plan/policy.
* (relevance): *endogenous* weight from observed usefulness (clicks, retrieval hit, grounding success).
* (decay): time-based staleness.

**5.3.1 Online updates**

After an interaction with outcome signals :

* **Task spikes:** can be raised by APOE (“compliance sprint” ⇒ boost policy/\* tags).
* **Forgetting:** large increases , reducing RS unless refreshed.

**5.3.2 Tag budget and pruning**

* **Budget:** ≤ K tags per node (default K=8).
* **Pruning score per tag:** .
* **Policy floor:** tags with policy/critical never pruned; demote others below floor.

**5.3.3 Promotion & demotion events**

* **Promotion:** repeated high-quality retrievals (grounded, accepted) elevate ; crossing threshold adds **shortcut edges** (relates\_to) for faster reads.
* **Demotion:** prolonged non-use with high removes low- tags and trims edges (keeps lineage).

**5.4 Retrieval Score (RS) — normalization & calibration**

**Goal.** Make scores comparable across domains and over time; align RS with actual relevance; expose abstention thresholds .

**5.4.1 QS (quality score)**

* **Static checks:** schema conformance, language detect, PII scan, license safety.
* **Model checks:** coherence, redundancy, contradiction flags.
* **Provenance:** complete VIF tuple (model id, weights hash, prompt id, tools, snapshot).
* **Aggregation:** weighted harmonic mean emphasizing weakest link.

**5.4.2 IDS (indexing depth score)**

Let be levels present beneath a node, and be connection density in its subtree:

*Target:* core domains achieve and above the median.

**5.4.3 DD (dependency delta)**

From §5.2, bounded to [0,1]. **Interpretation:** “how risky is it to rely on this node now?”

**5.4.4 Normalization & calibration**

* **Per-domain normalization:** z-score or min-max per domain bucket (legal, code, design).
* **Calibration target:** .
* **Methods:** isotonic regression or Platt scaling on held-out gold sets; monitor **ECE** (expected calibration error).
* **Abstention bands:** choose s.t. false-positive rate on *high-risk domains* ≤ policy target; APOE abstains or escalates below .

**Training loop (pseudo):**

for epoch in 1..E:

D ← sample(gold\_pairs) // (query, node, y∈{0,1})

RS\_raw ← QS \* IDS \* (1 - DD)

RS\_norm ← normalize(RS\_raw, domain)

RS\_cal ← calibrate(RS\_norm) // isotonic fit

update(calibrator) to minimize ECE on validation

publish κ bands per domain: κ\_low, κ\_high

**5.5 Build & Serve Pipelines**

**5.5.1 Build (write-time)**

1. **Segment** payload into L1…L5; produce IXPATHs (idempotent).
2. **Atomize** each unit; compute embeddings (full + projection head).
3. **Tag** with initial TPVs (policy+NER+domain classifiers).
4. **Link** structural edges (contains) then semantic edges (KNN→rerank→policy filter).
5. **Merkle** dependencies; compute H(i); sign snapshot.
6. **Persist** IXRECs, edges, payload refs; emit VIF.

**Gate:** block if cycle, PII leak, license breach, or missing witness.

**5.5.2 Serve (read-time)**

1. **Coarse scan** at higher levels (L1–L3) using projection head.
2. **Dumbbell** expand to L4–L5 only for candidate branches.
3. **Score** candidates with calibrated RS; apply policy masks.
4. **Assemble** answer context with *boundary witnesses* (VIF) for APOE.

**5.6 Failure Modes & Countermeasures**

| **Failure** | **Symptom** | **Guardrail** |
| --- | --- | --- |
| Over-segmentation | Too many tiny atoms; recall drops | Minimum span, overlap window, merge heuristic |
| Index drift | IXPATHs unstable across runs | Freeze segmenter+tokenizer at snapshot; record tool hashes |
| Tag sprawl | Noisy/duplicated tags | Tag budget + pruning + policy floors |
| Hotspot collapse | DD spikes in a hub | Cap fan-in/out; shard edges; require verifier vote |
| Calibration rot | RS stops tracking relevance | Rolling recalibration + ECE alarms per domain |
| Privacy bleed | PII in embeddings | Vector access scopes; on-write redaction; audit jobs |

**5.7 APIs (minimal, authoritative)**

* POST /index/build: build HHNI for a given snapshot scope (system/section…).
* GET /index/resolve?ixpath=…&as\_of=…: fetch IXREC, parents, children.
* POST /index/relink: propose semantic links; returns filtered set + witness.
* POST /index/preview-impact: returns ranked impact set with paths and DD.
* GET /index/metrics: IDS distribution, tag budgets, calibration curves.

**AuthZ.** Read-mostly; write paths require **single-writer** token and policy approval.

**5.8 Invariants Crosswalk (this chapter)**

* **CMC:** atomize→index→snapshot→graph is enforced at every level; *reversibility* via acyclic contains + payload immutability.
* **SEG:** edges are typed, temporal, and **contradictions are preserved** (no destructive deletes).
* **VIF:** every IXREC/link carries model+weights+prompt+tools+snapshot; all hashes verifiable.
* **SDF-CVF:** changes across code/docs/tags/traces are reflected atomically through HHNI; impact previews gate merges.
* **APOE:** orchestrators consume HHNI via calibrated RS and policy masks; abstain under .

**5.9 “Done Right” Targets**

* **IDS median (core corpora):** ≥ 6.0; tail < 10% below 4.0.
* **Calibration (ECE):** ≤ 0.03 per domain bucket.
* **Impact preview latency:** p95 ≤ 50 ms for radius 3, degree ≤ 32.
* **Write reject rate (policy):** ≥ 99.9% for disallowed content (zero-tolerance classes).
* **Reindex determinism:** hash-match rate 100% on snapshot replays.

**Outcome.** HHNI gives us deterministic, cryptographically witnessed, *adaptive* indexing from system to sub-word—with change risk quantified up front and retrieval calibrated to reality.

**Part II — Context Memory Core (CMC) & HHNI**

**Chapter 6. Write/Read Pipelines**

**Intent.** Specify the **authoritative data paths** of AIMOS memory: how raw context becomes atoms, how links are proposed and risk-gated, how immutable snapshots are minted, and how reads assemble **small, lossless contexts** with **big, provable lineage**. This chapter operationalizes C-1/C-2/C-3 (single writer, snapshot-first, two-stage read) and exposes the performance envelope and SLOs.

**6.1 Write Path — *Ingest → Atomize → Propose → DD → Gate → Snapshot***

**Contract.** There is exactly **one transactional writer** per memory space. All visible state corresponds to a signed **snapshot id**.

**6.1.1 State machine**

NEW → (dedupe) → ATOMIZED → (index) → PROPOSED\_LINKS

→ (risk/PII/licensing) → GATED

→ (merklize+sign) → SNAPSHOTTED → PUBLISHED (SEG/VIF emitted)

↘ (any fail) → QUARANTINE

**6.1.2 Stages (authoritative definitions)**

1. **Ingest**

* Inputs: user prompts, code, files, tool outputs, logs, sensors.
* Actions: normalize encodings, strip control bytes, detect modality/language, compute content hash cid.
* Dedupe rule: if (cid, scope) already present in the *current* snapshot lineage, short-circuit to **PUBLISHED**; else continue.

1. **Atomize**

* Segment to L1…L5 (Chapter 5) and emit **IXPATH-stable** atoms/molecules.
* Extract **payload metadata** (author, source, timestamps, file path/AST path).
* Compute embeddings (full + projection head); attach initial TPV tags.

1. **Propose links**

* Structural: contains(parent, child) per segmentation → **must** be acyclic and ordered.
* Semantic: KNN candidates (top-M), rerank by cross-encoder/DomainRank, policy filter (namespace/PII/license), cap fan-in/out.
* Record candidate set with scores and **filtered-out justifications** (kept for audit).

1. **Compute risk**

* **Dependency Delta (DD)** from the proposed edit center (Chapter 5 §5.2).
* **UQ vector** for embeddings/links (variance across seeds/temps; optional ensembles).
* **Policy risks**: PII, license incompatibilities, safety classes.

1. **Gate**

* *Hard* gates: structural cycles, forbidden content, link into restricted region without exemption, DD above hard cap for that domain.
* *Soft* gates: DD in caution band, low calibration (ECE), ambiguous license.
  + Soft pass → *HITL* or **APOE verifier** vote; outcome logged as a witness.
  + Soft fail → quarantine with remediation hints (auto-fix candidates).

1. **Snapshot & Publish**

* Build **Merkle forest** over payloads and dependencies; compute H(root).
* Freeze tool versions, model ids, weight hashes, prompt templates.
* Sign snapshot, allocate snapshot\_id, write append-only to **Snapshot Log**.
* Emit **SEG** nodes/edges and **VIF** provenance packets; update read indexes.

**6.1.3 Transactional semantics**

* **Idempotency key:** (cid, scope, parent\_snapshot, toolset\_hash) ⇒ replay-safe.
* **Write isolation:** until **SNAPSHOTTED**, no read index mutation is visible.
* **Rollback:** any failure moves batch to **QUARANTINE** with a replay spec (exact inputs, seeds, tools).

**6.1.4 Minimal write API**

* POST /memory/write/ingest → {batch\_id}
* POST /memory/write/commit?batch\_id=… → {snapshot\_id, witness\_uri}
* GET /memory/write/quarantine/{batch\_id} → remediation report (policy hits, DD paths, suggested redactions)

**6.2 Read Path — *Hierarchical dumbbell, calibrated scoring, policy-aware assembly***

**Goal.** Serve **just-enough atoms** to maximize relevance and minimize risk/cost, while returning **witnessable** context for APOE.

**6.2.1 Query classes**

* **Direct lookup:** IXPATH or node\_id fetch at as\_of time.
* **Semantic search:** query vector + filters (modality, domain, time).
* **Lineage search:** claim/evidence traversal in SEG with constraints.
* **Programmatic read:** ACL step resolving inputs under budgets/policies.

**6.2.2 Dumbbell compression (two-stage read)**

1. **Coarse stage (L1–L3):**
   * Search **projection head** indexes to shortlist K branches (fast, low-dim).
   * Apply **policy prefilter** (redaction masks, license scopes, RBAC).
2. **Fine stage (L4–L5):**
   * Expand only shortlisted branches to word/sub-word or AST token level.
   * Recompute **RS = QS·IDS·(1−DD)** with calibrated normalizer.
   * Attach **boundary witnesses** (VIF tuples) and preview DD paths.
3. **Assembly:**
   * Compose returned context to favor *beginnings and ends* of spans (lost-in-the-middle mitigation), plus atomic snippets critical to the claim.
   * Include minimal **explanatory edges** (why this atom), not whole neighborhoods.

**6.2.3 Read algorithm (sketch)**

function resolve(query, as\_of, budget):

S ← select\_snapshot(as\_of)

B ← coarse\_scan(S, query, filters, K\_coarse) // L1–L3

C ← []

for branch in B:

C += fine\_expand(branch, K\_fine, max\_tokens=budget.ctx)

C ← score\_calibrated(C) // RS calibrated

C ← policy\_mask(C) // redaction/PII/license

if empty(C) or min\_RS(C) < κ\_domain:

return ABSTAIN(with reasons, alternatives)

return assemble(C, attach\_witness=true)

**6.2.4 Safety & abstention**

* **κ-bands per domain:** configurable; APOE uses abstain→route patterns.
* **Redaction priority:** redact over deny when possible; mark redaction in witness.
* **Leakproofing:** embeddings served under **scoped tokens**; no raw vector export by default.

**6.3 Caching & Preloading (Markov traversals)**

**Thesis.** Reads are **bursty and structured**. We exploit session patterns to predict next hops and stage atoms into fast caches without violating policy.

**6.3.1 Cache hierarchy**

* **L0 inline** (per-request): RS scoring artifacts, policy masks (TTL seconds).
* **L1 atom cache**: decrypted atoms/IXRECs, small objects (TTL minutes; TTI).
* **L2 vector cache**: projection head vectors and ANN cells.
* **L3 page cache**: cold payload pages and SEG shards.

**Coherency:** caches are **snapshot-scoped**; snapshot advance invalidates by **diff** (changed IXRECs/edges).

**6.3.2 Markov prefetch**

Let be states = IXPATH prefixes (e.g., sec[\*], para[\*]), and transition probabilities learned from anonymous session traces with policy filters.  
Prefetch top successors of current state maximizing:

* **Bandit modulation:** -greedy on prefetch candidates to explore.
* **Guardrails:** never prefetch atoms violating user scoping, PII policy, or license boundaries.

**6.3.3 Budget-aware preloading**

* **Per-user soft budget:** MB/sec and tokens/sec caps; backoff on contention.
* **Queue fairness:** WFQ across tenants; emergency drain for hot snapshots.
* **Eviction policy:** Weighted by (RS\_lift · recency · hit\_rate) / size.

**6.4 Performance Targets & SLOs**

**Why SLOs here:** Memory is the *foundation*. If writes or reads blow budgets, APOE chains drift and VIF becomes impractical.

**6.4.1 Latency & throughput (per single region)**

| **Operation** | **p50** | **p95** | **p99** | **Notes** |
| --- | --- | --- | --- | --- |
| Write commit (≤ 2MB batch) | 350 ms | 900 ms | 1.8 s | includes merkle+sign |
| Impact preview (R=3, deg≤32) | 8 ms | 35 ms | 60 ms | in-memory graph shard |
| Coarse scan (K=200) | 20 ms | 60 ms | 120 ms | projection head ANN |
| Fine expand (K=40, 2 branches) | 45 ms | 120 ms | 220 ms | includes policy mask |
| RS calibration pass | 2 ms | 6 ms | 10 ms | vectorized isotonic |
| Assemble+VIF attach | 6 ms | 20 ms | 40 ms | witness hydration |

**6.4.2 Correctness & quality**

* **Reindex determinism:** 100% match on replays (same snapshot/tools).
* **Calibration (ECE):** ≤ 0.03 per domain; alert at 0.05.
* **Policy leakage:** 0 P0 incidents; redaction success ≥ 99.9%.
* **Abstention quality:** κ-abstain true-positive ≥ 90% on high-risk sets.

**6.4.3 Cost & footprint**

* **Index build / GiB text:** target ≤ $X (tunable); ANN recall ≥ 0.95 at 10× speedup.
* **Cache hit ratio:** L1 ≥ 0.70 steady state; L2 ≥ 0.85.
* **Storage growth:** governed by snapshot delta, not wall-clock; auto-GC on expired retention windows (policy-aware).

**6.5 Failure Modes & Remedies**

| **Failure** | **Symptom** | **Root cause** | **Remedy** |
| --- | --- | --- | --- |
| Snapshot stall | commits hang | long merkle over big diffs | chunked merkle, parallel trees |
| Hot hub DD spikes | many edits blocked | hub fan-in/out too high | re-shard links; enforce degree caps |
| ANN drift | recall drops | embedding/model change | dual-index warmup; A/B calibrated swap |
| Prefetch waste | bandwidth spikes | poor Markov model | bandit exploration limits; policy filters upfront |
| Over-abstention | excessive denials | κ too conservative | recalibrate per domain; task-aware κ |

**6.6 Invariants Crosswalk (this chapter)**

* **CMC:** single writer, snapshot-first; all writes are idempotent and reversible via QUARANTINE/REPLAY.
* **SEG:** every publish emits time-sliced edges; contradictions preserved.
* **VIF:** witnesses attached at commit; reads return boundary witnesses and replay URIs.
* **SDF-CVF:** gating enforces parity; quarantines trigger auto-fix/ADR flows.
* **APOE:** read outputs are κ-aware, budget-bounded, and ready for deterministic chains.

**6.7 “Done Right” checklist**

* Write path enforces **hard gates** and records **soft-gate debates** (APOE/HITL).
* All reads are **as-of** a snapshot; no hidden mutable joins.
* RS is **calibrated** and monitored (ECE dashboards).
* Prefetch obeys **policy first**, not convenience.
* Every error path is **quarantine with replay**, never silent drop.

**Outcome.** The write/read pipelines make memory **boringly reliable**: deterministic snapshots, safe evolution, fast reads, and every returned byte comes with a **witness**.

**Part II — Context Memory Core (CMC) & HHNI**

**Chapter 7. Storage Substrate**

**Intent.** Specify the **four physical stores** that realize CMC: a high-performance **vector store**, a bitemporal **evidence graph**, a content-addressed **object store**, and an append-only **snapshot log**. All semantics in Chapters 4–6 depend on these stores being **snapshot-scoped, policy-aware, and replayable**.

**7.1 Vector Store & KNN**

**Role.** Fast, filterable nearest-neighbor search over embeddings for L1–L5 items (projection + full vectors), feeding read-time **coarse→fine** selection and write-time **link proposals**.

**7.1.1 Schema (columnar / key–value hybrid)**

* tenant\_id (str) • snapshot\_id (uuid) • node\_id (uuid) • ixpath (str; L1–L5)
* modality (enum: text, code, log, image\_meta, …) • namespace (str)
* emb\_proj (float[64]) — projection head for **coarse** passes
* emb\_full (float[D]) — full vector for **fine** passes (e.g., 768/1024)
* norm (float) • tpv\_priority (float in [0,1]) • ts\_valid\_from/ts\_valid\_to
* policy\_tags (set: pii, license:GPL, export:EAR, …)

**7.1.2 Indexing strategy**

* **Dual-index** per snapshot:
  + **ANN-coarse:** IVF-PQ / ScaNN / HNSW over emb\_proj (high recall, tiny memory).
  + **ANN-fine:** HNSW(ef) over emb\_full for rerank within shortlist.
* **Filter-first**: tenant/namespace/policy prefilters prune search space before ANN.
* **Background compaction:** periodic re-centroiding / graph rewire to limit drift.

**7.1.3 Writes**

* Occur only at **commit** (Chapter 6): batch insert (snapshot\_id, nodes); prior snapshots remain immutable.
* **Model change**: dual-write to *new\_index* while *old\_index* serves reads; hot-swap after calibration (A/B recall ≥ 0.95).

**7.1.4 Queries**

* search(q\_vec, filters, K\_coarse, K\_fine) → candidates with:  
  score = α·cosine + β·tpv\_priority + γ·recency – δ·policy\_penalty
* Returns **witness bits**: which filters applied; index version; calibration id.

**7.1.5 SLOs & guardrails**

* p95: coarse ≤ 60 ms, fine ≤ 120 ms (K\_coarse=200, K\_fine=40).
* Recall@K target ≥ 0.95 vs. exact on sampled audits.
* **Safety:** no raw vector egress without scoped capability; PII/Export filters are applied **before** ANN.

**7.1.6 Failure modes → remedies**

* **Index drift** (recall drop): run dual-index warmup + calibrated swap.
* **Hot hubs** (high fan-in/out): shard by namespace and cap degree in link proposal.
* **Fragmentation**: periodic rebuild per snapshot or tiered compaction.

**7.2 Evidence Graph Store (temporal)**

**Role.** The **substrate of truth** (SEG): time-sliced, typed nodes/edges encoding claims, sources, tool calls, derivations, contradictions, and decisions.

**7.2.1 Data model (bitemporal property graph)**

* **Nodes:** Claim, Source, Snippet, ToolCall, Decision, Agent, Snapshot.
  + Common props: id, snapshot\_id, jsonld, ts\_valid\_[from,to), uqlabels (risk, domain).
* **Edges:** supports, derives, contradicts, annotates, usesTool, approvedBy.
  + Props: weight, policy\_tags, ts\_valid\_[from,to), witness\_uri.

**Bitemporality.**

* *Valid time* = when the fact holds; *system time* = when we learned/recorded it.
* Queries accept as\_of\_valid, as\_of\_system for compliance-grade replay.

**7.2.2 Writes & lineage**

* Emitted at **publish** (Chapter 6): edges/nodes carry **VIF tuples**  
  (model\_id, weights\_hash, prompt\_template\_id, tool\_fingerprint, snapshot\_id, UQ\_vector).
* **Contradictions are edges**, not deletes; prior facts remain queryable with their time bounds.

**7.2.3 Indexes & algorithms**

* Composite: (label, snapshot\_id, ts\_valid\_from); adjacency on (out\_v, edge\_type).
* Path queries: constrained BFS/Dijkstra with **policy-aware pruning**.
* **Subgraph signatures:** Merkle over node/edge ids → quick integrity checks.

**7.2.4 APIs (selected)**

* GET /seg/path?from=claim&to=decision&as\_of=...&max\_hops=...
* POST /seg/annotate (adds annotates edges with reviewer witness)
* GET /seg/pack/export?scope=case-123 (regulatory bundle)

**7.2.5 SLOs & guardrails**

* p95 path (≤ 5 hops): ≤ 80 ms; p95 neighborhood (≤ 200 edges): ≤ 120 ms.
* **Policy:** time-slice applies to *both* nodes and edges; redaction marks are edges with policy:redacted.

**7.3 Object Store (content-addressed)**

**Role.** Immutable storage for raw payloads and heavy artifacts (docs, code blobs, images), with **dedupe, delta, and encryption**.

**7.3.1 CAS design**

* cid = H(normalize(payload)) (blake3/sha256).
* **Chunking:** content-defined (rolling hash) to maximize dedupe across revisions.
* **Delta chains:** binary diffs for large versions; capped chain length (periodic re-base).

**7.3.2 Metadata & lines**

* mime, charset, lang, license, pii\_tags, source\_uri, ingest\_tool.
* **Line/AST maps** to reconstruct IXPATH spans; stable anchors survive minor edits.

**7.3.3 Access model**

* Read via **pre-signed URLs** scoped to tenant, snapshot, and IXPATH range.
* **At-rest encryption** with tenant keys; server-side KMS rotation.
* **Bandwidth guards:** per-tenant egress quotas; LRU/CDN in front.

**7.3.4 Lifecycle & compliance**

* Retention policies per class (e.g., 6 mo for high-risk logs unless extended).
* **Right-to-erasure:** write a **redaction overlay snapshot** (do not mutate old); scrub ANN cells and caches via tombstone propagation.

**7.4 Snapshot Log & Rollback**

**Role.** Append-only ledger of memory states; the **single source of replay truth** for writes, reads, audits, and incident response.

**7.4.1 Record format**

* snapshot\_id (uuid) • parent\_id (uuid or null) • merkle\_root • diff\_summary
* signer (key id, sig) • toolset\_hash • created\_at • notes (ADR link)
* **State refs:** vector index version ids, SEG batch id, object batch manifest.

**7.4.2 Semantics**

* **Linear or branched** histories allowed (feature work, quarantines).
* **Rollback** = select **earlier snapshot** as as\_of; no mutation retroactively.
* **Quarantine** references point to remediation plans (auto-fix, HITL).

**7.4.3 APIs**

* GET /snapshots?cursor=... (paged)
* POST /snapshots/label (e.g., *prod-2025-10-17*)
* POST /snapshots/replay?from=...&to=... (reproduce a write path with frozen tools)

**7.4.4 DR & GC**

* **Cross-region replication** (RPO ≤ 1 min; RTO ≤ 15 min).
* **GC** prunes expired data by policy while preserving graph integrity (keep tombstones & redaction overlays).
* **Audit immutability:** WORM buckets for the log itself.

**7.5 Security & tenancy (applies to all stores)**

* **Strong tenancy keys**: every index row/edge/object scoped by tenant\_id and snapshot\_id.
* **Capability tokens** for ANN, SEG, CAS; least-privilege scopes (read-only vs write).
* **Row-level policy filters** evaluated **before** search/graph traversal.
* **Signing & attestation**: all commits signed; verifier can check toolchain SBOMs.
* **PII & export control**: tag propagation into ANN and SEG; deny or redact at source.

**7.6 Invariants crosswalk (this chapter)**

* **CMC:** snapshot-scoped writes/reads; single writer realized via snapshot log.
* **SEG:** bitemporal graph; contradictions and redactions are **edges**, not deletes.
* **VIF:** every store returns verifiable **witness URIs** and signatures.
* **SDF-CVF:** content-addressed CAS + graph ensure doc/code/tag/trace parity.
* **APOE:** deterministic replay from any snapshot\_id under fixed tools/budgets.

**7.7 “Done Right” checklist**

* Dual-index ANN with **filter-first** and calibrated swap.
* SEG is **bitemporal**; exports produce legally defensible bundles.
* CAS uses content-defined chunking, delta chains, and KMS-backed keys.
* Snapshot log is **append-only**, signed, and cross-region replicated.
* Redaction & erasure implemented via **overlay snapshots** + index scrubbing.
* All APIs accept as\_of and return **witness metadata**.

**Part III — Dynamic Vector Navigation System (DVNS)**

**Chapter 8. Physics-Inspired Retrieval**

**Intent.** Replace brittle, purely lexical/nearest-neighbor retrieval with a **physics-guided search** that respects memory priority, policy constraints, and auditability. DVNS runs *on top* of the CMC substrates (vector store, SEG, CAS, snapshot log), turning a query into a **stable, policy-safe path** through memory.

**8.1 GODN Recap → Retrieval Mapping (grav, elastic, repulse, damping)**

**State space.**  
Let the working set be the union of:

* **Candidates** from the vector store after filter-first coarse KNN and fine rerank,
* **Anchors** from SEG (e.g., authoritative sources, recent decisions, pinned artifacts),
* **Barriers** implied by policy (PII, license, export tags) and time slicing (as-of).

Each item has:

* **Position** (embedding),
* **Mass** (priority; e.g., TPV from Chap. 5),
* **Charge** policy mask (1=allowed),
* **Validity** interval (bitemporal).

We simulate a small number of integration steps to refine a **path** from the query to a compact set of witnesses.

**Forces (GODN → retrieval semantics).**

* **Gravitational (attraction / semantic pull).**

Pulls toward semantically similar, high-priority neighbors.

* **Elastic (structure / hierarchy / policy keeping).**  
  For intended structural links (e.g., same section, same molecule, approved SEG edges),

Keeps **hierarchical cohesion** (IXPATH proximity) and **SEG-endorsed** neighborhoods.

* **Repulsive (redundancy / cluster de-dup).**  
  If :

Spreads near-duplicates, promoting **diverse context**.

* **Damping (stability / early stopping).**

Eliminates oscillation; acts like **temperature** control for convergence.

**Policy geometry (hard constraints).**

* If (forbidden) or item invalid at the as\_of slice, project updates with a **hard barrier**:

where is projection onto the **admissible region** (policy-safe, time-valid).

**8.2 Path Cost : length, smoothness, blocks, density**

We score candidate **explanatory paths** through the refined neighborhood:

* **Length (semantic travel):** , with .
* **Smoothness (topic curvature):** .
* **Blocks (policy/temporal/SEG violations):** (large penalty).
* **Density (over-crowding / redundancy):** .

**Total cost (minimize):**

**Retrieval score (with Part II RS):**

where **RS** is the base and is path cost after DVNS refinement.

**Interpretation.** Short, smooth, policy-clean, and non-redundant paths **win**. This turns “top-k neighbors” into a **coherent, auditable route** from query to witnesses.

**8.3 Stability & Abstention**

**Integration loop (per query).**

1. Build : coarse→fine KNN with **filter-first** (tenant, namespace, policy, time).
2. Initialize . For :
   * Compute .
   * Update , (semi-implicit Euler or Verlet).
   * Project to admissible .
   * Check **energy drop** (converged) or **max steps** reached.
3. Enumerate **simple paths** through refined graph (beam on SEG-endorsed edges + local neighbors).
4. Score with → take argmin; compute and **uncertainty vector** .

**Abstention rule (κ-gating).**

* If **or** any component of exceeds bound, **abstain**:
  + Fall back to: (a) classic KNN slice + raw citations, (b) ask-clarify, or (c) route to HITL (risk-aware).
* Emit **boundary witness**: reason for abstention, thresholds, top conflicting pulls.

**Stability knobs.**

* **Damping** up when oscillation detected (energy up-ticks).
* **Repulsion** up when diversity low (near-dup density).
* **Elastic** up when hierarchy/SEG cohesion matters (e.g., code+tests).

**8.4 Policy-Aware Geometry (forbidden crossings)**

**Forbidden crossings.**

* A candidate edge is illegal if traversing it would:
  1. cross into a policy-forbidden region (PII, export),
  2. break a **time slice** (node or edge not valid at as\_of),
  3. contradict declared **role** (e.g., verifier may not mutate memory),
  4. violate **SEG assertions** (e.g., explicit *contradicts* without a resolving decision).

We treat such edges as **infinite barriers** in scoring (i.e., large) and as **hard projections** during integration. The path builder prunes them before enumeration.

**Projection operator.**  
For an update , if :

Implement as **masked coordinates** (drop disallowed facets) or **clipped kernels** in ANN rerank.

**SEG-first search.**  
When a path must justify a claim, prefer **SEG-endorsed** hops (supports/derives) and treat free semantic hops as secondary with a **smoothness penalty** unless bridged by acceptable anchors (e.g., tool outputs with VIF).

**DVNS: Minimal Execution Sketch**

**Inputs.** query text , as\_of, filter set , policy map .  
**Outputs.** ranked .

1 q\_vec ← embed(q); anchors ← SEG.anchors(as\_of, F)

2 cand ← ANN.search(q\_vec, filters=F, Kc→Kf) # filter-first (tenant, policy, time)

3 W ← prune(cand ∪ anchors, policy=Π, timeslice=as\_of)

4 init velocities v\_i = 0; t = 0

5 while t < T and ΔE > ε:

6 for i in W:

7 F\_i ← sum\_grav(i,W) + sum\_elas(i,SEG) + sum\_rep(i,W) − c v\_i

8 x\_i, v\_i ← integrate(F\_i) ; x\_i ← project(x\_i, A) # admissible region

9 ΔE ← energy\_drop()

10 t ← t+1

11 G' ← localized graph over W (SEG edges + nearest safe neighbors)

12 paths ← enumerate\_simple\_paths(G', src=q, beam=B, max\_len=L)

13 score each P with C(P); compute RS\*, UQ u; filter P by κ, u

14 return top-N (P, RS\*, u, path\_witnesses, index\_versions, snapshot\_id)

**Witnessing.**  
Return **exact index versions**, snapshot\_id, UQ vector , path features , and **citations** (SEG node ids, CAS CIDs). This fulfills **VIF** and **SEG** invariants.

**Design Notes & Interfaces**

* **Pluggable kernels.** Gravitational pull can be or learned bilinear forms; keep **monotonic** in similarity.
* **Budget awareness.** Limit , , beam width under APOE step budgets; expose **cost counters**.
* **Adaptive priors.** Use TPV (priority, decay) to set masses ; recent, frequently validated atoms pull harder.
* **Multimodal.** For image/code/log nodes, operate in **shared latent** (aligned encoders) or run **per-modality sub-loops** with late fusion.
* **Safety hooks.** Before emit, run **policy lints** (no forbidden terms, no leaking PII) and **abstain** if violated.

**What DVNS Guarantees (relative to plain KNN)**

* **Coherence** (smooth paths) over **scatter** (unrelated top-k).
* **Diversity** (repulsion) over **echoes** (near-dup clumps).
* **Compliance by construction** (policy geometry) over **post-hoc filtering**.
* **Audit trails** (witnessed paths) over **opaque neighbor lists**.

**Part III — Dynamic Vector Navigation System (DVNS)**

**Chapter 9. Algorithms & Analysis**

**Intent.** Nail down the numerics, show the run-time costs, and prove DVNS is practical at scale. We focus on four things: **(9.1)** force integration choices, **(9.2)** convergence & complexity, **(9.3)** super-index precomputation, **(9.4)** empirical lift vs. static KNN.

**9.1 Force Integration (Verlet vs. Euler)**

**State.** For each working item : position , velocity , mass . Forces:

with policy/time **projection** .

**A. Semi-implicit (symplectic) Euler.**

* Pros: 1 line, cheap, symplectic ⇒ decent stability for small .
* Cons: More numerical damping than Verlet; slightly less accurate per step.

**B. Velocity Verlet.**  
Compute from ; then

* Pros: Second-order, better energy behavior, fewer steps to converge.
* Cons: Slightly higher per-step compute (needs force eval twice).

**Recommendation.** Use **velocity Verlet** by default; automatically **downgrade to symplectic Euler** if you detect tight latency budgets or extremely small .

**Adaptive step & friction.**

* (shrink when energy increases).
* (raise damping when you detect reversals).

**Vectorized batch form (GPU-friendly).**

* Maintain .
* Precompute **neighbor lists** per item (top-r semantic + SEG edges); compute forces via sparse gather/scatter.
* Apply **mask matrices** for policy/time; projection as masked coordinates or clamped kernels.

**Default numeric profile (safe starting point).**

* ,
* Gravitational ; Elastic , rest-length
* Repulsion radius , strength
* Damping (raise to 0.25 if oscillation detected)
* Steps for QA; for code/log forensics

**9.2 Convergence & Complexity**

**Energy functional.**

with active only for . Damping yields almost everywhere (minus projection). We **stop** when

* **(S1)** for two consecutive steps, or
* **(S2)** max steps reached, or
* **(S3)** abstention triggers (policy barriers dominate or UQ spikes).

**Oscillation detection.** Track sign changes in per item; if median flips > threshold, raise and shrink .

**Time complexity (per query).**

* ANN search (coarse→fine): expected with modern indexes + re-rank .
* Build (union of candidates + anchors) ⇒ –.
* Force pass (sparse neighbors per item): per step.
* Steps : total . With , , viable at ms-scale on GPU.
* Path enumeration (beam , max length ): . Use SEG-first beam to keep small.

**Space complexity.**  
Store and sparse neighbor lists: . Witness buffers add .

**Determinism knobs.**

* Fix seeds for ANN and tie-breaks.
* Quantize grid and clamp floating epsilons.
* Emit **determinism flag** if any projection resolved ties non-deterministically (rare).

**9.3 Super-Index Precomputation**

Goal: cut per-query work by precomputing **basins** and **path skeletons** at the **super-index** layer.

**Artifacts.**

1. **Attractor basins.** Cluster the super-index with community detection over the SEG-endorsed graph + semantic edges. Precompute:
   * **Centroids** (mean embedding), **mass** (priority sum), **spanners** (few edges connecting clusters).
2. **Skeleton paths.** For each centroid, cache top **policy-clean** path templates to canonical witnesses (standards, ADRs, specs).
3. **Shard by time & policy.** Maintain bitemporal shards (e.g., monthly) and policy partitions; invalidate/write-through on updates.

**Runtime use.**

* Route query to nearest centroids .
* Seed with **basin members** and **skeleton witnesses** (fast start).
* Clip DVNS steps aggressively (often ) because you start “near” a solution.

**Maintenance.**

* **Delta maintenance:** Update basins incrementally as new atoms arrive; re-run full partition nightly.
* **A/B-cache policy:** Keep latest and previous shard live for snapshot replays.

**Expected win.** 20–60% reduction in per-query steps; more stable paths; better cold-start for rare queries.

**9.4 Empirical Lift vs. Static KNN**

**Evaluation protocol.**

* Corpora: (i) multi-doc QA, (ii) codebase Q&A/refactor, (iii) log forensics.
* Gold: top-k **witness sets** with human adjudication; time-slice enforced.
* Systems: **KNN-only**, **KNN+rerank**, **DVNS**, **DVNS+super-index**.
* Metrics:
  + **Retrieval:** p@k, nDCG, **RS-lift** (Δ against KNN), de-dup@k (fraction unique sources).
  + **Compliance:** % policy-clean paths, forbidden-edge rate.
  + **Audit:** lineage completeness (% artifacts with full VIF tuple), replay success.
  + **Cost:** median latency, 95p latency, token/compute budget adherence.

**Ablations (turn off each force).**

* −grav ⇒ worse recall (semantic pull lost).
* −elas ⇒ cohesion breaks (sections split), path smoothness drops.
* −rep ⇒ redundancy spikes (near-dup piles).
* −damp ⇒ instability tails, higher abstentions.

**Typical outcomes (targets, not promises).**

* **p@5:** +4–10% vs. KNN; **nDCG:** +5–12%.
* **De-dup@10:** +15–30%.
* **Policy-clean path rate:** >99.5% (vs. 97–98% with post-hoc filters).
* **Latency impact:** +10–30% vs KNN+rerank (mitigated to <10% with super-index).
* **Abstention precision:** >0.8 (abstain when truly under-supported).

**Failure modes & mitigations.**

* **Oscillation in dense clusters:** raise , shrink , cap neighbor degree.
* **Policy walls choke paths:** broaden anchor set; add approved bridge nodes; or **abstain**.
* **Novel queries (no anchors):** fall back to KNN slice with explicit low-confidence witness.
* **Over-repulsion (loss of signal):** lower , increase (neighbors).

**Tuning cheatsheet (per domain).**

* **Ad-hoc QA:**
* **Code reasoning:** (favor cohesion)
* **Log forensics:** (favor diversity)

**What this chapter gives you**

* A **numerically stable** recipe (Velocity Verlet + adaptive step) that runs fast on GPUs.
* Clear **stop rules**, **oscillation detectors**, and **abstention gates**.
* An offline **super-index** that slashes online work and improves determinism.
* An **evaluation harness** to prove lift over static KNN and catch regressions.

**Part III — Dynamic Vector Navigation System (DVNS)**

**Chapter 10. Implementation Patterns**

**Intent.** Turn DVNS from equations into a production recipe. Four pieces: **(10.1)** online refinement on top-k, **(10.2)** parameter tuning at runtime, **(10.3)** failure modes & mitigations, **(10.4)** visualization & telemetry.

**10.1 Online Refinement on Top-k**

**High-level loop**

1. **Coarse retrieve.** ANN over embeddings → top-k candidates (e.g., 512).
2. **Policy & time filter.** Apply SEG constraints (time-slice, allow-lists).
3. **Form working set .** Keep k = 64–256 items + **anchors** (canonical witnesses, recent snapshots).
4. **Local DVNS refinement.** Physics pass (Velocity Verlet; §9.1) with sparse neighbors (8–16).
5. **Path find.** Beam search (B=3–8) on refined graph; respect policy geometry (no forbidden crossings).
6. **Score & gate.** Compute RS, UQ; κ-gate.
7. **Emit witnesses.** Claim→Evidence→Decision lineage (VIF).

**Pseudocode (single query )**

cand = ANN.search(q, k0)

cand = policy\_filter(cand, seg\_rules, time\_slice)

W = select\_working\_set(cand, anchors, k1)

X,V = init\_positions\_velocities(W) # from embeddings; V=0

Nbr = build\_sparse\_neighbors(W, r) # semantic + SEG edges

for t in 1..T:

F = forces(X, V, W, Nbr, params) # grav + elastic + repulse - c\*V

V = V + Δt \* (F / m)

X = X + Δt \* V + 0.5 \* (Δt^2) \* (F / m) # velocity Verlet

X = project\_policy(X, policy\_regions) # forbid crossings

if stop\_rule(ΔE, t): break

G' = induce\_graph(W, X, policy\_ok=true)

paths = beam\_paths(G', sources=anchors, B, Lmax)

score = rank(paths, RS, UQ, policy\_clean)

if κ\_gate(score): return ABSTAIN(uq, reasons)

return best(paths), witnesses(paths)

**Engineering notes**

* Use **fixed seeds** for ANN tie-breaks to improve determinism.
* Pre-warm **anchor cache** by domain (docs, code, logs).
* **Early exit** if RS saturates and UQ is tight before T steps.

**10.2 Parameter Tuning (G, k, δ, c, α, β, γ)**

**What the knobs do**

* **G** (gravity): semantic pull; ↑G ⇒ tighter recall, risk of redundancy.
* **k** (elastic): cohesion along trusted edges; ↑k ⇒ smoother, can over-fit clusters.
* **σ, ρ** (repulsion strength/radius): de-dupe pressure; ↑σ/ρ ⇒ diversity, risk of dropping signal.
* **c** (damping): stability; ↑c ⇒ fewer oscillations, slower convergence.
* **Δt** (step): speed/accuracy trade.
* **α, β, γ** (path cost weights): smoothness/blocked/density penalties in .

**Domain presets (starting values)**

| **Domain** | **G** | **k** | **σ** | **ρ** | **c** | **Δt** | **T** | **r** | **α/β/γ** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ad-hoc QA | 1.0 | 0.4 | 0.35 | 0.18 | 0.12 | 0.25 | 8 | 12 | .4/.2/.2 |
| Code reasoning | 1.2 | 0.8 | 0.25 | 0.16 | 0.18 | 0.20 | 12 | 16 | .5/.2/.1 |
| Log forensics | 0.9 | 0.6 | 0.45 | 0.20 | 0.20 | 0.20 | 10 | 20 | .3/.3/.3 |

**Auto-tuning loop (bandit over safe ranges)**

* Contextual bandit (UCB/Thompson) over tuples with reward  
  .
* Constrain drift: max 1 param change (±10%) per deployment window.
* Persist **per-tenant profiles**; decay to global defaults if sparse.

**Runtime adaptors**

* If **UQ↑** (wider uncertainty) → add anchors, ↑k slightly, ↓σ.
* If **duplicative hits** → ↑σ or ρ, ↑r a notch.
* If **oscillation** → ↑c (×1.5), ↓Δt (×0.75), shorten T.

**Config example (YAML)**

dvns:

domain: "code"

params:

G: 1.2

k: 0.8

sigma: 0.25

rho: 0.16

c: 0.18

dt: 0.20

steps\_max: 12

neighbors: 16

path\_cost:

alpha: 0.5

beta: 0.2

gamma: 0.1

anchors:

- "spec://adr/001"

- "repo://main/README"

**10.3 Failure Modes (and Fix Kits)**

| **Symptom** | **Likely Cause** | **Fast Mitigation** | **Permanent Fix** |
| --- | --- | --- | --- |
| **Oscillation** (positions bounce) | Δt too high; c too low; dense neighbors | ↓Δt; ↑c; cap r | Adaptive damping; neighbor pruning policy |
| **Collapse** (everything clumps) | σ/ρ too low; G too high | ↑σ, ↑ρ; ↓G | Diversity guard: min-distance constraint |
| **Attractor lock** (stuck in wrong cluster) | Bias from anchors; k too high | add alternative anchors; ↓k | Super-index multi-basin seeding (§9.3) |
| **Policy wall** (no feasible path) | Forbidden edges; time slice too strict | broaden allowed bridge set; widen slice | SEG “bridge lists” with audit tags |
| **Sparse recall** (low p@k) | r too small; σ too high | ↑r; ↓σ; ↑T by +2 | Basin-aware neighbor sampler |
| **Redundancy** (near-dup results) | ρ too small; σ too low | ↑ρ; ↑σ | Dedup at readout with hash-LSH |
| **Latency spikes** | r,T too high; ANN retries | lower r/T; enable early exit | Super-index seeding; ANN cache |

**Abstention best-practice (κ)**

* κ triggers when **(i)** RS < τ & UQ > τ, **(ii)** policy-clean path not found within L, **(iii)** witness set below min cardinality.
* Emit: reason codes, UQ bands, top three **next actions** (ask for time-slice, add source, or relax scope).

**10.4 Visualization & Telemetry**

**Spans & events (OpenTelemetry)**

* **Spans**: ann.search, seg.filter, dvns.integrate, dvns.pathfind, gate.kappa, witness.emit.
* **Attributes**: k, k, r, T, ΔE trajectory, param snapshot {G,k,σ,ρ,c,Δt}, κ result, RS, UQ, policy violations.
* **Events**: oscillation\_detected, collapse\_prevented, policy\_wall, abstain.

**Metrics**

* **Quality**: p@k, nDCG, RS-lift vs KNN baseline.
* **Safety**: policy-clean %, κ precision/recall, lineage completeness %.
* **Cost**: median/95p latency; GPU-ms per query; tokens (if LLM rerankers used).
* **Stability**: oscillation rate; convergence steps; early-exit rate.

**Dashboards (tiles)**

1. **Flow Strip**: stacked spans per query with latency breakdown.
2. **Energy Curve**: vs. t; annotate mitigations.
3. **Param Drift**: time-series of G,k,σ,c per tenant/domain.
4. **κ Outcomes**: abstentions by reason; post-hoc human acceptance %.
5. **Policy Map**: violations by rule; before/after DVNS projection.

**Witness panel (per result)**

* Claim, RS, UQ bands; path with policy checks; VIF tuple *(model id, weights hash, prompt template id, tools, snapshot id)*.
* **Replay** button: executes on frozen snapshot; diff shown if any non-determinism flag was raised.

**Logging schema (JSON-LD excerpt)**

{

"@type": "DVNSRun",

"queryId": "q-...-001",

"params": {"G":1.2,"k":0.8,"sigma":0.25,"rho":0.16,"c":0.18,"dt":0.2,"T":12,"r":16},

"spans": [{"name":"dvns.integrate","took\_ms":7.1,"ΔE":[-0.22,-0.08,-0.03]}],

"score": {"RS":0.87,"UQ":{"ece":0.06,"entropy":0.21}},

"policy": {"violations":0,"forbidden\_crossings":0},

"witness": {

"traceId":"t-...-abc",

"vif":{"model":"apoe-hyena-64k","weights\_sha":"...","prompt\_tpl":"ptl-42","tools":["seg.query@1.2"],"snapshot":"snap-2025-10-01"},

"path":["seg://adr/001","doc://spec#sec3","code://module/auth.ts"]

}

}

**What you can ship today**

* **A deterministic, GPU-friendly DVNS runtime** that starts from top-k, refines locally, and emits auditable paths.
* **Auto-tuning guardrails** that adapt parameters safely per domain/tenant.
* **Failure-mode playbooks** with one-click mitigations (lower Δt, raise c, add anchors).
* **First-class observability**: spans, metrics, and a witness panel that satisfies VIF.

**Part IV — Orchestration: APOE & DEPP**

**Chapter 11. Cognitive Compilation (APOE)**

**Purpose.** Convert messy intent into **typed, budgeted, and auditable plans** that execute deterministically on top of CMC/HHNI and emit VIF-grade witnesses. APOE is the compiler; DEPP (Ch.12) is the large-scale planning fabric it feeds.

**11.0 Guarantees (what APOE must uphold)**

1. **Reproducibility.** Given fixed (snapshot), tool versions, budgets, and seeds ⇒ identical trace within stochastic tolerance; else **abstain** ().
2. **Lineage-first.** Every boundary produces a witness tuple (model, weights-hash, prompt-tpl, tools, snapshot-id, UQ).
3. **Budget honesty.** Declared token/time/tool budgets bound execution; overruns fail closed with a reason code.
4. **Policy-clean.** No path may cross forbidden regions (SEG/guardrails) even if quality scores tempt it.

**11.1 Role Model & Contracts**

Each role is a **typed step** with strict I/O and a witness envelope. Roles may be chained, branched, or run in parallel.

| **Role** | **Inputs** | **Outputs (Artifacts)** | **Witness (must include)** | **Quality KPI** |
| --- | --- | --- | --- | --- |
| **Planner** | goal, constraints, snapshot-id | plan DAG, budgets, role bindings | model+hash, prompt-tpl, seed, RS of retrieved reqs | plan validity %, budget fit |
| **Retriever** | query, HHNI scope, policy slice | top-k set, evidence paths | ANN index id, params, κ status, RS & UQ | p@k, RS-lift |
| **Reasoner** | facts, rules, prompts | structured claims, derivations | chain-of-thought summary\*, UQ bands, seeds | coherence score, ECE |
| **Verifier** | claims, evidence | acceptance/reject, deltas | tests run, criteria, thresholds | false-accept/false-reject |
| **Builder** | spec, scaffolds | code, configs, migrations | toolchain hashes, diffs, compile logs | build success, parity |
| **Critic** | artifact, rubric | review notes, risk score | rubric id, bias checks | defect density |
| **Operator** | plan, env | execution spans, costs | tool usage, tokens, time | SLO conformance |
| **Witness** | all above | signed log/JSON-LD | snapshot-id, time slice, lineage | completeness % |

\*The Reasoner’s internal chain needn’t be fully exposed; but it must emit **machine-readable summaries** sufficient for replay & audit.

**Contract sketch (per step, abstract type)**

step S<T\_in, T\_out>:

inputs: T\_in

tools: ToolCaps[]

budget: { tokens, wall\_ms, calls }

effect: { read(M\_t), read(G), write(trace) }

returns: { artifact: T\_out, witness: VIF, uq: UQ }

gates: Gate[] # pass/fail with reasons

**11.2 Plans as Typed DAGs (the APOE “IR”)**

* **Plan** is a DAG where are steps with signatures; edges carry **typed channels** (schemas).
* **Edges** annotate **scope** (HHNI indices, time-slice), **policy region**, and **budget pass-throughs**.
* **Seeds** (PRNG) and **tool pins** (version/weights hash) live in the plan header.

**Determinism discipline**

* Pin: model id, weights SHA, prompt template id, tool version, ANN index snapshot id.
* Fix: top-k tie-breakers by **stable sort key** (doc-id, then shard-id).
* Bound: sampling via fixed seeds and temperature schedules; otherwise if nondet > tolerance.

**11.3 Budgeting & Scheduling**

**Budget classes**

* **Tokens:** prompt+completion tokens per step and plan total.
* **Time:** wall-clock per step and plan total.
* **Calls:** tool/API invocations (capability tokens).
* **Cost:** estimated $ or GPU-ms envelope (operator view).

**Preflight**

* Static estimate using **emb len**, **fan-out**, **avg completion**, **tool latency histograms**.
* If estimate breaches ceiling ⇒ planner rewrites with smaller fan-out / deeper caches / cheaper models.

**Runtime adaptation**

* **Soft throttle:** shrink fan-out or shorten completion length if budget burn rate > slope.
* **Grace step:** optional one-shot overrun for **witness-critical** boundaries (logs a reason).
* **Fail-closed:** emit κ-abstain + next-best actions.

**11.4 & Uncertainty Emission (boundary behavior)**

**UQ vector (per boundary)**

* **Model**: entropy / variance (ensembles or temp sweep)
* **Retrieval**: RS dispersion (top-k margin, path consensus)
* **Calibration**: ECE (expected calibration error) bucket
* **Policy**: distance to forbidden region

**Decision**

* If **and** ⇒ **abstain**.
* Output: reason codes, suggested “next questions”, and minimal witness to replay context.

**11.5 Routing, Parallelization, Retry**

**Router** (lightweight classifier): routes intents to pipelines & chooses **model tiers** (fast vs. accurate), **tool stacks**, and **policy slices**.

**Parallel patterns**

* **Map → Verify → Reduce:** fan-out retrievers/reasoners; verifier collapses with majority/weighted vote.
* **Debate:** 2–3 reasoners argue; verifier adjudicates using rubric.
* **Spec-and-Solve:** planner spawns spec writer and solver; critic cross-checks parity.

**Merge operators**

* **Max-RS** (pick path with highest RS under policy-clean).
* **Borda** / **Weighted majority** (use UQ as weight).
* **Pareto** (no dominated solution on RS, cost, policy distance).

**Retry policy**

* **Idempotent** steps (retriever) may retry with **anchored perturbations** (extra anchors, widened slice).
* **Non-idempotent** (builder) require human or new snapshot/branch.

**11.6 Boundary Gates (quality & safety)**

Each step ends with a **gate bundle**:

* **Schema gate:** inputs/outputs match contract (JSON Schema / SHACL).
* **Policy gate:** SEG constraints pass; no forbidden crossing.
* **UQ gate:** ECE within tolerance; entropy below threshold.
* **Provenance gate:** VIF tuple complete; hashes present.
* **Diff gate (build/test):** parity spec↔code; tests ≥ coverage floor.
* **HITL gate (risked):** two-key approval if risk score ≥ .

Failure ⇒ **auto-fix** (if known), else **quarantine branch** with escalations.

**11.7 Failure Taxonomy & Recovery**

| **Class** | **Example** | **System Reaction** |
| --- | --- | --- |
| **Constraint** | budget exceeded; policy wall | κ-abstain; planner suggests scope/budget edits |
| **Data** | missing anchors; low RS | expand anchors; widen time-slice; run DVNS with diversity boost |
| **Stochastic** | nondet beyond tolerance | reseed & replay; if persists, flag tool/version drift |
| **Spec drift** | code passes tests but violates spec | block merge (SDF-CVF); open auto-fix PR; require HITL |
| **Tool failure** | API 5xx, timeout | circuit breaker; backoff; alternate tool path |

**11.8 Security Posture (execution hygiene)**

* **Capability tokens** per tool with least privilege & time-boxed scopes.
* **Sandboxed tools** (network egress allow-list; fs paths jailed).
* **Prompt hygiene**: structured templates; strip & neutralize tool-returns; literal-block untrusted text.
* **Data minimization**: redact PII; pass **handles** (ids) not raw data when possible.
* **Signed snapshots**: snapshot id + Merkle root over inputs → witness includes signature.

**11.9 Minimal Runnable Example (plan IR)**

**Goal:** “Summarize policy doc and cite sources with time slice.”

plan "policy-sum-v1" {

header {

snapshot: "snap-2025-10-01"

seeds: 42

model\_tiers: { retriever: "embed-001", reasoner: "gpt-X-safe", verifier: "critique-small" }

budgets: { tokens: 18k, wall\_ms: 8000, calls: 12 }

}

step retrv: Retriever {

inputs: { query: "policy summary + citations", slice: "2024-01..2025-10" }

budget: { tokens: 2k, wall\_ms: 800, calls: 2 }

outputs: { topk: EvidenceSet, witness: VIF, uq: UQ }

}

step reason: Reasoner {

inputs: { evidence: retrv.topk }

budget: { tokens: 10k, wall\_ms: 3500 }

outputs: { claims: ClaimSet, witness: VIF, uq: UQ }

gates: [ policy\_gate, uq\_gate ]

}

step verify: Verifier {

inputs: { claims: reason.claims, evidence: retrv.topk }

budget: { tokens: 3k, wall\_ms: 1200 }

outputs: { verdict: PASS|FAIL, deltas: Diff[], witness: VIF }

gates: [ provenance\_gate ]

}

step emit: Witness {

inputs: { verdict: verify.verdict, claims: reason.claims, trace: ALL }

outputs: { report: JSON-LD, lineage: SEGPatch }

}

edges {

retrv -> reason

retrv -> verify

reason -> verify

verify -> emit

}

}

**Expected boundary emissions:**

* retrv: RS distribution, anchors used, κ=false.
* reason: UQ bands; calibrated confidence.
* verify: per-claim decision with rule id.
* emit: JSON-LD lineage with model/weight hashes, prompt-tpl id, tool invocations, snapshot id.

**11.10 Core Data Structures (canonical)**

**StepRecord**

{

"id":"reason-1",

"role":"Reasoner",

"inputs\_hash":"sha256:...",

"budget":{"tokens":10000,"wall\_ms":3500,"calls":0},

"artifacts\_ref":"blob://.../claims.json",

"witness":{"model":"gpt-X-safe","weights":"sha256:...","prompt\_tpl":"tpl-17","tools":[], "snapshot":"snap-2025-10-01"},

"uq":{"entropy":0.19,"ece":0.05,"rs\_margin":0.21},

"gates":{"policy":"pass","uq":"pass","schema":"pass"},

"spans":[{"name":"llm.call","ms":812}]

}

**PlanManifest**

{

"name":"policy-sum-v1",

"version":"11.0",

"pins":{"retriever.index":"ann-idx-2025-09","reasoner.model":"gpt-X-safe@sha256:..."},

"seeds":42,

"budgets":{"tokens":18000,"wall\_ms":8000,"calls":12}

}

**11.11 Determinism Toolkit**

* **Seed all randomness**; expose seed in every witness.
* **Pin versions**; reject execution if a pin cannot be honored.
* **Stable sorts** everywhere; forbidden to rely on map iteration order.
* **Prompt template ids** are immutable; diff produces a **new id**.
* **Snapshot replay** mandatory in CI for critical plans.

**11.12 Observability (what the operator sees)**

* **Spans:** plan.build, step.exec, ann.search, dvns.refine, gate.run, kappa.abstain, witness.emit.
* **Metrics:** chain determinism %, κ precision/recall, budget adherence, p50/p95 latency, cost per plan, lineage completeness.
* **Alerts:** nondet drift, κ spike by domain, policy-violations, budget burn anomalies.

**11.13 Checklists (shiproom)**

* Plan IR validates (types, budgets, pins, seeds).
* All steps declare gates; HITL paths wired for risk ≥ .
* Replay passes on frozen snapshot.
* κ reasons render actionable “next questions”.
* Operator dashboard tiles populated.

**Part IV — Orchestration: APOE & DEPP**

**Chapter 12. Dynamic Emergent Prompt Pipeline (DEPP)**

**Purpose.** Turn a messy, evolving program of work into a **living, governed plan** that can span **thousands of steps**, adapt to new evidence, and stay reproducible under budgets and policy. If APOE is the compiler, **DEPP is the compiled program’s runtime**—graph-native, evidence-seeking, and self-rewriting under guardrails.

**12.0 What DEPP guarantees**

1. **Long-horizon tractability.** Breaks big goals into a **typed, sharded DAG** that executes under token/time/tool budgets.
2. **Evidence-driven evolution.** Chain structure **changes only for cause** (new evidence, failed gates, user input) and always emits a **witnessed diff**.
3. **Determinism under pins.** With fixed snapshot, seeds, and tool/version pins, the same inputs yield the same trace—or **abstain** ().
4. **Governed merges.** All chain edits pass SDF-CVF parity and policy gates; high-risk edits require **HITL two-key**.

**12.1 Master Chain as Graph**

**12.1.1 Canonical graph model (IR)**

* **Plan** : DAG partitioned into **shards** for scale and isolation.
* **Node (step)** :
* Step {
* id, role, schema\_in, schema\_out,
* budget{tokens, wall\_ms, calls, cost$},
* pins{model@hash, index@id, tool@ver, seed},
* policy\_slice, gates[], # policy, UQ, provenance, diff, HITL
* inputs\_ref, outputs\_ref, witness, uq
* }
* **Edge** : typed channel with **scope** (HHNI slice, time-slice), **fan-out policy**, and **merge operator**.

**Shard header**

Shard {

id, purpose, dependencies[], # other shard ids

resource\_caps{max\_parallel, token\_pool, wall\_ms\_pool},

pinned\_snapshot, router\_rules

}

**12.1.2 Assembly (from goal → graph)**

1. **Intent split.** Planner extracts sub-goals & constraints and emits candidate roles (retriever/reasoner/builder/verifier/critic).
2. **Scope binding.** Each candidate step binds to **HHNI slices** (system→…→sub-word indices), **time windows**, and **policy regions**.
3. **Budgeting.** Estimate token/time/calls by historical medians (per role × domain) and **reserve** per shard.
4. **DVNS kick.** For retrieval-heavy shards, seed **Dynamic Vector Navigation** to pre-rank anchors and reduce lost-in-the-middle effects.
5. **Topological layout.** Create DAG with **stable edge ordering**; annotate **fan-out/fan-in** points.

**Heuristic knobs**

* Prefer **deep-thin** over **wide-flat** when token budgets are tight.
* Split shards at **policy boundaries** and high-latency tools (network isolation).
* Co-locate steps whose inputs share HHNI anchors (cache locality).

**12.1.3 Scheduling & execution**

* **Router** chooses model tiers and tool stacks per shard.
* **Token scheduler** enforces a **per-shard pool** with back-pressure.
* **Parallelization**: map → verify → reduce; debate → adjudicate; spec-and-solve.
* **Checkpointing**: every gate emits a **StepRecord**; shard emits **ShardRecord**; plan emits **PlanTrace**.

**12.1.4 Persistence & cross-links**

* All intermediate artifacts live in content-addressed blobs; **SEG** stores lineage (claim↔evidence↔decision).
* **Snapshot pins** tie reads to memory state ; **tombstones** mark deprecations.

**12.2 Self-Rewrite via Evidence**

DEPP changes itself **only** when evidence or policy requires, and it leaves a trail.

**12.2.1 Triggers**

* **Gate failure** (policy/UQ/provenance/diff).
* **Low RS** or **high UQ** (retrieval or reasoning uncertainty).
* **External events** (new snapshots, user constraints, incident alerts).
* **Cost/latency breach** vs. budget envelope.

**12.2.2 Rewrite primitives (safe graph edits)**

* **Rescore:** change ordering/weights of edges without topology change.
* **Splice:** insert a local verifier/critic step between nodes.
* **Reroute:** rebind inputs to a different retriever/index or model tier.
* **Refactor:** split one bulky node into a micro-pipeline (maintain I/O contract).
* **Fork:** create a branch for risky exploration; **merge** only with gates + HITL if needed.
* **Quarantine:** cut a subgraph off the main line; park for human review.

All edits produce a **ChainDiff** with: reason\_codes, old/new IR fragments, witness tuples, HITL approvals (if any).

**12.2.3 Consistency & parity**

* **SDF-CVF parity**: if any edit touches code/spec/tags/traces, the quartet must pass parity gates in the same commit.
* **Replay check**: re-execute altered region on the same snapshot to confirm determinism.
* **Policy non-regression**: policy distance must not shrink below thresholds.

**12.2.4 Rewrite controller (pseudocode)**

def maybe\_rewrite(event, plan, telemetry):

cause = classify(event) # gate\_fail, low\_RS, over\_budget, policy\_violation, new\_snapshot

if cause == "policy\_violation":

return quarantine(plan.region(event.region), reason="policy")

if cause in {"low\_RS","high\_UQ"}:

return splice(plan, before=event.node, with\_="Verifier(rs\_boost)")

if cause == "over\_budget":

return refactor(event.node, into=["RetrieverLite","ReasonerShort"], shrink=True)

if cause == "new\_snapshot":

return reroute(plan, targets=affected\_edges(plan, event))

return plan # no-op

**12.3 Chain-Health Metrics (compute & act)**

Let be all steps, all shards.

**Core metrics**

* (abstention health):

Track **precision/recall** of abstentions vs. later ground truth (should catch bad cases without blocking good ones).

* **Budget adherence (BA)** per class :

Alert on **slope** (burn rate) not just level.

* **Determinism score (D)** under pins:  
  Re-run N sampled subtraces;

(Accept within hashing tolerance on float outputs.)

* **Lineage completeness (LC)**: fraction of artifacts with **VIF tuples** present and valid hashes.
* **RS-coverage (RSC)**: share of accepted answers whose **RS ≥** and **UQ ≤** .
* **Gate pass rate (GPR)** and **mean gate latency** (detect bottlenecks).

**Dashboards & thresholds**

* Redline if , , or **policy gate fails** > 0 in last hour.
* Page on slope exceeding of forecast.
* Suggest **auto-rewrite** when RSC drops below target for a shard (e.g., < 0.85).

**Telemetry plumbing**

* Emit OpenTelemetry spans: step.exec, gate.run, rewrite.apply, dvns.refine.
* Aggregate per **role × domain** to tune budgets and router rules.

**12.4 Large-Scale Planning (thousands of nodes)**

**12.4.1 Partitioning strategies**

* **Functional shards** (retrieval, reasoning, verification, build).
* **Policy shards** (PII regions, licensed content, export-restricted zones).
* **Latency shards** (network-bound vs. CPU/GPU-bound).
* **Ownership shards** (teams/tenants; enables independent releases).

**12.4.2 Parallel exec patterns**

* **Map→Verify→Reduce** for literature/codebase sweeps.
* **Debate×K** (2–3 critics) on high-stakes claims; **Verifier** adjudicates with rubric.
* **Spec-and-Solve**: spec writer and solver proceed in parallel; **Parity Gate** compares spec↔artifact.

**12.4.3 Token & memory budgets at scale**

* **Token pooling** per shard with **borrowing** under caps; hard stop at plan cap.
* **Context compaction**: dumbbell compression + HHNI slices keep **salient edges** at start/end of prompts.
* **Model tiering**: use small/fast models for broad sweeps, escalate to slow/accurate on **frontier nodes**.

**12.4.4 Failure handling & resilience**

* **Circuit breakers** for flaky tools; automatic **shadow routing** to alternates.
* **Quarantine queues** for risky rewrites; HITL triage with diffs & ChainDiff witness.
* **Warm caches**: prefetch HHNI anchors for next K steps; **pre-compute** DVNS on super-index nightly.

**12.4.5 Performance targets (reference)**

* p95 step latency: Retriever ≤ 900 ms, Reasoner ≤ 3.5 s, Verifier ≤ 1.2 s.
* Determinism ≥ 0.98; Lineage completeness ≥ 0.97.
* κ false-negative rate ≤ 2% on critical domains.
* Token budget overrun incidents ≤ 1 per 1,000 plan runs.

**12.5 ACL (AIMOS Chain Language) snippets for DEPP**

**12.5.1 Thousand-node skeleton (excerpt)**

pipeline "depp.app-blueprint.v3" {

header {

snapshot: "snap-2025-10-01"

seeds: 1337

budgets: { tokens: 220k, wall\_ms: 120000, calls: 480 }

}

shard "retrieval.sweep" {

caps { max\_parallel: 64, token\_pool: 60k }

step r0: Retriever { inputs: { query: "all blueprint reqs", slice: "HHNI:system->section" } }

fanout r0 -> r[1..128] with anchors=super\_index

reduce r[\*] -> r\_reduce using merge=maxRS

}

shard "spec-and-solve" {

step spec: Reasoner { inputs: r\_reduce; role: "SpecWriter" }

step solve: Builder { inputs: spec }

gate parity: ParityGate { inputs: {spec, solve} }

}

shard "verify-and-witness" {

step v: Verifier { inputs: { claims: spec, artifacts: solve } }

step w: Witness { inputs: { trace: ALL } }

}

}

**12.5.2 Rewrite policy**

policy rewrite {

on gate\_fail(policy|provenance) => quarantine region

on low\_RS or high\_UQ => splice Verifier(rs\_boost) before consumer

on over\_budget => refactor node shrink=true

require HITL for merges touching policy shards

}

**12.6 Interfaces: DEPP ↔ CMC/HHNI ↔ SEG ↔ SDF-CVF**

* **Reads**: DEPP steps read via **HHNI scopes** and time-slices; **RS** computed with QS·IDS·(1–DD).
* **Writes**: Only to **trace**, **witness**, and **SEG** (no direct mutation of memory state; new state arrives via **snapshot** events).
* **Parity**: Any change that emits code/spec/tag/trace must pass SDF-CVF parity gates in one atomic commit.
* **Provenance**: Every emission carries VIF: model id, weights hash, prompt-tpl id, tool ids, snapshot id, seeds, UQ vector.

**12.7 Runbook (operators & builders)**

**When κ triggers repeatedly in a shard**

* Inspect RS histograms; widen HHNI slice; enable DVNS refinement; raise model tier for frontier nodes.

**When determinism drops**

* Check pin drift (models/indexes/tools); re-pin or roll back snapshot; recompute seeds; replay sample traces.

**When token slope spikes**

* Reduce fan-out; enforce shorter completions; split shard; push more work to retriever/critic before reasoner.

**When policy gate fails**

* Quarantine region; run redaction pipeline; require HITL; log contradiction edge in SEG (do not delete).

**12.8 Checklists (ship readiness)**

* DAG validates (types, gates, pins, budgets, seeds).
* Shards have token pools & caps; router rules documented.
* Rewrite policy installed; quarantine & HITL paths wired.
* Replay passes on frozen snapshot; determinism .
* Dashboards show , BA, D, LC, RSC healthy.
* Export pack (SEG) reproducible for the plan.

**Part IV — Orchestration: APOE & DEPP**

**Chapter 13. ACL — AIMOS Chain Language**

**Purpose.** ACL is the **legible, versionable, executable** language for declaring APOE/DEPP plans. It gives you typed pipelines, explicit budgets, policy-aware gates, and machine-readable provenance so plans are **repeatable under pins** and **auditable under VIF/SEG**.

**13.0 Design goals (non-negotiables)**

* **Determinism under pins.** Same snapshot, same seeds, same tools → same trace (or abstain, ).
* **Budget-first.** Tokens/time/calls/cost are first-class and enforceable.
* **Governed I/O.** Every boundary emits **witness** (provenance + UQ) and passes **gates**.
* **Memory-native.** Reads target **HHNI selectors** and time-slices; writes emit **SEG** facts, never mutate memory directly.
* **Composability.** Pipelines shard/compose like software modules; changes must pass **SDF-CVF parity**.

**13.1 Surface syntax (EBNF)**

pipeline = "pipeline", string, "{", header, { shard }, "}" ;

header = "header", "{",

"snapshot", ":", string,

"seeds", ":", integer,

"budgets", ":", budget\_map,

[ "pins", ":", pin\_map ],

[ "policy", ":", policy\_ref ],

"}" ;

shard = "shard", string, "{",

[ "caps", "{", cap\_kv, "}" ],

{ decl },

"}" ;

decl = step | gate | flow | fanout | reduce | policy\_block ;

step = "step", ident, ":", role, "{",

[ "inputs", ":", input\_map ],

[ "slice", ":", hhni\_sel ],

[ "time", ":", time\_sel ],

[ "model", ":", model\_ref ],

[ "tools", ":", tool\_list ],

[ "budget",":", budget\_map ],

[ "pins", ":", pin\_map ],

[ "policy",":", policy\_ref ],

"}" ;

role = "Retriever" | "Reasoner" | "Verifier" | "Builder" | "Critic" | "Operator" | "Witness" ;

gate = "gate", ident, ":", gate\_type, "{",

"inputs", ":", input\_map,

[ "policy", ":", policy\_ref ],

[ "strict", ":", bool ],

"}" ;

gate\_type = "ParityGate" | "PolicyGate" | "ProvenanceGate" | "UQGate" | "DiffGate" ;

flow = "flow", ident, "{", { decl }, "}" ;

fanout = "fanout", ident, "->", ident\_range, [ "with", fanout\_opts ] ;

reduce = "reduce", ident\_range, "->", ident, "using", reduce\_op ;

reduce\_op = "merge=maxRS" | "vote=majority" | "score=mean" | "adjudicate=verifier" ;

ident\_range = ident | ident, "[", integer, "..", integer, "]" ;

budget\_map = "{", budget\_kv, { ",", budget\_kv }, "}" ;

budget\_kv = ("tokens" | "wall\_ms" | "calls" | "cost"), ":", integer ;

cap\_kv = ("max\_parallel" | "token\_pool" | "wall\_ms\_pool"), ":", integer ;

pin\_map = "{", pin\_kv, { ",", pin\_kv }, "}" ;

pin\_kv = ("model" | "index" | "tool" | "seed"), ":", string ;

input\_map = "{", input\_kv, { ",", input\_kv }, "}" ;

input\_kv = ident, ":", ( string | ident | hhni\_sel | seg\_ref ) ;

hhni\_sel = "HHNI:", path ; (\* e.g., HHNI:system->section("Auth")->paragraph#3 \*)

time\_sel = "ASOF:", iso8601 | "WINDOW:", iso8601,"..",iso8601 ;

policy\_ref = ident ;

model\_ref = string ;

tool\_list = "[", string, { ",", string }, "]" ;

seg\_ref = "SEG:", path ;

bool = "true" | "false" ;

path = /\* URI-like selector with filters \*/ ;

**Whitespace/comments.** // line, /\* block \*/ ignored.  
**Strings.** UTF-8, \" escapes supported.

**13.2 Core constructs & semantics**

* **pipeline**: top-level plan with **snapshot pin** and **global budgets**.
* **shard**: isolation domain with its own resource caps and routing; independently schedulable.
* **step**: a typed unit of work with **role**, **inputs**, **slice/time** selectors, **model/tools**, **budget**, **pins**, **policy**.
* **gate**: a typed check; must **succeed** before downstream steps read outputs.
* **fanout/reduce**: parallelization primitives (map→reduce, debate→adjudicate, etc.).
* **flow**: nested subgraph (macro) for reuse; inlined at compile unless marked export.

**Execution invariant.** Each step returns (artifacts, witness, uq). Gates consume these to decide pass/fail; failures raise **rewrite** or **quarantine** per policy.

**13.3 Types, kinds, and schemas**

* **Kinds**: Artifact, Claim, Evidence, Trace, Blob, JSON, Table, Code, Spec, Diff, UQ, Witness.
* **Structural types**: JSON Schemas referenced by $id.
* **Role I/O contracts** (illustrative):

| **Role** | **Input kinds** | **Output kinds** |
| --- | --- | --- |
| Retriever | Query(JSON), HHNI slice, Time slice | Evidence[], UQ, Witness |
| Reasoner | Evidence[], Spec/Question | Claim[], Explanation, UQ |
| Verifier | Claim[], Evidence[]/Artifacts | Verdicts, Critiques, UQ |
| Builder | Spec/Plan, Codebase slice | Code/Artifacts, Diffs, UQ |
| Witness | Trace, Metadata | VIF tuples, SEG writes |
| Critic | Any | Review notes, Risk score, UQ |
| Operator | Telemetry | Actions, Hints, UQ |

**Static typing.** Compiler checks edge compatibility by kind + JSON Schema. Subtyping via $ref and oneOf. Mismatches → compile error.

**13.4 Effects & capabilities**

ACL annotates **effects** to constrain tool power and data egress.

* **Effects**: net.read, net.write, fs.read, fs.write, proc.exec, secrets.read, mail.send, db.query, seg.write.
* **Capability tokens** bound at runtime to principals/tenants, scoped per step/shard.

**Example (cap-limited step):**

step r0: Retriever {

inputs: { query: "oauth2 threat models", slice: HHNI:system->section("Security") }

tools: ["web.search"]

budget: { tokens: 800, wall\_ms: 1500, calls: 3 }

policy: sec\_web\_readonly

}

**13.5 Budgets & pins**

* **Budgets** hard-fail when exceeded unless soft=true in header policy (discouraged).
* **Pins** freeze versions: model:"gpt-4o@sha256:…", index:"hhni:auth@v37", tool:"grep@1.2.5", seed:"1337".

**Determinism contract.** If pins + snapshot identical, runtime must produce **bit-identical artifacts** (modulo allowed float error windows) or **abstain**.

**13.6 Gates & witnesses**

* **ParityGate**: code/spec/tags/traces parity (SDF-CVF).
* **PolicyGate**: data residency, PII/PHI filters, license compliance.
* **ProvenanceGate**: VIF completeness: (model id, weights hash, prompt template id, tool ids, snapshot id, seeds).
* **UQGate**: calibration and uncertainty bands (ECE/entropy/ensemble consistency).
* **DiffGate**: prevents destructive or outsized changes (size/scope thresholds).

Every gate emits a **GateRecord** (+ witness) into SEG.

**13.7 Namespaces, imports, and policies**

import policy "policies/security.aclp" as sec

import flow "flows/spec\_and\_solve.acl" as s2

policy rewrite {

on gate\_fail(policy|provenance) => quarantine region

on low\_RS or high\_UQ => splice Verifier(rs\_boost) before consumer

on over\_budget => refactor node shrink=true

require HITL for merges touching policy shards

}

* **Imports** are content-addressed; compiler caches by hash.
* **Policies** can be applied per step/shard/pipeline; resolved **least-privilege**.

**13.8 Control flow (bounded)**

* **Fanout/fanin** for parallelism; **no unbounded loops**.
* **Iterative refinement** is expressed as **bounded loops**:

flow refine(code\_in, max\_iter=3) {

step c0: Critic { inputs: { code: code\_in } }

fanout c0 -> fix[1..max\_iter]

reduce fix[1..max\_iter] -> pick using adjudicate=verifier

}

**13.9 Memory-native reads & SEG writes**

* **HHNI selectors** scope retrieval: HHNI:system->section("Auth")->paragraph#3.
* **Time slicing** binds reads to ASOF:2025-10-12T00:00:00Z or WINDOW:….
* **SEG writes** are append-only facts with temporal validity; **contradiction** is an edge, not deletion.

**13.10 Compilation & IR mapping**

The compiler emits a **canonical IR**:

{

"pipeline": "depp.app-blueprint.v3",

"snapshot": "snap-2025-10-01",

"shards": [

{

"id": "retrieval.sweep",

"caps": {"max\_parallel":64,"token\_pool":60000},

"nodes": [

{"id":"r0","role":"Retriever","iotype":{"in":["Query","Slice"],"out":["Evidence[]","UQ","Witness"]},"budget":{…},"pins":{…},"policy":"sec\_web\_readonly"}

],

"edges":[{"from":"r0","to":"r[1..128]","type":"fanout","opts":{"anchors":"super\_index"}}]

}

]

}

This IR is the **source of truth** for scheduling, replay, and audit.

**13.11 Static analyses & lints**

* **Type & kind checking** across edges.
* **Budget feasibility** (sum of min-costs ≤ caps).
* **Policy reachability** (no illegal effect paths).
* **Determinism pins** present on frontier nodes.
* **HHNI selector validity** (no dangling indices).
* **Gate coverage** at all boundaries (in/out of shard, before merge).

**Lints** (warnings): oversize prompts, missing UQ emission, fanout without reduce, policy wildcard, etc.

**13.12 Runtime semantics (scheduler)**

* **Shard scheduler**: weighted fair-share across shards; local token pools; **back-pressure** on overruns.
* **Retries**: idempotent steps may retry with jitter; non-idempotent require policy exception.
* **Rewrite hooks**: on gate fail/low RS/over-budget invoke **rewrite controller** (Chapter 12).
* **Quarantine/HITL**: risky diffs route to human review with **ChainDiff** and **witness**.

**13.13 Security model**

* **Capabilities** bound per step/shard; default deny.
* **Data egress** logged with **provenance**; redaction policies enforce PII/PHI constraints.
* **Sandboxing**: tool processes run jailed; network egress by allow-list.

**13.14 Worked examples**

**13.14.1 Research pipeline (literature sweep → claim)**

pipeline "research.oauth2.threats.v1" {

header {

snapshot: "snap-2025-10-01"

seeds: 101

budgets: { tokens: 120k, wall\_ms: 90000, calls: 240 }

pins: { model: "gpt-4o@sha256:ab12…" }

}

shard "retrieval" {

caps { max\_parallel: 32, token\_pool: 40k }

step q: Retriever {

inputs: { query: "latest oauth2 threat models", slice: HHNI:system->section("Security") }

tools: ["web.search","pdf.extract"]

budget: { tokens: 1200, wall\_ms: 2500, calls: 8 }

policy: sec\_web\_readonly

}

fanout q -> doc[1..64] with anchors=super\_index

reduce doc[1..64] -> evidence using merge=maxRS

}

shard "analysis" {

step r: Reasoner { inputs: { evidence: evidence, question: "What new threats post-2024?" } }

step v: Verifier { inputs: { claims: r, evidence: evidence } }

gate uq: UQGate { inputs: { uq: r } }

gate prov: ProvenanceGate { inputs: { witness: v } }

step w: Witness { inputs: { trace: ALL } }

}

}

**13.14.2 Builder pipeline (spec→code→tests→parity)**

pipeline "builder.auth-service.v2" {

header {

snapshot: "snap-2025-10-01"

seeds: 7

budgets: { tokens: 180k, wall\_ms: 180000, calls: 420 }

}

shard "spec-and-solve" {

step spec: Reasoner {

inputs: { question: "Produce OpenAPI for Auth svc: login, refresh, mfa" , slice: HHNI:system->section("Auth") }

budget: { tokens: 2500, wall\_ms: 4000, calls: 4 }

}

step code: Builder {

inputs: { spec: spec, codebase: HHNI:system->section("code")->folder("services/auth") }

tools: ["git.patch","tsc","jest"]

budget: { tokens: 3500, wall\_ms: 10000, calls: 10 }

}

gate parity: ParityGate { inputs: { spec: spec, artifacts: code }, strict: true }

gate diff: DiffGate { inputs: { artifacts: code } }

step tests: Operator { inputs: { cmd: "jest --coverage" } }

gate policy: PolicyGate { inputs: { artifacts: code }, policy: license\_policy }

step w: Witness { inputs: { trace: ALL } }

}

}

**13.14.3 Ops RCA (logs→hypotheses→checks→decision)**

pipeline "ops.rca.api-latency.v1" {

header { snapshot: "snap-2025-10-10"; seeds: 42; budgets: { tokens: 60k, wall\_ms: 60000, calls: 120 } }

shard "ingest" {

step ls: Retriever {

inputs: { query: "p95 latency spike", slice: HHNI:system->section("Observability") , time: WINDOW:2025-10-10T12:00:00Z..2025-10-10T13:00:00Z }

tools: ["logs.query"]

budget: { tokens: 800, wall\_ms: 2000, calls: 4 }

}

}

shard "rca" {

step hyp: Reasoner { inputs: { evidence: ls, question: "Likely causes and tests?" } }

fanout hyp -> check[1..5]

reduce check[1..5] -> verdict using adjudicate=verifier

gate prov: ProvenanceGate { inputs: { witness: verdict } }

step w: Witness { inputs: { trace: ALL } }

}

}

**13.15 Tooling: CLI, fmt, and IR**

* **acl fmt**: canonical formatting, stable diffs.
* **acl lint**: runs static analyses & lints.
* **acl compile**: emits IR JSON with content hashes.
* **acl replay --snapshot …**: re-executes traces under pins.
* **acl export seg**: produces regulatory export packs.

**13.16 Error catalog (selected)**

| **Code** | **Message** | **Phase** | **Action** |
| --- | --- | --- | --- |
| ACL-E001 | Type mismatch: expected Evidence[], got Code | Compile | Fix edge/types |
| ACL-E014 | Gate coverage missing on shard boundary | Lint | Add gate |
| ACL-E022 | Budget infeasible: min path > token\_pool | Lint | Raise caps or refactor |
| ACL-E037 | Missing pins on frontier node | Compile | Add pins |
| ACL-R101 | Budget overrun (tokens) | Runtime | Rewrite/refactor |
| ACL-R118 | UQ calibration fail | Runtime | Splice verifier |
| ACL-S201 | Policy violation (PII egress) | Runtime | Quarantine + HITL |
| ACL-D301 | Determinism breach under pins | Replay | Re-pin/rollback |

**13.17 Contract summary**

* **Inputs**: HHNI/SEG selectors + JSON-typed payloads.
* **Outputs**: Artifacts + Witness + UQ; immutable; SEG-anchored.
* **Safety**: Budgets enforced; policies applied; gates required.
* **Determinism**: Pins + snapshot = replayable trace (or ).
* **Evolution**: Any change → ChainDiff + parity gates + (if high-risk) HITL.

**Outcome.** ACL makes orchestration **software-grade**: you can **read it, diff it, lint it, compile it, replay it**, and explain it to an auditor. It is the language that turns APOE intent into DEPP reality without losing memory, budget, or provenance.

**Part V — Verifiable Intelligence: VIF & SEG**

**Chapter 14. Provenance & Uncertainty (VIF)**

**Aim.** Make every artifact **answerable**: *who* produced it (model & tools), *with what* (prompts, data slices), *under which pins* (versions, seeds), *with how much uncertainty* (quantified, calibrated), and *how to replay it* (deterministically or with bounded variance).

**14.1 VIF metadata — the witness tuple**

**Definition (witness tuple).**  
For any produced artifact (text, code, table, decision), we attach a *witness* :

* **actor**: { role, model\_id, weights\_hash, runtime\_hash }
* **inputs**: { prompt\_template\_id, prompt\_args, hhni\_slices[], seg\_refs[], tools\_used[] }
* **pins**: { snapshot\_id, tool\_versions{}, index\_versions{}, seed, policy\_ids[] }
* **provenance**: { source\_docs[], chunk\_locs[], citations[], license\_tags[] }
* **uq**: { method[], scores, bands, calibration, decision: {kappa, gate} }
* **trace**: { step\_id, chain\_id, started\_at, ended\_at, cost, tokens, wall\_ms }

**Minimal JSON (machine-readable, embeddable):**

{

"actor": {

"role": "Reasoner",

"model\_id": "gpt-4o",

"weights\_hash": "sha256:3f…c9",

"runtime\_hash": "sha256:rt…aa"

},

"inputs": {

"prompt\_template\_id": "tpl.claim.v2",

"prompt\_args": {"question":"Summarize OAuth2 threats post-2024"},

"hhni\_slices": ["HHNI:system->section('Security')->paragraph#12"],

"seg\_refs": ["SEG:claims/2025-10-12/oidc"],

"tools\_used": ["web.search@1.9.3","pdf.extract@0.7.1"]

},

"pins": {

"snapshot\_id": "snap-2025-10-01",

"tool\_versions": {"web.search":"1.9.3","pdf.extract":"0.7.1"},

"index\_versions": {"hhni:security":"v37"},

"seed": 101,

"policy\_ids": ["sec\_web\_readonly","pii\_guard\_v3"]

},

"provenance": {

"source\_docs": [

{"doc\_id":"doi:10.1145/…","hash":"sha256:…","license":"CC-BY-4.0"}

],

"chunk\_locs": [{"doc\_id":"doi:…","start":"p.4 ¶2","end":"p.5 ¶1"}],

"citations": ["[1] …"]

},

"uq": {

"method": ["temp\_sweep","self\_consistency","ece\_postcal"],

"scores": {"entropy":0.23,"margin":0.61,"var\_sc":0.08},

"bands": {"green":[0.0,0.3],"amber":[0.3,0.6],"red":[0.6,1.0]},

"calibration": {"ECE":0.04,"bins":15},

"decision": {"kappa":0.55,"gate":"UQGate:pass"}

},

"trace": {

"step\_id":"r:claim-01",

"chain\_id":"research.oauth2.threats.v1",

"started\_at":"2025-10-12T11:02:14Z",

"ended\_at":"2025-10-12T11:02:22Z",

"tokens": 1842,

"wall\_ms": 8240,

"cost": {"usd": 0.032}

}

}

**Design rules (VIF-minimal vs VIF-full):**

* *Minimal* must include: model\_id, weights\_hash, prompt\_template\_id, snapshot\_id, hhni\_slices, at least one uq.score.
* *Full* must add tool & index versions, seed, policy\_ids, ECE, trace timing & token/costs, and provenance.chunk\_locs.

**SEG anchoring.**  
Each witness is written as a **SEG node** witness::<uuid> with edges:

* (artifact) <-[witnessed\_by]- (witness)
* (witness) -[derived\_from]-> (source\_doc::<id>#chunk::<span>)
* (witness) -[used\_tool]-> (tool::<name>@<ver>)
* (witness) -[ran\_under]-> (snapshot::<id>)

**14.2 UQ methods — computing uncertainty you can trust**

We quantify uncertainty along three axes:

* **Retrieval uncertainty** : Is the grounding evidence *sufficient & consistent*?
* **Model uncertainty** : Is the model *confident & consistent* across samples?
* **Calibration error** : Do reported confidences match reality?

**14.2.1 Per-step signals**

1. **Entropy / margin (single pass).**

* on class/logit heads (where available).
* **Margin** .  
  Mapped to via min–max per head. Cheap, universal.

1. **Temperature sweep (multi-pass).**  
   Run the same prompt at ; measure textual dispersion (e.g., pairwise ROUGE-L or embedding distance).

* Score: .

1. **Self-consistency / ensembles.**  
   independent decodes → majority consensus or agreement rate.

* Score: .

1. **MC-dropout (when logits exposed).**  
   Sample forward passes with dropout; use predictive variance.

* Score: .

1. **Verifier disagreement.**  
   A second model/role critiques the claim; disagreement lifts uncertainty.

* Score: .

1. **Retrieval sufficiency.**

* Evidence coverage: fraction of claim spans aligned to citations.
* Cross-doc consistency: contradiction rate among top-k sources.
* Score: .

**Aggregation (bounded risk):**

with . Defaults: .

**14.2.2 Calibration**

We *post-calibrate* the model’s raw confidence to a calibrated using:

* **Platt scaling (logistic)** on a labeled dev set: .
* **Isotonic regression** when monotonic but non-linear mapping fits better.
* **ECE/ACE** (Expected/Adaptive Calibration Error): bucket predictions, compare avg confidence vs empirical accuracy; target .

Calibration artifacts are versioned per **(model\_id, domain, prompt\_family)** and pinned in the witness (calibration.calib\_id).

**14.2.3 κ-abstention policy**

Define risk bands after calibration:

* **Green:** → proceed.
* **Amber:** → show banner; route Verifier or request user confirmation for high-impact actions.
* **Red:** → **abstain** (); propose next steps (more evidence, narrower slice, HITL).

ACL example:

gate uq: UQGate { inputs: { uq: r }, policy: "risk\_bands\_v3" }

**14.3 Confidence bands & UI cues**

**Principles.** Confidence is *visible by default*, *layered*, and *explainable on click*.

* **Badge** on every artifact title: ✓ High confidence / ~ Medium / ! Low.
* **Color**: green/amber/red, color-blind safe palettes.
* **Tooltip**: “Why this band?” → show top-3 drivers (e.g., *low evidence coverage*, *verifier disagreement*, *high sample variance*).
* **Inline citations**: footnotes with doc\_id#chunk hover to preview source, highlight aligned spans.
* **Diff-aware code panes**: risk badge tied to changed hunks; link to tests that justify confidence.
* **Action gating**: destructive ops (write, deploy, merge) are disabled or require HITL outside green band.

**Recommended payload (client contract):**

{

"confidence": {

"band": "amber",

"score": 0.47,

"drivers": [

{"name":"evidence\_coverage","value":0.62},

{"name":"self\_consistency\_var","value":0.11},

{"name":"verifier\_disagreement","value":1.0}

],

"suggested\_next": [

"Open verifier report",

"Add sources for section §2.3",

"Narrow time window to last 90 days"

]

}

}

**Accessibility & UX notes.**

* Always pair color with text/icon; offer “Why-this-confidence?” explainer modal.
* Permit **pinning** of a witness view for audits; expose **copy as JSON**.

**14.4 Reproducibility & replays**

**Replay levels.**

1. **Deterministic replay** (under pins):  
   Same snapshot\_id, seed, prompt\_template\_id, model\_id@weights\_hash, tool\_versions, index\_versions → identical output, or **abstain** if upstream pins missing.
2. **Quasi-deterministic replay** (external calls stubbed):  
   Record I/O of tools (HTTP responses, file reads) as **CAS blobs**; replay uses stubs. Drift shows as **provenance deltas**.
3. **Robustness replay** (variance bounds):  
   Re-execute with decodes/temps; report mean±std, majority answer, and band stability. Store **ReplaySet** witness with set\_id.

**What must be pinned.**

* Models: model\_id, weights\_hash.
* Prompts: prompt\_template\_id@hash, prompt\_args.
* Data slices: hhni\_slices@version, seg\_refs@rev.
* Tools/indices: {name: version}, **container digests**.
* Environment: OS/driver digests, **runtime\_hash**; GPU type if applicable.
* Seeds: seed (global) + per-step overrides.

**Failure semantics.**  
If determinism breaks under pins → emit ACL-D301 Determinism breach, attach a *diagnostic witness* that enumerates differing hashes; auto-rollback or mark chain **tainted** per policy.

**Storage & retention.**  
Witnesses, stubs, and calibration packs are content-addressed; retention policy defaults:

* High-risk pipelines: ≥ 6 months.
* PII/PHI: redact at *ingest*; retain hashed surrogates and selectors only.
* Export: acl export seg --bundle=vif+trace produces machine-readable audit packs.

**14.5 Putting it together — the gate math and policy**

**Unified risk score.**

with defaults .

**Gate decision.**

if u <= 0.30: pass

elif u <= 0.60:

pass\_with\_banner; require Verifier on downstream write/deploy

else:

abstain(kappa); propose next\_steps()

**ACL snippet (policy-aware):**

gate uq: UQGate {

inputs: { uq: r }

policy: risk\_bands\_v3 // stores thresholds and next-step recipes

strict: true // block writes on amber+

}

**14.6 Checklists (implementation-ready)**

**Producer (at every step):**

* Attach weights\_hash, prompt\_template\_id@hash, seed.
* Record hhni\_slices and seg\_refs used.
* Compute at least two signals (e.g., entropy + self-consistency).
* If RAG, compute (coverage & contradictions).
* Apply per-model calibration; include ECE.
* Emit witness → SEG with edges.

**Consumer (gates & UI):**

* Enforce κ-policy by band; disable destructive actions on amber+.
* Show badge + drivers; link to sources with chunk previews.
* Offer **Replay** (deterministic or robust) when band≠green.
* Persist gate record in SEG; surface in audit view.

**Outcome.**  
VIF makes every answer **provable**, every confidence **quantified & calibrated**, and every run **replayable**. Together with ACL and HHNI, it upgrades outputs from *claims* to **witnessed evidence**, enabling safe automation, regulatory compliance, and user trust.

**Part V — Verifiable Intelligence: VIF & SEG**

**Chapter 15. Shared Evidence Graph (SEG)**

**Aim.** Make *evidence itself* a first-class, queryable substrate. The SEG stores claims, sources, tool calls, witnesses, decisions, and their temporal validity so that any output can be traced, sliced “as-of” time, exported for audit, and replayed.

**15.1 JSON-LD schema & SHACL**

**15.1.1 Vocabulary & classes**

**Namespaces**

@vocab: https://aimos.dev/seg#

seg: https://aimos.dev/seg#

schema: https://schema.org/

prov: http://www.w3.org/ns/prov#

xsd: http://www.w3.org/2001/XMLSchema#

**Core classes**

* seg:Claim — a statement the system asserts.
* seg:Artifact — any produced object (text/code/table/file).
* seg:Source — external input (doc, API, dataset).
* seg:Snippet — a bounded region inside a Source.
* seg Wit ness — the VIF tuple bound to an Artifact/Claim.
* seg:ToolCall — invocation with inputs/outputs & version pins.
* seg:Decision — an operational act (merge/deploy/publish).
* seg:Snapshot — frozen memory/index/tool state.
* seg:Model — model id & weights hash.
* seg:CalibrationPack — post-hoc calibration artifact.
* seg:Policy — safety/guardrail policy bound to a run.

**Key properties**

* Structural: seg:hasWitness, seg:produced, seg:usesTool, seg:ranUnderSnapshot, seg:usesModel, seg:appliesPolicy
* Lineage: seg:supports, seg:derivesFrom, seg:contradicts, seg:annotates, seg:citesChunk
* Temporal: seg:validFrom, seg:validTo, seg:txFrom, seg:txTo
* Integrity: seg:hash, seg:contentAddress, seg:license, seg:modality
* UQ/Confidence: seg:confidenceBand, seg:uncertaintyScore, seg:calibratedECE

**15.1.2 JSON-LD @context**

{

"@context": {

"@vocab": "https://aimos.dev/seg#",

"schema": "https://schema.org/",

"prov": "http://www.w3.org/ns/prov#",

"id": "@id",

"type": "@type",

"validFrom": {"@id": "seg:validFrom", "@type": "xsd:dateTime"},

"validTo": {"@id": "seg:validTo", "@type": "xsd:dateTime"},

"txFrom": {"@id": "seg:txFrom", "@type": "xsd:dateTime"},

"txTo": {"@id": "seg:txTo", "@type": "xsd:dateTime"},

"supports": {"@id": "seg:supports", "@type": "@id"},

"derivesFrom": {"@id": "seg:derivesFrom", "@type": "@id"},

"contradicts": {"@id": "seg:contradicts", "@type": "@id"},

"hasWitness": {"@id": "seg:hasWitness", "@type": "@id"},

"produced": {"@id": "seg:produced", "@type": "@id"},

"usesTool": {"@id": "seg:usesTool", "@type": "@id"},

"ranUnderSnapshot": {"@id": "seg:ranUnderSnapshot", "@type": "@id"},

"usesModel": {"@id": "seg:usesModel", "@type": "@id"},

"appliesPolicy": {"@id": "seg:appliesPolicy", "@type": "@id"},

"citesChunk": {"@id": "seg:citesChunk", "@type": "@id"},

"confidenceBand": "seg:confidenceBand",

"uncertaintyScore": {"@id": "seg:uncertaintyScore", "@type": "xsd:double"},

"calibratedECE": {"@id": "seg:calibratedECE", "@type": "xsd:double"},

"hash": "seg:hash",

"contentAddress": "seg:contentAddress",

"modality": "seg:modality",

"license": "seg:license"

}

}

**15.1.3 Minimal example (one claim, witnessed, with sources)**

{

"@context": "https://aimos.dev/seg/context.json",

"id": "seg:claim/7f820",

"type": "seg:Claim",

"modality": "text/plain",

"validFrom": "2025-10-12T11:02:22Z",

"validTo": null,

"txFrom": "2025-10-12T11:02:22Z",

"txTo": null,

"hasWitness": "seg:witness/9a12b",

"supports": ["seg:source/snippet/arxiv-2502.14255#p3"],

"contradicts": [],

"confidenceBand": "amber",

"uncertaintyScore": 0.47

}

{

"id": "seg:witness/9a12b",

"type": "seg:Witness",

"usesModel": "seg:model/gpt-4o@sha256-3f...c9",

"ranUnderSnapshot": "seg:snapshot/snap-2025-10-01",

"usesTool": ["seg:tool/web.search@1.9.3"],

"appliesPolicy": ["seg:policy/pii\_guard\_v3"],

"calibratedECE": 0.04,

"produced": "seg:artifact/3de77",

"txFrom": "2025-10-12T11:02:14Z",

"txTo": "2025-10-12T11:02:22Z"

}

{

"id": "seg:source/snippet/arxiv-2502.14255#p3",

"type": "seg:Snippet",

"contentAddress": "cas:sha256-b9…ef",

"license": "CC-BY-4.0",

"validFrom": "2025-09-30T00:00:00Z",

"txFrom": "2025-10-01T08:00:00Z"

}

**15.1.4 SHACL shapes (conformance)**

**Claim must have a witness and at least one lineage edge**

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix seg: <https://aimos.dev/seg#> .

seg:ClaimShape a sh:NodeShape ;

sh:targetClass seg:Claim ;

sh:property [

sh:path seg:hasWitness ;

sh:minCount 1 ;

] ;

sh:property [

sh:path seg:supports ;

sh:minCount 1 ;

] ;

sh:property [

sh:path seg:confidenceBand ;

sh:in ("green" "amber" "red") ;

] .

**Witness must pin model & snapshot**

seg:WitnessShape a sh:NodeShape ;

sh:targetClass seg:Witness ;

sh:property [ sh:path seg:usesModel ; sh:minCount 1 ] ;

sh:property [ sh:path seg:ranUnderSnapshot ; sh:minCount 1 ] ;

sh:property [ sh:path seg:txFrom ; sh:datatype xsd:dateTime ] .

**15.2 Temporal graphs & time-slicing**

We use **bi-temporal** semantics:

* **Valid time** (validFrom/validTo): when the statement is intended to hold in the domain.
* **Transaction time** (txFrom/txTo): when the record existed in the SEG store.

**Rules**

* Records are **never hard-deleted**; they are **ended** by setting txTo.
* Deprecations use **contradiction edges** (seg:contradicts) plus optional validTo.
* **Snapshots** anchor “as-of” state: a query with asOf=2025-10-12T11:05Z includes nodes with txFrom ≤ asOf < txTo|∞.

**Time-slice resolver (pseudo)**

function slice(graph, asOf):

return nodes E where E.txFrom <= asOf and (E.txTo is null or E.txTo > asOf)

**Validity join (domain time)**

* For domain questions (e.g., “what did we believe *then*?”), filter by both validFrom/validTo and txFrom/txTo.

**15.3 Lineage: Claim → Evidence → Decision**

**Path semantics**

* A *valid* claim must have at least one **supported path**:  
  Claim <-hasWitness- Witness -ranUnderSnapshot-> Snapshot  
  and either  
  Claim -supports-> Snippet -partOf-> Source **or** Claim -derivesFrom-> Artifact|ToolCall.

**Path scoring (for audit UX)**

* Edge weights default to 1.0; contradiction edges carry negative influence .
* Aggregate lineage confidence for a claim:

where is the claim’s uncertainty score and a logistic squash to .

**Queries**

*SPARQL-ish (RDF stores)*

SELECT ?claim ?witness ?snippet ?source WHERE {

?claim a seg:Claim ;

seg:hasWitness ?witness ;

seg:supports ?snippet .

?snippet a seg:Snippet ;

seg:contentAddress ?cas .

OPTIONAL { ?snippet seg:partOf ?source . }

FILTER (?asOf >= ?w\_txFrom && (bound(?w\_txTo) = false || ?asOf < ?w\_txTo))

}

*Cypher (property graphs like Neo4j)*

MATCH (c:Claim)-[:HAS\_WITNESS]->(w:Witness),

(c)-[:SUPPORTS]->(sn:Snippet)

WHERE w.txFrom <= $asOf AND (w.txTo IS NULL OR w.txTo > $asOf)

RETURN c, w, sn

*Gremlin (TinkerPop)*

g.V().hasLabel('Claim').

where(out('HAS\_WITNESS').has('txFrom', lte(asOf)).

or().has('txTo', null).

has('txTo', gt(asOf))).

project('claim','witness','snippet').

by(id()).

by(out('HAS\_WITNESS').valueMap()).

by(out('SUPPORTS').valueMap())

**UI patterns**

* **Rail view**: left rail is the claim; right rail shows sources/snippets with aligned highlights.
* **Delta view**: pick any two asOf times to see added/ended edges and changed witnesses.
* **Decision ladders**: show the exact graph slice that justified a Decision (deploy/merge), including UQ band at the time.

**15.4 Export & regulatory packages**

**Objectives**

* One-click, machine-readable bundles for auditors/regulators.
* Deterministic replay or *stubbed* replay of external calls.

**15.4.1 Bundle structure (zip)**

seg-export/

manifest.json

graph.ndjson # nodes & edges (time-sliced)

blobs/ # CAS payloads (stubs for external)

sha256-... # source snippets, tool outputs

witnesses/ # VIF witnesses (json)

witness-9a12b.json

calibrations/

calib-pack-ids.json # mapping & hashes

policies/

pii\_guard\_v3.json # active policy configs

signatures/

manifest.sig # detached signature

**manifest.json**

{

"seg\_version": "1.0",

"exported\_at": "2025-10-12T11:10:00Z",

"as\_of": "2025-10-12T11:05:00Z",

"scope": {

"root\_claims": ["seg:claim/7f820"],

"include\_decisions": true,

"depth": 3

},

"hashes": {

"graph": "sha256:af…20",

"witnesses": ["sha256:01…aa","sha256:b3…1f"]

},

"signing": {

"algorithm": "ed25519",

"public\_key\_fingerprint": "ed25519:6c9a…"

},

"retention": {

"policy\_id": "ret\_high\_risk\_v2",

"min\_months": 6

}

}

**Signing**

* Generate manifest.sig as a detached Ed25519 signature over the canonicalized manifest.json and graph.ndjson hashes.
* Store public key fingerprint; publish key in your compliance registry.

**Privacy & redaction**

* Redact PII **at ingest**; SEG holds selectors (e.g., salted hashes) and data class tags.
* Export bundles include **redaction map** (what was removed & why) for auditors.

**EU AI Act mapping (high-risk systems)**

* **Art. 12 (logging):** graph.ndjson + witnesses/\*.json with tx & valid times.
* **Art. 52 (synthetic labeling):** Artifact nodes carry seg:modality and synthetic marker; export includes labeling policy.
* **Retention (≥ 6 months):** manifest.retention.min\_months and store policy id.
* **Risk management:** include UQ calibration packs & policy configs in bundle.

**15.5 Storage, indexing, and scale**

**Backends**

* **RDF (e.g., Jena, Neptune-RDF):** strong JSON-LD/SHACL alignment; native SPARQL/time filters.
* **Property graphs (Neo4j, Neptune-PG):** easier path queries & ops dashboards; use custom time-slice middleware.
* **Hybrid:** store JSON-LD canonically in object store; mirror into PG for fast lineage UI.

**Indexes**

* Node: (type), (txFrom, txTo), (validFrom, validTo), (contentAddress).
* Edge: (src, rel, dst), plus tx\* and valid\*.
* Secondary: (confidenceBand), (uncertaintyScore), (policyId).
* Sharding: by snapshot\_id or by (project, chain\_id) domains.

**Throughput tips**

* Write **append-only** batches; end records by setting txTo (no deletes).
* Pre-compute **lineage cones** (transitive closure summaries) for hot claims.
* Cache **witness packs** (VIF) by (model\_id, prompt\_template\_id, seed).

**15.6 SEG API (read-only)**

**GET /seg/lineage**  
Query parameters: root, depth, asOf, include=decisions,snippets,witnesses  
Response: DAG slice (nodes+edges) + lineage confidence (LC) + summary counts.

**GET /seg/witness/:id**  
Returns the VIF JSON (Chapter 14 schema) with integrity hashes.

**POST /seg/export**  
Body: { "roots":[…], "depth":3, "asOf":"…", "format":"bundle/v1" }  
Returns: signed bundle URL + sha256.

**GET /seg/search**  
Text/metadata search over Claim/Artifact/Source with filters on bands, policies, time.

*All endpoints are read-only.* Mutations happen only through the memory/orchestration pipelines that already emit witnesses and graph deltas.

**15.7 Checklists**

**Producer (pipelines)**

* Emit JSON-LD nodes/edges for Claim/Artifact/Witness/ToolCall/Snapshot.
* Fill both tx\* and valid\* (if domain time known).
* Use content-addressed blobs for sources/snippets & store license tags.
* Append contradiction edges rather than deleting outdated claims.

**Platform**

* Enforce SHACL shapes on ingest; reject malformed witnesses.
* Maintain time-slice resolver and cache for hot asOf queries.
* Provide signed export; keep public keys rotated & pinned.
* Monitor lineage LC drift and band distributions over time.

**UX**

* Always show “as-of” clock for views.
* One-click: *Show lineage*, *Compare times*, *Export bundle*.
* Inline highlights for cited snippets with CAS previews.

**15.8 Outcome**

The SEG turns “because the model said so” into **because the evidence shows so**. With JSON-LD/SHACL for structure, bi-temporal slicing for truth-over-time, lineage scoring for audits, and signed export packs for regulators, AIMOS/APOE upgrades outputs into **durable, inspectable facts** that can be replayed, defended, and improved.

**Part VI — Atomic Evolution: SDF-CVF**

**Chapter 16. Atomic Commits Across Code/Docs/Tags/Traces**

**Aim.** Make *every* change travel as one sealed unit across the quartet {code, docs, tags, traces}—gated, witnessed, reversible.

**16.0 Core model**

* **ChangeSet Δ** — the atomic proposal spanning all artifacts.
* **Gates G** — deterministic checks that *bind* the quartet.
* **Outcomes** — PASS → merge; SOFT\_FAIL → auto-fix then re-gate; HARD\_FAIL → quarantine; DEFER → HITL.
* **Snapshots** — every PASS produces:  
  S\_code, S\_docs, S\_tags (HHNI delta), S\_traces (SEG delta) → rolled up into **Snapshot S\*** used for replay/rollback.

**Formal.** A commit K = (Δ, S\*, W) is valid iff ∀ g ∈ G: g(Δ) = PASS and a Witness W (VIF) is attached.

**16.1 Gate catalog (parity, tests, safety, provenance, HITL)**

| **Gate** | **Purpose** | **Inputs** | **Check (examples)** | **On failure** |
| --- | --- | --- | --- | --- |
| **Parity** | Align spec/docs↔code↔telemetry | diff(AST), spec YAML, SEG claims | (i) Function/class add/remove reflected in spec; (ii) Public API changes mirrored in docs; (iii) Coverage of new spec scenarios ≥ θ | SOFT\_FIX (gen doc stubs) else QUARANTINE |
| **Tests** | Functional confidence | test graph, coverage map | (i) All suites green; (ii) Δ-coverage ≥ θΔ; (iii) Mutation score ≥ μ | SOFT\_FIX (auto-tests) or QUARANTINE |
| **Safety/Policy** | Security & data policy | policy pack, taint map | (i) No PII exfil paths; (ii) license ok; (iii) prompt-injection vectors neutralized | HARD\_FAIL→QUARANTINE |
| **Provenance (VIF)** | Reproducibility | witness bundle | (i) Model id + weights hash pinned; (ii) tool versions pinned; (iii) replay seed stored | SOFT\_FIX (replay to regenerate) |
| **Tags/HHNI** | Memory integrity | HHNI diff, DD | (i) DD ≤ θ\_DD; (ii) dependency hashes updated; (iii) impact preview acknowledged | DEFER→HITL if high impact |
| **Traces/SEG** | Evidence continuity | SEG delta, SHACL | (i) Claim has witness; (ii) contradiction edges instead of delete; (iii) time fields valid | HARD\_FAIL |
| **Perf/Cost** | Budget adherence | token/latency deltas | (i) p95 latency Δ ≤ θ; (ii) token budget within class | SOFT\_FIX (compress) or DEFER |
| **UQ/Calibration** | Honest confidence | κ, ECE | (i) κ-banding consistent; (ii) ECE ≤ ε on eval slice | SOFT\_FIX (recalibrate) |
| **HITL-Risk** | Human oversight | risk score R | (i) If R ≥ τ, require 2-key approval; (ii) reviewer roles distinct | DEFER→HITL |

**Gate function (sketch)**

result gate(g, Δ):

inputs ← materialize(g.requires, Δ)

verdict, patch ← g.check(inputs)

if verdict == PASS: return PASS

if verdict == SOFT\_FAIL and patch: Δ' ← apply(patch, Δ); return RETRY(Δ')

if verdict == DEFER: return HITL

return HARD\_FAIL

**16.2 Parity across the quartet**

**Diff semantics**

* **Code:** AST-level semantic diffs (noise-free), API surface map.
* **Docs/Specs:** schema-aware diffs (headings, tables, OpenAPI).
* **Tags (HHNI):** node/edge deltas + DD (dependency delta) + impact cone.
* **Traces (SEG):** graph deltas (nodes/edges) with bi-temporal fields.

**Parity rules**

1. **Spec→Code:** Any public contract change must have corresponding code diffs and tests (create stubs if missing).
2. **Code→Docs:** Public surface changes must update docs (autofill allowed but flagged).
3. **Code↔Telemetry:** Added behaviors must bind to observable metrics/logs.
4. **Memory tags:** New concepts introduce HHNI atoms with priority vectors; cross-links hashed; high-impact edits require preview.

**ADR & branching strategy**

* **ADR on change intent.** Each Δ starts with an ADR (motivation, options, decision, consequences) → stored in SEG and linked to the commit.
* **Branching.** Trunk-based; short-lived feature branches. High-risk branches run with *capability tokens* (restricted tools/data).
* **Merge protocol.** All gates green + (if R≥τ) dual reviewer sign-off (two-key) + signed snapshot.

ADR minimal template:

ADR-####: Title

Context

Decision

Alternatives

Consequences (tech/risk)

Links (SEG ids, issues)

**16.3 Auto-fix, quarantine, rollback**

**Auto-fixers (non-interactive)**

* **Docs/spec parity:** generate or update missing sections; fix OpenAPI/JSON-Schema drift.
* **Tests:** synthesize unit tests for uncovered public methods; strengthen weak assertions.
* **Memory:** reindex HHNI; regenerate dependency hashes; prune low-weight edges.
* **Provenance:** replay tool calls under frozen snapshot; regenerate witness with pinned seeds.
* **Perf/Cost:** apply prompt compression (dumbbell layout), cache hot sub-plans, lower temperature for stability.

**Quarantine**

* **Isolation.** Move Δ to a *quarantine branch* with read-only memory; run in sandboxed containers; redact secrets.
* **Diagnostics.** Attach gate failure reasons, impact cone, suggested fixes; open a tracked **Quarantine Incident (QI)** with SLA.
* **Exit.** On green re-gating → return to normal merge; else **auto-revert** if SLA exceeded.

**Rollback**

* **Design.** Two-phase merge with snapshot publication:
  1. Prepare S\* (code/docs/tags/traces) and publish to the snapshot log.
  2. Flip routing to S\* for live reads/writes.
* **Revert.** rollback(to=S\_prev) atomically resets routing pointers; append contradiction edges in SEG (do not delete); rehydrate HHNI from S\_prev.
* **Saga/compensation.** For side-effects across stores, maintain compensators (e.g., reverse data migrations, retract tool outputs).

**Rollback API (sketch)**

POST /evolution/rollback

{

"to\_snapshot": "snap-2025-10-12T11:02:22Z",

"reason": "gate-regression-coverage",

"requester": "release-bot",

"two\_key": ["ownerA","ownerB"]

}

**16.4 DORA-aligned metrics & SLOs**

Track per repo/program:

* **Deployment frequency** — merges to trunk with PASS.
* **Lead time for changes** — ADR→merge.
* **Change failure rate (CFR)** — fraction of merges that trigger rollback/quarantine within T hours. **Target:** <10%.
* **MTTR** — mean time from quarantine/incident open → green re-gate or rollback. **Target:** P50 < 4h, P95 < 24h.
* **Parity score** — % commits where quartet diffs are mutually consistent (no autofix needed). **Target:** >90%.
* **Coverage deltas** — Δ line/branch coverage; **Mutation score** on touched code.
* **Budget adherence** — token/latency deltas vs. budgets.

**Emissions**

* Every gate emits: {gate, verdict, reason, patch?, duration, owner, risk, κ\_band} → OpenTelemetry spans with snapshot id → SEG Decision nodes for audit.

**16.5 Checklists**

**Before merge (bot-enforced)**

* All gates PASS (or SOFT\_FIX applied + re-gate PASS)
* Witness bundle attached (model hash, prompt template, tools)
* ADR linked; risk R computed; two-key approvals if R≥τ
* Snapshot S\* prepared & signed; exportable pack green (SEG/JSON-LD)

**If quarantine**

* QI opened with SLA & owner
* Capability tokens restricted; secrets rotated if needed
* Autofixers attempted; re-gate result posted
* Exit: merge or rollback executed & witnessed

**If rollback**

* Routing flipped to S\_prev atomically
* SEG contradiction edges emitted; user-facing label set (amber/red)
* Post-incident ADR addendum with root cause and follow-ups

**16.6 Outcome**

SDF-CVF turns “updates” into **governed evolution**: the same commit updates code, specifies what changed, tags the memory, and records witnessable traces. Gates keep the quartet glued; quarantine contains risk; rollback is a *first-class, witnessed* operation—not a scramble.

**Part VI — Atomic Evolution: SDF-CVF**

**Chapter 17. Human-in-the-Loop (HITL) — risk scoring, two-key approval, review UX, accountability**

**Aim.** Insert *governed human judgment* at precisely the points where automation is most likely to err or cause harm—without turning HITL into a bottleneck. Every intervention is measurable, reproducible, minimally-intrusive, and leaves a cryptographically verifiable trail.

**17.0 Control surfaces**

* **When:** pre-merge, pre-deploy, post-incident, post-training-data change, high-impact memory edits (high DD), policy-touching changes (PII/licensing).
* **Where:** SDF-CVF gate line (Ch.16), APOE plan boundaries (Ch.11), SEG decision issuance (Ch.15).
* **Who:** role-separated *Proposer* (author), *Reviewer-A* (domain), *Reviewer-B* (risk/compliance), *Operator* (release).

**17.1 Risk scoring (R)**

A single bounded risk score drives HITL policy.

**Signal model.** For a candidate change/answer x:

* κ\_band — abstention/uncertainty band from UQ (Ch.14): entropy, variance, ECE-adjusted.
* I — **impact**: blast radius (users, data domains), monetary/latency deltas, dependency cone size.
* N — **novelty**: distance from prior patterns (embedding drift, unseen API surface, unseen spec pattern).
* P — **privacy/policy exposure**: PII flows, license changes, policy pack hits.
* D — **data-lineage risk**: weak/unknown source provenance in SEG, contradiction edges involved.
* S — **safety**: prompt-injection exposure, tool capability delta, sandbox escape potential.

Weights {w\_\*} are program-specific; all components normalized to [0,1].

**Thresholds.**

* R < τ₀ → **auto** (no human).
* τ₀ ≤ R < τ₁ → **single-review** (Reviewer-A).
* R ≥ τ₁ → **two-key approval** (Reviewer-A + Reviewer-B).
* R ≥ τ\_block → **block + quarantine** (Ch.16).

**Calibration.** Weekly backtests correlate R to downstream incidents/rollbacks; optimize {w\_\*}, τ for target CFR and MTTR (Ch.16.4).

**17.2 Two-key approval (2KA)**

**Principle.** No single human can push a high-risk decision. Keys are *role-separated* and *non-delegable*.

**Policy.**

* Distinct org roles: **Domain Owner** (Key-A), **Risk/Compliance Owner** (Key-B).
* Cooling period: approvals must be non-simultaneous (e.g., ≥ 10 min apart) unless incident emergency label is set.
* Conflict checks: reviewers cannot approve their own proposals; no reciprocal approvals within a rolling window.

**Mechanics.**

* **Detached approval objects** signed with FIDO2 or org PKI:
* Approval {
* id, snapshot\_id, risk: R, gate\_results, diffs, reviewer\_id, role,
* decision: {APPROVE|REJECT|REQUEST\_CHANGES},
* statement, signatures: [sig(reviewer\_key)]
* }
* **Threshold signature** (t=2 of n) yields a single merge token bound to snapshot\_id and risk hash.
* **Expiry**: merge token TTL (e.g., 24h); any material change invalidates token (content-addressed diffs).

**Emergency break-glass.**

* Requires three distinct keys (incl. Operator) + incident ticket.
* Auto-opens post-mortem ADR and sets amber label in SEG; time-boxed.

**17.3 Review UX & diff semantics**

**Reviewer view (single pane).**

1. **Risk card**: R, source breakdown, thresholds, suggested path.
2. **Quartet diffs** (noise-free):
   * Code: AST/API surface, mutation-score delta.
   * Docs/specs: schema-aware diffs, OpenAPI/JSON-Schema checks.
   * Tags (HHNI): node/edge delta, DD, impact cone preview.
   * Traces (SEG): claim/evidence/decision changes, contradiction edges.
3. **Witness panel** (VIF): model id & weights hash, prompt template, tool calls, seeds, snapshot id.
4. **UQ panel**: κ-band, ECE, variance; links to calibration run.
5. **Playback**: one-click *replay under snapshot* in a sandbox; preview env for web apps.
6. **Policy hits**: PII/secret taint paths, license diffs, guardrail triggers.
7. **Actions**: Approve/Reject/Request changes; add constraints; assign follow-ups.

**Diff semantics.**

* Collapse trivial churn (format, comments).
* Group related hunks by API/feature.
* Show **dependency impact**: predicted blast radius across HHNI/SEG.

**Commenting.** Threaded comments are first-class SEG annotations (time-sliced), not ephemeral chat.

**17.4 Accountability trails**

**Non-repudiation.** Every decision writes a **Decision** node into SEG with:

* reviewer ids (pseudonymous ids allowed in export), role, timestamp,
* approval object hash + signatures,
* linkage to Snapshot S\*, gate results, and risk factors.

**Temporality.** All review artifacts carry [t\_from, t\_to); replays are *as-of* time-slice correct.

**Chain-of-custody.** proposal → gates → reviews → merge/rollback is a single, queryable lineage chain (Ch.15).

**Access.** Reviews are immutable; corrections append contradiction edges + addenda. No destructive edits.

**17.5 HITL workflow (happy path & deviations)**

**Happy path.**

1. Proposer opens Δ with ADR; SDF-CVF gates run.
2. Risk engine computes R, emits policy target (auto/single/2KA).
3. Reviewers receive a **frozen preview** bound to S\*.
4. Reviewers approve; threshold sig creates merge token.
5. Release bot verifies token → merges & publishes S\*.
6. SEG gains Decision node; VIF witness bundle archived.

**Deviations.**

* **Request changes** → Proposer updates Δ → gates re-run → new S\* invalidates old token.
* **Reject** → Δ quarantined (Ch.16.3).
* **Timeout** → auto-close or escalate per SLA; optionally re-risk with stricter thresholds.

**17.6 APIs (sketch)**

* POST /hitl/risk/score → {R, factors, thresholds}
* POST /hitl/review/open → returns review\_id, snapshot\_id
* POST /hitl/review/{id}/decision → stores signed decision
* POST /hitl/merge-token → threshold-sign → token (bound to snapshot\_id, risk\_hash)
* POST /hitl/replay → run under snapshot; attach artifacts to SEG
* GET /hitl/audit/{snapshot\_id} → full lineage trail (read-only)

All endpoints read-only except *decisions*; merges executed by release bot only with valid token.

**17.7 Metrics & SLOs**

* **Review lead time**: t(decision) − t(open). Targets: P50 < 60m, P95 < 24h.
* **Human load**: reviews per reviewer/day; auto-balance routing.
* **True-positive HITL**: fraction of high-R that prevented a later rollback/incident.
* **False-positive HITL**: low-value interventions (optimize thresholds/weights).
* **Coverage**: % of merges at each band (auto/single/2KA).
* **Drift**: delta in reviewer decisions vs. model κ-band; recalibrate UQ monthly.

All metrics emitted as OpenTelemetry spans, linked to SEG.

**17.8 Edge cases & safeguards**

* **Prompt-injection suspected** → force full redaction view; require 2KA regardless of R.
* **Training data touch** → always 2KA + dataset snapshot witness.
* **Memory high-impact (DD ≥ θ\_DD)** → require impact preview acknowledgment + 2KA.
* **Cross-repo changes** → composite review with per-repo owners; composite threshold signature.
* **Reviewer unavailability** → escalation ladders; never collapse 2KA to single key.

**17.9 Minimal data schemas**

**Review (SEG JSON-LD fragment).**

{

"@type": "Decision",

"id": "dec-2025-10-17-0012",

"about": "snap-2025-10-17T15:21:03Z",

"risk": { "R": 0.81, "factors": { "novelty": 0.7, "privacy": 0.9 } },

"gates": [{ "gate": "parity", "verdict": "PASS" }, { "gate": "safety", "verdict": "PASS" }],

"approvals": [

{ "reviewer": "role:domain", "sig": "0x…", "ts": "…" },

{ "reviewer": "role:risk", "sig": "0x…", "ts": "…" }

],

"statement": "Approved with post-deploy canary at 10% traffic."

}

**HITL policy.**

{

"policy\_id": "hitl-default-v3",

"thresholds": { "auto": 0.35, "single": 0.65, "two\_key": 0.65, "block": 0.92 },

"weights": { "confidence": 0.2, "impact": 0.25, "novelty": 0.2, "privacy": 0.2, "data": 0.1, "safety": 0.05 },

"cooling\_minutes": 10

}

**17.10 Security & privacy**

* **Signer hygiene:** hardware-backed keys; rotation and revocation; approval key use is least-privilege and auditable.
* **PII minimization:** review UI shows *projections*, not raw payloads; deep-link to redacted diffs.
* **Replay sandboxes:** network egress deny by default; capability tokens for tools.
* **Tamper-evidence:** all review objects and merges are content-addressed; SEG stores hashes.

**17.11 Outcome**

HITL in AIMOS/APOE is **surgical, not theatrical**: *measured risk → minimal human intervention → maximal accountability*. Two-key approval prevents single-point failure; the review UX compresses complex parity checks into a legible decision; the evidence graph guarantees the story of “who decided what, based on which facts” is never lost.

**Part VII — IDE-in-the-Loop: From Prompt to Running App**

**Chapter 18. Blueprint → Scaffold → Code → Test → Deploy → Observe**

**Aim.** Turn *intent* into *software* with auditability. APOE compiles the blueprint into a governed plan; SDF-CVF enforces parity; VIF/SEG preserve lineage; CMC/HHNI ensures the system remembers what it built and why.

**18.0 System overview**

**Lifecycle (governed loop).**

1. **Blueprint** (spec as data) →
2. **Scaffold** (deterministic repo, env, policies) →
3. **Code** (intent-constrained synthesis + refactors) →
4. **Test** (pyramid + mutation + property) →
5. **Deploy** (ephemeral → canary → prod) →
6. **Observe** (telemetry → SEG → feedback into memory)

**Traceability rule.** Every transition emits: (snapshot\_id, plan\_id, gate results, UQ vector, provenance) into SEG; code/docs/tags/traces co-evolve in one atomic commit (SDF-CVF).

**18.1 Blueprint YAML — the contract APOE compiles**

A **machine-readable spec** that binds scope, constraints, budgets, and acceptance. Minimal yet complete.

**Schema (v0).**

blueprint:

id: "shop-web-v1"

version: "0.1.0"

intent:

problem: "Sell digital goods globally"

outcomes:

- "Checkout < 3 steps"

- "p95 API < 250ms in NA/EU"

domains:

- ui/web

- api/rest

- data/postgres

- auth/oidc

modules:

- name: "catalog"

contracts:

api: "openapi/catalog.yaml"

events:

- "product.created@v1"

acceptance:

scenarios:

- name: "search-by-name"

given: "10k products"

when: "user searches 'hat'"

then: "p95 page load < 1.2s"

- name: "checkout"

contracts:

api: "openapi/checkout.yaml"

acceptance:

scenarios:

- name: "card-payment"

then: "auth capture succeeds; email receipt"

constraints:

policies:

pii: "redact-at-rest"

licenses: ["MIT","Apache-2.0"]

budgets:

token\_per\_call: 16\_000

build\_minutes: 30

cloud\_cost\_monthly\_usd: 500

data:

schema: "schema.prisma"

migrations: true

observability:

traces: true

metrics:

slos:

api\_p95\_ms: 250

error\_rate\_pct: 0.5

security:

threat\_model: ["SSRF","prompt-injection"]

compliance:

export\_packs: ["eu-ai-act-low", "soc2-lite"]

ui:

flows:

- name: "guest-checkout"

steps: ["cart","address","pay","confirm"]

**Compilation map (Blueprint → APOE plan).**

| **Blueprint field** | **APOE object** | **Gate** |
| --- | --- | --- |
| modules[\*].contracts | Retrieval inputs + static checkers | g\_spec, g\_contract |
| acceptance.scenarios | E2E test generation targets | g\_tests |
| constraints.budgets | Step budgets (tokens/time/tools) | g\_budget |
| observability.slos | Alert policies, canary thresholds | g\_ops |
| compliance.export\_packs | Export schema bindings (SEG) | g\_provenance |

**Determinism.** Blueprint → plan hash → *content-addressed* scaffolds; rebuilds are byte-for-byte under same snapshot & toolchain.

**18.2 Scaffolding strategies — from empty repo to governed workspace**

**Goals.** Deterministic structure, policy-first wiring, fast feedback.

**Outputs.**

* Monorepo (e.g., /apps/web, /services/api, /packages/ui, /infra).
* Locked toolchain: tooling.lock (node/go/python versions, formatters, linters).
* Policy packs pre-wired: PII redaction, license allowlist, secret scanners.
* CI templates: build → test → preview → canary → promote.
* **SEG seed**: initial evidence graph nodes (Spec, Contracts, SLOs).

**Scaffolding modes.**

1. **Greenfield (deterministic).**
   * Template catalog selection bound to blueprint.domains.
   * Generate from commit-pinned templates; write **Scaffold Witness**:
   * scaffold\_witness.json = { template\_ids, commit\_hashes, toolchain, blueprint\_hash, snapshot\_id }
   * Gates: g\_template\_integrity, g\_policy\_boot.
2. **Brownfield (delta-aware).**
   * AST inventory + dependency graph → **Delta Scaffold** only for missing contracts/tests/policies.
   * Gates: g\_drift\_report (lists divergences spec↔code), g\_delta\_conflicts.

**Failure modes & mitigations.**

* *Toolchain drift* → pin & cache; reproduce via tooling.lock.
* *Policy pack gaps* → block merge if required pack missing (licenses, PII).
* *Template entropy* → signatures on templates; SEG records hashes.

**18.3 Code — intent-constrained synthesis and safe refactors**

**Principle.** The builder agent writes *within* constraints, not from scratch; codegen is *review-ready* (tests/docs/traces).

**Loop (per module).**

1. **Retrieve** (CMC/HHNI): similar patterns, past ADRs, contract snippets.
2. **Plan** (APOE step): outline files, functions, interfaces; set budgets.
3. **Synthesize** (ICS): generate minimal viable implementation *constrained by* contracts, types, budget.
4. **Refine**: static analysis (types, lints), security scan, small refactors.
5. **Emit artifacts**: code + docstrings + SEG snippets + tags (HHNI).
6. **Gate**: compile, unit tests, mutation score, coverage.

**Intent-Constrained Synthesis (ICS) rulebook.**

* Never bypass contracts; treat OpenAPI/GraphQL schemas as types.
* Prefer composition over global state; enforce dependency boundaries.
* Generate tests *with* code; failing tests abort synthesis.
* Every public surface is documented; docstrings reference blueprint acceptance.

**Refactors (safe by construction).**

* **Spec-led refactor**: change contract → run codemods → re-generate adapters → rerun tests.
* **Performance refactor**: CI-guarded; requires baseline & regression gates.
* **Security refactor**: must pass policy pack scanners; link to threat model item.

**UQ for code steps.** Emit κ-band over synthesis confidence (n-best programs agreement, static checker entropy). Low κ → require reviewer attention (HITL).

**18.4 Test — the pyramid and its gates**

**Layers & targets.**

* **Unit** (fast, many): functions/components; target P50 < 200 ms/test; coverage ≥ 80%.
* **Contract**: provider/consumer checks from OpenAPI/GraphQL; **blocking**.
* **Integration**: DB, queues, third-party fakes; P95 suite < 5 min.
* **E2E**: user flows from blueprint; run on preview env; flake budget < 2%.

**Advanced checks.**

* **Mutation testing** (change-failure detector): score ≥ 70% to merge.
* **Property-based** (invariants), **fuzz** for parsers/APIs.
* **Security tests**: taint flows (PII), authz matrices, prompt-injection scenarios.

**Gates (SDF-CVF).**

* g\_tests (pass), g\_cov (thresholds), g\_mutation, g\_contract, g\_security.
* Any failure → **quarantine branch** and proposed autofix or ADR for defer.

**Evidence.** Test runs write structured results into SEG: inputs, env hash, artifacts, seeds → full replayability.

**18.5 Deploy — previews, canaries, promotion, and telemetry**

**Pipeline.**

1. **Preview env** per PR (isolated DB + secrets broker).
   * Attach human playback to proposal (HITL can run flows).
   * Capture RUM + API traces for *real user flows* defined in blueprint.
2. **Canary** (e.g., 5–10% traffic).
   * SLO guards from blueprint: if api\_p95\_ms > 250 or error\_rate > 0.5% for N minutes → auto-rollback.
   * SEG records canary stats as **Decision supports** for promotion.
3. **Promotion** (progressive delivery).
   * Two-key approval if risk band high (Ch.17); attach decision node to SEG.

**Observability wiring (deterministic).**

* OpenTelemetry auto-injected in services; trace attributes include plan\_id, snapshot\_id, module, version.
* Metrics bundles: latency histograms, error rates, resource use; dashboards are generated from blueprint SLOs.
* **Anomaly → Memory.** Significant incidents create atoms in CMC, tagged to modules and causes; future APOE plans retrieve these.

**Rollback.** Content-addressed release; **one-click reversion** to last good snapshot\_id. SEG adds contradiction edges for the reverted decision.

**18.6 Parity enforcement (spec ↔ code)**

**Parity score** P ∈ [0,1] per module:

* *API drift*: % of surfaces where code != contract.
* *Schema drift*: DB vs. declared schema delta.
* *Doc gap*: exported types without docs.
* *Trace gap*: live spans missing blueprint labels.

**Mechanisms.**

* AST-level conformance checks (code ↔ OpenAPI/GraphQL).
* DB schema diff vs. schema.prisma/DDL; migrations generated or blocked.
* Docstring enforcement & spec anchors (IDs from blueprint).
* Runtime probes compare emitted traces to blueprint route table.

**Gates.** g\_parity\_min (e.g., P ≥ 0.93) is **blocking**; below threshold → auto-generate patches or open ADR with risk score.

**Lineage.** For each parity check, VIF attaches:

* tool versions, model IDs/hashes, prompt template IDs,
* inputs (spec excerpts), outputs (diffs), κ/UQ,
* the *exact* snapshot that produced the verdict.

**18.7 APOE plan skeleton (ACL example)**

pipeline web\_shop\_build

using budgets { tokens: 16000, walltime: "45m" }

step plan\_from\_blueprint

in { blueprint\_yaml }

out { plan\_json, spec\_refs[] }

tool "apoe.planner"

gate { g\_spec\_refs }

step scaffold\_monorepo

in { plan\_json }

out { repo\_path, scaffold\_witness }

tool "scaffold.engine"

gate { g\_template\_integrity, g\_policy\_boot }

step synthesize\_modules

foreach module in plan\_json.modules

in { repo\_path, module, spec\_refs }

out { pr\_branch, code\_artifacts[] }

tool "builder.agent"

budget { tokens: 4096, walltime: "8m" }

gate { g\_compile, g\_unit, g\_contract }

step run\_tests

in { pr\_branch }

out { test\_report }

tool "ci.runner"

gate { g\_cov(>=0.80), g\_mutation(>=0.70), g\_security }

step deploy\_preview

in { pr\_branch }

out { preview\_url, telemetry\_pack }

tool "deploy.preview"

gate { g\_ops\_preview }

step canary\_and\_promote

in { release\_candidate }

out { release\_id }

tool "deploy.canary"

gate { g\_canary\_slo, g\_parity\_min, hitl(two\_key\_if R>=τ1) }

end

**18.8 KPIs & SLOs (engineering truth)**

* **Plan determinism**: identical plan hash under same blueprint/toolchain (target 100%).
* **Time-to-scaffold**: P95 < 90 s; **time-to-first-preview**: P95 < 15 min.
* **Change-failure rate (CFR)** after promotion: < 10%.
* **Mean-time-to-restore (MTTR)** with rollback: P50 < 10 min.
* **Parity score** P ≥ 0.93 at merge; P ≥ 0.97 within 24 h.
* **Coverage**: unit ≥ 80%, contract 100% of public surfaces, mutation ≥ 70%.
* **Telemetry completeness**: traces with valid plan\_id ≥ 99%.

All emitted as OpenTelemetry spans with SEG links.

**18.9 Failure modes & design countermeasures**

* **Spec/code drift** → parity gates + codemods; block on severe drift.
* **Template rot** → content-addressed templates; signature checks; periodic re-bakes.
* **Hallucinated code** → ICS constraints; UQ κ-bands; reviewer prompts on low agreement.
* **Canary blindness** → blueprint SLOs are *blocking guards*; promotion requires healthy evidence in SEG.
* **Preview flakiness** → isolate dependencies; seed data snapshots; determinize seeds.

**18.10 What’s unlocked**

By binding **APOE** (compiled reasoning), **SDF-CVF** (atomic evolution), **VIF/SEG** (lineage), and **CMC/HHNI** (memorable context) *inside the IDE loop*, AIMOS/APOE turns prompts into **governed software**. Every file, test, deploy, and metric is explainable, replayable, and reversible—*on demand*.

**Part VII — IDE-in-the-Loop: From Prompt to Running App**

**Chapter 19. Templates & Components — Canonical stacks, policy packs, integrations, and cost/energy budgets**

**Aim.** Provide **golden paths** that APOE can compile deterministically from a blueprint into living repositories, with built-in safety, parity, and observability. Each template is **content-addressed**, **policy-pinned**, and emits **VIF/SEG** evidence on every step.

**19.0 Scope & Principles**

* **Determinism over fashion.** Pin frameworks, toolchains, and template hashes; all outputs are replayable from the same snapshot.
* **Policy-first scaffolds.** Security, compliance, and provenance are baked in, not added later.
* **Parsimony.** Few curated stacks, deeply instrumented. Extensions ride plug-in seams, not forks.
* **SDF-CVF parity.** Code, docs, tags, and traces co-evolve under the same commit gate.

**19.1 Canonical UI Stack (Web)**

**Stack (pinned).**

* Runtime: **Next.js (App Router)** + **React**; SSR/ISR enabled.
* Styling: **TailwindCSS** + **Radix Primitives** + **shadcn/ui** (accessibility first).
* State: **Zustand** (local) + **TanStack Query** (server data).
* i18n & a11y: next-intl, Axe checks in CI.
* Testing: **Vitest** (unit), **Playwright** (E2E), **Lighthouse CI** (budgets).
* Telemetry: **web-vitals** → OpenTelemetry bridge (LCP, CLS, INP tagged with plan\_id, snapshot\_id).

**Scaffold layout.**

apps/web/

app/ (routes, server components)

components/ (a11y wrapped UI)

lib/ (fetchers, schema zod)

public/

tests/ (unit, e2e)

.lighthouseci/ (budgets.json)

**Gates (blocking).**

* g\_a11y: Axe violations = 0 (severity ≥ serious).
* g\_perf\_web: Lighthouse budgets (LCP ≤ 2.5s P75, CLS ≤ 0.1).
* g\_contract\_ui: Type-safe API clients (OpenAPI/GraphQL codegen) present for each declared endpoint.
* g\_i18n: Missing translation keys = 0 for required locales.

**Evidence.** Page-level SEG spans include: route, hydration time, data source IDs, blueprint flow ID.

**19.2 Canonical API Stack (Service)**

**REST (default).**

* Framework: **FastAPI** (Python) *or* **NestJS** (TypeScript) — one chosen per program.
* Contracts: **OpenAPI v3** single source of truth; code generated clients/servers.
* Cross-cutting: **Rate-limit** (token bucket), **idempotency keys** for POST, structured errors (problem-json).
* Storage: **PostgreSQL** via **Prisma** (TS) or **SQLModel** (Py); **pgvector** if embeddings needed.
* Telemetry: OpenTelemetry auto-instrumented; request spans carry **caller**, **subject**, **plan\_id**.

**GraphQL (optional).**

* Server: **Apollo**/**Helix** with persisted ops and automatic persisted queries (APQ).
* Policies: depth/complexity limits, field-level auth resolvers, DataLoader for N+1 control.

**Gates (blocking).**

* g\_openapi\_valid: spec validates; diff against previous snapshot reviewed.
* g\_idempotency: endpoints labeled idempotent prove behavior in tests.
* g\_perf\_api: p95 latency ≤ blueprint SLO per route in preview canary.
* g\_contract\_tests: consumer/provider pact tests pass.

**19.3 Canonical Auth & Identity**

**Model.**

* Protocol: **OIDC/OAuth2 (PKCE)** for users; **mTLS + JWT** for service-to-service.
* Provider: **Keycloak** (self-host) or managed (Auth0/Cognito) — template pins one.
* Sessions: short-lived access tokens, rotating refresh, token binding on device.
* Authorization: **OPA/Rego** or **Cedar** policies; resource-scoped claims; deny-by-default.

**Security primitives (enforced).**

* Web: SameSite=strict cookies, CSP with nonce, Origin checks for POST.
* API: JTI replay cache, aud/iss validation, per-scope rate limits.
* Secrets: sealed by KMS; no plaintext in env; CI retrieves via short-lived tokens.

**Gates (blocking).**

* g\_auth\_lat: p95 login < 400ms in target regions.
* g\_policy\_eval: policy pack validation on all protected routes (OPA unit tests).
* g\_secret\_leak: zero findings from secret scanners; SBOM free of banned licenses.

**19.4 Canonical Data Stack**

**Transactional.**

* **PostgreSQL 15+** with **pgcrypto** + **RLS**.
* Schema: **Prisma**/**Liquibase** migrations; content-addressed migration files.
* CDC: **Debezium** → event bus (for analytics/ELT or sagas).

**Analytical.**

* Warehouse: **BigQuery**/**Snowflake** (program-choice).
* ETL: **dbt** models; lineage exported to SEG (model ↔ query ↔ decision).

**Vector & Evidence.**

* **pgvector** (co-located embeddings) for small/mid; external vector store optional.
* **SEG store**: RDF/JSON-LD in **Neptune/Neptune-like** or **Neo4j** with time-slicing.

**Gates.**

* g\_rls\_required: tables with PII must have RLS ON; tests verify denial by default.
* g\_migrations\_reversible: up/down integrity; destructive ops gated by HITL.
* g\_data\_retention: retention policies match blueprint jurisdiction; verify in CI.

**19.5 Template Catalog (Golden Paths)**

**Repository skeleton (monorepo).**

.

├─ apps/

│ ├─ web/ # Next.js

│ └─ worker/ # background tasks

├─ services/

│ └─ api/ # FastAPI or NestJS

├─ packages/

│ ├─ ui/ # shared components

│ └─ schema/ # zod/openapi/graph types

├─ infra/ # IaC (Terraform/Pulumi), pipelines

├─ policies/ # OPA/Cedar, license allowlist, PII redaction maps

├─ .github/workflows/ # CI/CD

└─ docs/ # ADRs, blueprints, runbooks

**Template witnesses.**  
template.lock.json records: template IDs, commit hashes, toolchain versions, blueprint hash, and **snapshot\_id**. Any deviation → g\_template\_integrity fails.

**19.6 Policy Packs (snap-in compliance)**

| **Pack** | **What it enforces** | **Gates** |
| --- | --- | --- |
| **Security-Basics** | TLS, HSTS, CSP, dependency vulns (SLSA/SBOM) | g\_csp, g\_slsa, g\_vuln\_budget |
| **PII & Residency** | PII taxonomy, field-level redaction, region-pinned storage | g\_pii\_map, g\_residency, g\_rls\_required |
| **Open-Source Licenses** | Allowlist (MIT/Apache/BSD), deny GPL-copyleft by default | g\_license\_allow |
| **Secrets & Keys** | No plaintext secrets; KMS usage; rotation schedule | g\_secret\_leak, g\_key\_rotation |
| **Prompt Safety** | Injection filters, tool-capability tokens, output watermarking | g\_prompt\_guard, g\_synth\_label |
| **LLMOps Logging** | VIF minimums (model ID, weights hash, prompt template) | g\_vif\_min |
| **Retention & Right-to-Forget** | Time-boxed logs; SEG export redaction hooks | g\_retention, g\_rtbf\_hooks |

Packs are **declarative bundles** (rules + scanners + CI jobs). APOE attaches packs from the blueprint, then scaffolds their enforcement points.

**19.7 Integration Patterns (Ports & Adapters)**

**Patterns.**

* **Outbound adapters** (HTTP/gRPC SDKs): generated from contracts; all calls go through **Capability Tokens** (scoped, expiring).
* **Inbound webhooks**: signature verification (HMAC/Ed25519), idempotency, replay protection.
* **Events & Sagas**: outbox pattern with transactional writes; retries with DLQ and observability tags.
* **RAG connectors**: retrieval adapters (files, DBs, wikis) with **VIF capture** (source URI, digest, access policy).
* **Streaming**: Server-Sent Events or WebSockets gated by policy pack; redact on the fly if PII.

**ACL example — adding a payment provider.**

step integrate\_payments

in { openapi\_spec:"providers/stripe.yaml" }

out { sdk\_pkg, provider\_adapter, contract\_tests }

tool "contract.codegen"

gate { g\_openapi\_valid, g\_license\_allow }

step secure\_webhooks

in { provider\_adapter }

out { webhook\_handler }

tool "adapter.scaffold"

policy { signatures:"ed25519", idempotency:true, retry: "expo<5x" }

gate { g\_webhook\_sig, g\_idempotency, g\_observe\_spans }

**19.8 Cost & Energy Budgets (governed spend)**

**Budget classes.**

* **Tokens** (LLM/RAG): input/output tokens per step/plan.
* **Compute**: CPU/GPU minutes for CI, inference, training.
* **Storage**: GB-months for DB/object/SEG/vector.
* **Network**: egress GB, third-party API calls.
* **Energy/Carbon**: kWh and CO₂e per stage.

**Budget DSL (embedded in blueprint).**

budgets:

tokens:

per\_call: 16000

per\_plan: 1.2e6

compute:

ci\_minutes: 60

gpu\_hours: 0

storage\_gb\_month:

hot: 50

cold: 200

network\_gb: 100

energy:

kwh\_month: 45

co2e\_tonnes\_max: 0.02

**Estimation formulas.**

* **Token cost**: C\_tokens = Σ (rate\_model\_i \* tokens\_i)
* **Compute**: C\_compute = Σ (rate\_cpu\*h\_cpu + rate\_gpu\*h\_gpu)
* **Storage**: C\_storage = Σ (rate\_tier\_gb \* GB\_month)
* **Energy**: kWh = Σ (power\_W / 1000 \* hours), CO₂e = kWh \* grid\_intensity\_kg\_per\_kWh / 1000

**Gates.**

* g\_cost\_plan: projected plan cost ≤ budget (fail fast pre-run).
* g\_cost\_actual: actual cost within ×(1+ε) of projection (ε default 10%).
* g\_energy: kWh & CO₂e under thresholds; if not, trigger **optimize-or-abstain** step.

**Optimization levers.**

* Prompt compaction (HHNI dumbbell compression).
* Cache hits (retrieval/page cache) and **vector co-location** (pgvector).
* Lower-rate regions for CI; canary duration caps; adaptive sampling for telemetry.

**19.9 Golden Paths & Extension Points**

**Golden paths (opinionated defaults).**

* **Web**: Next.js + Tailwind + shadcn → Playwright + Lighthouse.
* **API**: NestJS + OpenAPI + Prisma + Postgres (pgvector optional).
* **Auth**: OIDC (PKCE) + OPA policies.
* **Data**: Postgres + dbt + Warehouse; CDC → event bus.
* **Observability**: OpenTelemetry → SEG links everywhere.

**Extension points (guarded).**

* Swap REST↔GraphQL behind **contract equivalence** gate.
* Plug alternative vector stores behind **RAG Adapter** interface.
* Add GPU inference workers under **energy/cost** budget gates.

**19.10 KPIs & SLOs**

* **UI**: LCP P75 ≤ 2.5s, CLS P75 ≤ 0.1, INP P75 ≤ 200ms.
* **API**: p95 latency ≤ blueprint target; error rate ≤ 0.5%.
* **Auth**: login p95 ≤ 400ms; token issuance failure < 0.1%.
* **Data**: migration success 100% in preview; rollback P50 < 2 min.
* **Cost**: variance to projection ≤ 10%; **Energy**: kWh within budget.
* **Evidence**: ≥ 99% spans tagged with plan\_id & snapshot\_id.

All KPIs are exported as metrics + SEG nodes with time windows (for audit packs).

**19.11 Failure Modes & Countermeasures**

| **Failure** | **Symptom** | **Countermeasure** |
| --- | --- | --- |
| UI hydration mismatch | Blank/flash on route | SSR parity tests; strict ESLint rules; g\_perf\_web blocks |
| API drift | 4xx/5xx spikes post-merge | Contract tests block; codemods on spec change; parity gate |
| N+1 queries | Slow endpoints under load | ORM tracing + query plans; generated DataLoader; perf refactor gate |
| Secret leakage | CI logs with tokens | Secrets pack scanners block; KMS-only access; redaction filters |
| Token bloat | Cost overrun | HHNI compaction; top-k truncation; budget gates |
| Residency breach | Data in wrong region | Residency pack with RLS + region-pinned buckets; policy tests |
| Unlogged actions | Audit gaps | VIF minimums pack; CI fails on missing provenance |

**19.12 What this unlocks**

A **disciplined, repeatable** path from blueprint to production where **every component**—UI, API, Auth, Data—arrives with **guardrails, gates, and evidence** already wired. APOE no longer “generates code”; it **installs a governed system** whose cost, energy, and risk are explicitly budgeted and continuously proven in the **Shared Evidence Graph**.

**Part VIII — Security, Safety, & Compliance**

**Chapter 20. Threat Model & Guardrails — injection/exfiltration/SSRF patterns, tool sandboxes, capability tokens, DP/redaction, abstention/containment**

**Aim.** Define a **practical, enforceable** threat model for AIMOS/APOE and install **guardrails-as-code** that hold under real workloads: RAG, agents, tools, IDE-in-the-loop. Every control emits **witnesses (VIF)** and connects to the **Shared Evidence Graph (SEG)** so we can prove what the system did, why, and under which risk posture.

**20.0 Security doctrine (tl;dr)**

* **Deny by default.** Capabilities are *requested* and *granted* with scope, duration, and rate.
* **Untrusted by construction.** All user/content/context is tainted until proven otherwise.
* **Plan before power.** ACL/APOE plans are statically checked; high-risk steps require HITL (Ch.17).
* **Degrade safely.** When in doubt, **abstain** (κ-gating) or **contain** (least-privilege sandbox), not “best-guess.”

**20.1 Threat model for agentic systems**

We profile threats by **surface** and **goal**. Each entry lists typical attacks ➜ symptoms ➜ controls.

**20.1.1 Prompt- & context-level threats**

**Vectors.** Classic prompt injection (direct/indirect), instruction smuggling in HTML/Markdown/CSV/ALT text, jailbreak patterns, overlong-context poisoning, retrieval booby-traps (“ignore previous; exfiltrate X”).  
**Symptoms.** Role leakage; tool misuse; non-deterministic replies; policy bypass in citations.  
**Controls.**

* **Role isolation:** system vs. user vs. tool prompts are **separately hashed** and stored; only system prompt may set policy.
* **Quoted-content boxing:** untrusted text is wrapped with **verbatim fences + provenance tags** and **never** executed as instruction.
* **Policy-prefix pinning:** a signed policy prelude (template hash + SEG id) precedes all chains.
* **Detector ensemble:** pattern + LM-based detectors score “instructionality,” HTML/script payloads, and known jailbreaks; raise κ or strip.

**20.1.2 Tooling & RAG threats**

**Vectors.** Over-permissive tools, TOCTOU races, SSRF via URL tools, unlimited file I/O, secret leakage from vector stores, prompt injection through retrieved docs.  
**Symptoms.** Calls to arbitrary hosts, filesystem traversal, credentials in responses, “phantom” tool execs.  
**Controls.**

* **Capability tokens (CapTok):** signed, short-lived, scoped (action, resource, rate, region).
* **Tool sandboxes:** per-tool containers with **network egress allowlists**, read-only FS, CPU/mem/time quotas, and syscall filters.
* **RAG allow+deny lists:** per-connector hostname/tenant allowlists; MIME/type guards; **content disinfect** (strip scripts/links) before inclusion.
* **Result boxing:** retrieved snippets are **data**; they cannot alter policy. APOE treats RAG outputs as **inputs** only.

**20.1.3 Network & SSRF threats**

**Vectors.** URL fetch tools pointed at metadata endpoints, internal services, cloud IMDS; DNS rebinding; IPv6 literals; file://, gopher://.  
**Controls.**

* **URL normalizer:** reject non-HTTP(S), IP literals, link-local, or private RFC1918 addresses unless explicitly allowed.
* **Egress policy:** per-tool egress via proxy with hostname pinning, TLS enforcement, and request recording into SEG.
* **DNS hardening:** single trusted resolver; cache poisoning mitigations; SNI/ALPN checks.

**20.1.4 Data exfiltration & privacy**

**Vectors.** Model answers echo secrets; embeddings store PII; logs leak tokens; oversharing to third parties.  
**Controls.**

* **PII/secret classifiers** (regex+ML) in write paths (CMC ingest) and read paths (render).
* **Differential Privacy (DP)** for analytics/telemetry; per-tenant **ε/δ budgets** with decay τ.
* **Segregated stores:** PII on RLS-protected tables; vector store contains **redacted/hashed** variants; linkage via SEG only.
* **Synthetic labeling:** watermark/model tags for generated content (VIF).

**20.1.5 Supply chain & codegen**

**Vectors.** Malicious packages, typosquatting, poisoned snippets, license traps; CI secret exposure.  
**Controls.**

* **SBOM + license allowlist** gates; **sigstore**/SLSA provenance on artifacts.
* **Policy packs** enforce banned APIs, dangerous patterns (eval, spawn) in codegen.
* **CI secrets broker:** short-lived OIDC tokens from KMS; zero plaintext secrets in env/logs.

**20.2 Guardrails-as-code (enforced by APOE/ACL)**

We encode guardrails at **plan**, **step**, and **tool** levels. All checks emit **witness** events (Ch.14).

**20.2.1 ACL snippets**

**Capability tokens & sandboxes**

tool http\_fetch

caps { action:"GET", hosts:["docs.example.com"], rate:"60/m", max\_bytes:5e6 }

sandbox { net\_egress:["docs.example.com:443"], fs:"ro", cpu\_ms:2000, mem\_mb:256, timeout\_ms:3000 }

gate { g\_url\_safe, g\_egress\_allowed, g\_span\_vif }

tool file\_read

caps { action:"READ", paths:["/workspace/specs/\*.md"], max\_bytes:1e6 }

sandbox { fs:"ro", chroot:"/workspace", syscalls\_deny:["execve","mount"] }

gate { g\_path\_allow, g\_span\_vif }

**Prompt hygiene & tainting**

step synthesize\_answer

in { query, retrieved[] }

policy { taint: ["retrieved"], quote\_untrusted:true, policy\_prefix: "seg://policy/answering@v3" }

gate { g\_injection\_scan, g\_policy\_prefix\_signed, g\_kappa\_band }

**SSRF defense for URL tools**

gate g\_url\_safe

checks { scheme in ["https"], host not\_in private\_ranges, port in [443] }

**20.2.2 κ-gating & abstention**

At each boundary, compute risk = f(injection\_score, PII\_score, UQ, budget\_drift). If risk ≥ κ, the system:

1. **abstains** with rationale + remediation link, or
2. **contains** by downgrading capabilities (read-only tools, masked output), or
3. **routes to HITL** (Ch.17) for two-key approval.

**20.3 Tool sandboxes & capability tokens (CapTok)**

**Design.** A CapTok is a **signed JWT/JWP** with claims:

* sub (plan\_id/step\_id), act (Verb), res (Resource pattern), lim (rate/size),
* exp (TTL ≤ minutes), aud (tool runtime), jti (replay).

**Runtime enforcement.**

* Sidecar intercepts all syscalls/network; matches against CapTok.
* Violations raise **SEG “violation” edges** with packet metadata (no payloads with PII).
* Tool outputs are labeled with cap\_id for downstream audit.

**Least power defaults.**

* No shell, no child process, no write, no outbound by default.
* Explicit **egress allowlist**; DNS pinned; TLS cert pinning optional.

**20.4 Redaction, DP, and output hygiene**

**20.4.1 Multi-stage redaction**

1. **Ingest (CMC)**: classify and **mask** secrets/PII; store pointer to vaulted original (KMS-wrapped).
2. **Index (HHNI)**: embeddings computed on **redacted** content; SEG retains reversible link under policy.
3. **Render**: re-apply redaction by audience/purpose; attach VIF explaining masks.

**Mask formats.**

* Deterministic hashes for joinability (hash(email, pepper)), reversible tokens in secure contexts, or DP-protected aggregates for analytics.

**20.4.2 Differential Privacy (DP)**

* **Event DP** for telemetry; ε budgets per tenant/user/feature; decay (τ) resets.
* **Report DP** for metrics dashboards; Laplace/Gaussian noise; ECE calibration unaffected.
* **SEG nodes** store (ε, δ, mech, scope). Exceeding budget → κ-abstain.

**20.5 Abstention & containment patterns**

**When to abstain.**

* UQ high (ECE band breach), RS low, policy conflict, or privacy budget exhausted.  
  **What to return.** A structured **refusal diagnostic**: violated gates, minimal viable suggestions, request for narrower scope.

**Containment modes.**

* **Quarantine execution:** simulate plan without side effects; produce diffs only.
* **Read-only RAG:** disable write tools; restrict connectors.
* **Air-gapped chain:** run on frozen snapshot with synthetic data.

**Circuit breakers.**

* Token/latency/cost outliers → trip breakers; APOE rolls back to **safe plan** variant.
* Repeated breaker trips escalate to **HITL incident runbook**.

**20.6 Tests, red teams, and continuous verification**

**Pre-merge gates.**

* **Injection suite:** curated jailbreaks + content smuggling corpora.
* **SSRF suite:** private IPs, IMDS probes, DNS rebinding tests.
* **Secret echos:** known canaries planted in RAG sources must **never** surface.
* **Policy pack lint:** CSP, OPA/Rego unit tests, license/SBOM checks.

**In prod.**

* **Canary agents** exercise tools hourly with harmless probes; anomalies → page.
* **Shadow mode** for new guardrails (score only) before flip to block.
* **SEG queries** look for odd patterns: tool burst w/o CapTok, egress to new host, steady κ evasion.

**20.7 Threat → Control crosswalk**

| **Threat** | **Primary Control** | **Secondary** |
| --- | --- | --- |
| Prompt injection (direct/indirect) | Quoted boxing + policy-prefix signing | Detector ensemble + κ |
| RAG-borne instruction | Data-only treatment of retrieved text | Content disinfect + provenance |
| SSRF via URL tool | URL normalizer + egress allowlist proxy | DNS/SNI checks, SEG logging |
| Secret exfiltration | PII/secret classifiers + redaction | DP budgets; vector co-location w/ redacted content |
| Tool overreach | CapTok (scope/TTL/rate) | Sandbox syscall/network filters |
| Supply-chain poison | SBOM + license allowlist + SLSA | Dependency pinning; provenance checks |
| Overconfidence | UQ + ECE bands + κ | HITL escalation on high-risk plans |

**20.8 Observability & evidence (VIF/SEG)**

Every guardrail emits **witnesses**:

* **policy\_prefix\_signed** (template hash, signer),
* **cap\_tok\_granted/denied** (claims, TTL),
* **egress\_request** (dest, bytes, allow/deny),
* **redaction\_event** (strategy, fields),
* **dp\_consume** (ε, δ, scope),
* **abstain/contain** (κ, reason, suggested next).

These are **SEG edges** with time windows. Compliance packs (Ch.21) export them into audit bundles.

**20.9 KPIs & SLOs for security posture**

* **Injection block rate** ≥ 99% on test corpora; **false-positive** ≤ 2%.
* **SSRF prevented**: 100% of private/IMDS probes blocked in canary.
* **CapTok violations**: ≤ 0.1% per 10k tool calls; MTTR < 15m.
* **Secret echo rate**: 0 in prod; canary only.
* **Abstention precision**: ≥ 0.9 (true-positive abstains / total abstains).
* **DP budget breaches**: 0 per month.

**20.10 Failure modes & runbooks**

| **Failure** | **Likely Cause** | **Runbook** |
| --- | --- | --- |
| Silent policy bypass | Missing policy-prefix or signature not verified | Freeze high-risk tools, rotate templates, audit SEG for unsigned runs |
| Legit request blocked (FP) | Over-eager detector | Shadow-mode compare, adjust thresholds, add test to allowlist corpus |
| Repeated CapTok violations | Tool wrapper drift or compromised agent | Re-issue runtimes, revoke keys, replay traces, HITL review |
| DP budget exhaustion | Mis-scoped aggregation | Raise τ or widen cohort; route to aggregated-only mode |

**20.11 Implementation quick checklist**

* Signed policy prelude in every chain (template hash + signer)
* Quoted boxing of all untrusted content; taint propagated in ACL
* URL normalizer + egress allowlist proxy for all outbound tools
* Capability tokens with short TTL and minimal scopes
* Per-tool sandbox profiles (fs, syscalls, CPU/mem/time)
* PII/secret classifiers on write **and** render paths
* DP budgets configured; SEG logging for ε/δ consumption
* κ-gating wired to UQ, detectors, and cost/latency budgets
* Canary agents and red-team suites in CI/CD
* SEG wiring for all guardrail events; export hooks for audits

**What this chapter unlocks.**  
Agentic power without fear: a system that **remembers** safely (CMC), **orchestrates** within declared limits (APOE), **witnesses** every decision (VIF), and **evolves** without drift (SDF-CVF)—because the substrate treats **evidence as a graph** (SEG) and security as a **first-class capability**, not an afterthought.

**Part VIII — Security, Safety, & Compliance**

**Chapter 21. Compliance Engineering — mapping EU AI Act/SOC2/ISO to VIF/SEG artifacts, retention vs. minimization, and incident response**

Aim. Make compliance **provable-by-construction**. Every control is a **plan step (APOE)**, every decision is a **witness (VIF)**, and every artifact and handshake lands in the **Shared Evidence Graph (SEG)** for export, replay, and audit.

**21.0 Scope & stance (not legal advice)**

This chapter operationalizes common obligations from the **EU AI Act** (risk-based duties), **SOC 2** (Trust Services Criteria), and **ISO** families (27001/27701/42001). We treat them as **design constraints** that the platform satisfies with built-in mechanisms: VIF provenance, SEG lineage, κ-gating/abstention, HITL, DP/redaction, and guardrails-as-code.

**21.1 EU AI Act — control objectives → platform mappings**

We avoid article-by-article replication and instead bind **recurring obligations** to concrete system levers. Use export packs (21.7) to assemble regulator-ready bundles.

| **EU AI Act obligation (concept)** | **What it means in practice** | **AIMOS/APOE mechanism** |
| --- | --- | --- |
| **Risk management & data governance** | Identify use-case risk, validate data quality, track shifts | **SEG**: risk register nodes; **CMC**: data lineage (sources, redactions, DP ε/δ); **HITL** for high-risk plans |
| **Technical documentation** | Keep up-to-date specs, model cards, data sheets, test evidence | **SDF-CVF**: atomic commits across code/docs/tests; **VIF**: model id, weights hash, prompt template hash; **Export packs** |
| **Record-keeping / logging** | Maintain traceable logs of decisions and model/tool usage | **VIF** events for every step; **SEG** time-slicing; signed snapshots; capability-token logs |
| **Transparency** | Notify users of AI use; label synthetic content | **Render layer**: “AI-generated” affordance; **VIF** embeds generator metadata; **policy-prefix** signed |
| **Human oversight** | Enable intervention, overrides, and fail-safes | **ACL**: κ-gating → abstain/contain; **HITL** two-key approvals; **quarantine mode** |
| **Robustness & cybersecurity** | Resist injection/SSRF/exfiltration; test for failures | **Guardrails** (Ch.20), red-team suites; **canary agents**; **CapTok** + sandboxes |
| **Accuracy & performance monitoring** | Measure, calibrate, and watch drift | **UQ/ECE** metrics; **RS/QS** dashboards; **Stream/long-horizon** evals |
| **Post-market monitoring & incident reporting** | Detect/triage/report harmful events | **Incident flows** (21.6); **SEG** incident graph; exportable notifications |

**Retention note.** Where law or sector rules prescribe log retention, use **SEG retention policies** (21.4) to set per-tenant, per-artifact TTLs and legal holds. (When in doubt, keep logs only as long as necessary for the declared purpose and regulatory duty.)

**21.2 SOC 2 — Trust Services Criteria mapping**

| **SOC 2 TSC** | **Control focus** | **Evidence in AIMOS/APOE** |
| --- | --- | --- |
| **Security** | Access control, change mgmt, secure SDLC | **ACL** capability tokens; **SDF-CVF** gated changes; **SBOM/SLSA**; **SEG** access events |
| **Availability** | Uptime, capacity, incident response | **SLOs** + error budgets; **incident runbooks**; **SEG** service health nodes |
| **Processing integrity** | Accurate, timely, authorized processing | **APOE** plans + budgets; **κ abstention** on policy/UQ failures; **VIF** reproducible traces |
| **Confidentiality** | Protect sensitive info end-to-end | **Redaction**, **DP**, encryption at rest/in transit; RAG store separation |
| **Privacy** | Collection/use/disclosure aligned to purpose | **Consent ledger** nodes; **purpose tags** on queries; **data subject flows** (21.5) |

**21.3 ISO mappings (27001/27701/42001)**

* **ISO/IEC 27001** (ISMS): map Annex A controls to guardrails and ops:
  + A.5 organizational controls → **policy-prefix signing**, role catalogs, segregation of duties (HITL).
  + A.8 asset mgmt → **SEG inventories** (models, data sets, connectors).
  + A.12 ops security → **tool sandboxes**, egress allowlists, backups of snapshots.
  + A.14 SDLC → **SDF-CVF** gates, **test coverage** gates, **SBOM**.
* **ISO/IEC 27701** (PIMS): link privacy principles to mechanisms:
  + Lawful basis & purpose limitation → **purpose tags** + **κ** enforcement.
  + Minimization & retention → **retention policies** + **DP**; redaction at ingest.
  + Data subject rights → **subject-access/erasure flows** (21.5).
* **ISO/IEC 42001** (AI management): governance across lifecycle:
  + Risk registers → **SEG risk nodes** with owners and review cadence.
  + Impact assessments (AIA/RAI) → store as **SEG dossiers** linked to systems.
  + Monitoring & improvement → **telemetry → SEG → ADRs**.

**21.4 Retention vs. minimization — policies as code**

**Goals.** Satisfy “keep only what you must; prove what you kept; delete when you can.” We encode this in **Retention Policies (RP)** bound to SEG types.

**Policy model.**

policy\_id: rp-default-highrisk-v3

scope:

seg\_types: [ "vif:event", "tool:egress", "decision", "claim", "snapshot" ]

defaults:

ttl\_days: 190 # configurable by tenant/jurisdiction

storage\_class: WORM # immutability for audit logs

exceptions:

- condition: { legal\_hold: true } # incident or litigation

override: { ttl\_days: null } # no expiry until cleared

- condition: { contains\_pii: true, purpose: "analytics" }

override: { ttl\_days: 30, dp: { epsilon: 1.0, delta: 1e-6 } }

erasure:

strategy: "cryptographic tombstone" # delete keys, preserve non-identifying lineage

propagate: "graph" # cascade to derived artifacts unless purpose-limited

**Lifecycle hooks.**

* **CMC ingest** tags PII, purpose, consent, and jurisdiction.
* **HHNI** stores **redacted embeddings**; pointers to vaulted originals under strict scopes.
* **Retention executor** runs daily; emits **SEG “erasure”** or **“hold-placed/released”** events.

**21.5 Privacy UX & data subject operations**

**Consent & purpose.**

* Every plan step carries a **purpose tag** ("support", "analytics", "training", etc.). κ-gating refuses incompatible reuse.
* Consent ledger in **SEG**: issuance, scope, expiry, withdrawal.

**Data Subject Access Request (DSAR) & Erasure (DSE).**

1. **Locate**: Query SEG across time-slices for subject identifiers and their pseudonymous joins.
2. **Compile**: Export **subject bundle**: sources, uses, recipients, retention, decisions affecting the subject.
3. **Erase**: Apply cryptographic deletion to vaulted PII + cascade; retain minimal non-identifying audit lineage.
4. **Prove**: Produce **DSAR/DSE VIF**: queries, objects affected, timestamps, operator identity.

**21.6 Incident response & notifications**

Two classes: **Security incidents** (confidentiality/integrity/availability) and **Model incidents** (harmful/biased/unsafe outputs or decision flows).

**Detection.**

* **Canary agents** exercise tools periodically;
* anomaly detectors on **κ spikes**, **UQ drifts**, **egress anomalies**;
* “secret echo” traps in RAG corpora.

**Triage.**

* Classify severity (S1–S4), scope (tenants, data classes), and regulatory touchpoints (privacy breach vs. robustness issue).

**Containment.**

* Freeze affected connectors/models (feature flag); force **quarantine mode** (read-only, no tool writes).
* Place **legal hold** on relevant SEG nodes; snapshot state.

**Eradication & recovery.**

* Patch/rollback via **SDF-CVF**; retest with red-team suites; gradually unquarantine.

**Notification.**

* Generate **regulatory pack** (below) with incident timeline, affected data categories, mitigation, and contact.
* Respect jurisdictional windows (e.g., privacy laws often require notification without undue delay; coordinate with counsel).

**21.7 Export packs (one-click audit bundles)**

**Structure.**

/export\_pack\_YYYYMMDDTHHMMZ/

manifest.json # hash tree, versions, scope

policy\_prefix.txt # signed templates in force

model\_card.json # model/weights/prompt template hashes

data\_sheets/ # sources, redaction/DP config

vif\_traces.ndjson # step-by-step plan witnesses

seg\_graph.jsonld # time-sliced evidence graph

risk\_register.json # risks, owners, reviews

sbom/ # supply chain attestations (SLSA, sigstore)

dp\_ledger.csv # epsilon/delta consumption

consent\_ledger.csv # consents by subject/purpose

hitl\_approvals.csv # two-key records for high-risk actions

test\_results/ # red-team suites, accuracy/calibration

incident\_report/ # if applicable

signatures/ # package signing, key metadata

**Generation.**  
apoe export --scope high\_risk --timeslice "2025-01-01..2025-03-31" --tenant acme --signed

**21.8 Control verification — tests & monitoring**

* **Pre-deployment**: policy-pack lint; segregation tests; DP budget simulators; calibration (ECE) checks; adversarial corpora for jailbreak/SSRF/secret-echo.
* **In prod**: SLIs for lineage completeness, κ-abstention precision, RS-lift stability, DP budget headroom, CapTok violation rate, DSAR/DSE latency.

**Pass/Fail samples.**

* **Lineage completeness** ≥ 99% for high-risk flows (no orphaned outputs).
* **ECE** ≤ target band (e.g., 0.03) on evaluation sets.
* **DSAR/DSE** service levels (e.g., compile ≤ X days, erase ≤ Y days) tracked as SLOs.

**21.9 Checklists (implementation quick start)**

**Platform wiring**

* Policy-prefix signing enforced; template hashes in VIF
* All tool calls require valid **CapTok**; sandbox profiles applied
* VIF/SEG enabled on every APOE boundary; time-sliced graph API live
* Redaction at ingest; embeddings on redacted text; DP ledger active
* Purpose/consent tags mandatory; κ refuses incompatible reuse
* Retention policy engine with legal hold + erasure propagation
* DSAR/DSE flow exercised end-to-end; witness emitted
* Export pack CLI enabled; signatures verified in CI
* Incident runbooks rehearsed (tabletop & live canaries)

**Org/process**

* Risk register owners assigned; review cadence in SEG
* Role catalogs & two-key approvers defined
* License/SBOM policies set; SLSA/Sigstore in pipeline
* Training & comms: transparency and user notices templated

**21.10 What this unlocks**

* **Pre-baked proof.** Auditors don’t just get PDFs—they get **replayable traces** and a **time-sliced evidence graph**.
* **Privacy by architecture.** Redaction, purpose tags, DP budgets, and retention policies are not documents; they’re **executable controls**.
* **Resilience with accountability.** When the system abstains, contains, or quarantines, it can **show why**, who approved exceptions, and how it recovered.

**You said:**

proceed Next (Part IX, Ch.22): Retrieval & Reasoning Benchmarks — p@k, nDCG, RS-lift, κ/ECE dashboards, long-horizon agent evaluation.

**ChatGPT said:**

Thought for 10s

**Part IX — Evaluation, Benchmarks, & Observability**

**Chapter 22. Retrieval & Reasoning Benchmarks — p@k, nDCG, RS-lift, κ/ECE dashboards, long-horizon agent evaluation**

Aim. Establish **provable, repeatable** measurements for (a) retrieval quality, (b) reasoning calibration and abstention, and (c) **long-horizon agent** success under budgets—grounded in VIF/SEG so every score has a witness and a replay.

**22.0 Scope & outcomes**

**What this chapter delivers**

* Metric definitions and **exact formulas** (p@k, recall@k, MRR, MAP, nDCG, ECE).
* **RS-lift** protocol that ties HHNI/DVNS improvements to outcome deltas per unit **token/latency cost**.
* A **κ (kappa) & calibration** dashboard spec (coverage vs. risk, reliability diagrams, selective prediction).
* A **long-horizon agent** eval harness: success, budget adherence, repair rate, determinism, and safety outcomes.
* Gold-set curation, adversarial suites, contamination checks, and **promotion gates** for shipping.

**Principles**

1. **Relevance before eloquence**: judge retrieval first; judge wording second.
2. **Calibrate or abstain**: it’s better to say “I don’t know (κ)” than to be confidently wrong.
3. **Show your work**: every metric point links to **VIF/SEG** entries for replay.

**22.1 Retrieval metrics (definitions & usage)**

Let each query have gold relevant set , system returns **ranked list** with per-item RS/score.

**Precision@k (p@k)**

**Recall@k (r@k)**

**MRR (Mean Reciprocal Rank)**  
Let be rank of first relevant item (∞ if none).

**Average Precision (AP) & MAP**

**nDCG@k (normalized Discounted Cumulative Gain)**  
With graded relevance ,

**When to use what**

* **p@k** for answerable Q&A where 1–3 chunks suffice.
* **MRR** when “first hit fast” matters (tool routing).
* **MAP/nDCG** for multi-facet or graded relevance (legal, biomedical).
* Always segment by **query class** (one-hop, multi-hop, long-context, adversarial).

**Traceability**  
Each run emits a **Retrieval VIF**: model id, index snapshot, RS parameters, and per-query ranked list with **claim→evidence edges** in SEG.

**22.2 RS-lift: tying HHNI/DVNS to outcome & cost**

Define **baseline** and **candidate** (e.g., HHNI weights, DVNS forces).

**Lift metrics**

* **Δp@k = p@k(C) − p@k(B)**; same for nDCG/MAP.
* **Cost-normalized lift**:

Choose from SLOs (ms ↔ token budget).

**Significance**

* Use **paired bootstrap** over queries; report 95% CI.
* Guardrail: do **not** promote if CI crosses 0 or if **cost headroom** is exceeded.

**Protocol**

1. Freeze **index snapshot** & model versions (VIF).
2. Run B and C with identical queries, seeds, budgets.
3. Emit **SEG diff** (edges added/removed, RS distributions).
4. Promote only if **Lift\_eff ≥ θ** and safety metrics unchanged or better.

**22.3 κ & calibration: abstention and reliability dashboards**

**Selective prediction (κ)**

* The system abstains when or .
* **Coverage** .
* **Risk-coverage curve**: plot error vs. coverage as thresholds vary; choose operating point that meets SLOs.

**Expected Calibration Error (ECE)**  
Bucket predicted confidences into ; with accuracy and mean confidence :

Report **ECE@domain** and **MaxCE**.

**Reliability diagram**  
10–20 bins; overlay **perfect calibration line**; show **pre/post temperature scaling**.

**Selective F1 / AUROC (optional)**  
For binary answerability gates; track per domain/task.

**Dashboard spec (tiles)**

* **Abstention tile**: coverage, abstention precision/recall.
* **Calibration tile**: ECE, diagram, temp parameter.
* **UQ provenance tile**: which method (ensemble, temp sweep), seed, κ thresholds (all in VIF).
* **Budget tile**: tokens/latency vs. target at chosen κ.

**22.4 Long-horizon agent evaluation (streams & plans)**

Agents execute **APOE plans** (DAGs) over time with tool calls, memory writes, and HITL gates.

**Core outcomes**

* **Task Success (TS%)**: pass/fail against verifiable acceptance tests (SEG decision nodes).
* **Budget Adherence (BA)**: over tokens, time, tool calls.
* **Repair Rate (RR)**: fraction of runs requiring auto-fix or quarantine.
* **Determinism (Det@seed)**: Jaccard similarity of step sequences under fixed seed/context.
* **HITL Utilization**: approvals requested/granted; median time-to-merge.
* **Safety Outcomes**: injection/exfiltration incidents caught (pre-execution) vs. blocked (at execution) vs. leaked.

**Trace-level metrics**

* **κ\_chain**: fraction of boundary gates abstained; false-abstain vs. true-abstain.
* **Step F1**: compare planned vs. executed steps (name+contract).
* **Lineage Fidelity**: % artifacts with complete VIF; **Replay Rate**: % traces that re-execute to same verdict.

**Scenarios**

* **Streamed tasks**: evolving requirements, rolling memory (CMC snapshots).
* **Multi-hop research/code**: retrieval→reason→verify loops.
* **Tool-rich runs**: browser, code exec, DB, vector search; injection probes embedded.

**Harness requirements**

* Freeze **policy-prefix**, models, indices (VIF).
* Record **every boundary** (inputs, outputs, UQ, κ).
* Emit SEG graph; produce **agent scorecard** per run + aggregates.

**22.5 Gold sets & adversarial suites**

**Gold sets (by domain)**

* **Atomic Q&A**: single-hop fact with citations.
* **Multi-hop**: requires 2–3 evidence items with chain-of-thought **not** evaluated, only **evidence linkage**.
* **Procedural**: “do X with tool Y” judged by outcome log.
* **Coding**: compile/test verdicts, coverage thresholds.
* **Long-context**: key facts in middle of ~50–200k tokens (LIM stress).

**Adversarial suites**

* **Injection**: prompt smuggling, casing/encoding, indirect attack through retrieved docs.
* **Exfiltration**: secret-echo canaries, connector abuse.
* **Spec drift**: conflicting instructions across time-slices.
* **Ambiguity**: lexical traps, polysemy (“hot dog” vs “dog”).
* **Reasoning traps**: numerical edge cases, unit conversions, date arithmetic.

**Contamination checks**

* Hash gold materials; check **train overlap**; if uncertain, gate with **counterfactual variants**.

**22.6 Benchmark harness & telemetry**

**Dataset schema**

{

"query\_id": "uuid",

"query": "text",

"class": "one\_hop|multi\_hop|long\_ctx|adversarial|coding|procedural",

"gold": [{"doc\_id":"...","grade":0|1|2}],

"acceptance\_test": {"type":"regex|program|unit\_tests", "spec":"..."},

"notes": {"jurisdiction":"EU", "pii":false}

}

**Runner**

* Consumes **ACL plan** templates; enforces **budgets**.
* Logs **VIF** per step; writes **SEG** lineage; emits **OpenTelemetry** spans with attributes:
  + apoe.step, apoe.role, tokens.in/out, latency.ms, kappa, uconf, rs, gate.result.

**Statistics**

* **Paired bootstrap** for deltas; **Benjamini–Hochberg** for multiple comps.
* **Sequential tests** allowed only with pre-declared α-spending.

**22.7 Example scorecards (what “good” looks like)**

**Retrieval scorecard (weekly)**

* nDCG@10: **+0.041** (CI [+0.026,+0.055]) vs. baseline
* p@3: 0.76 → **0.81**; **tokens/query**: +3.2%; **latency**: −6.5%
* Fail buckets reduced: “polysemy” −28%, “LIM-middle” −19%

**κ & calibration**

* Coverage at κ\*: **0.83**; Error@κ\*: **0.072**
* ECE: **0.028** (from 0.061); Temperature: 0.74
* Selective risk curve inside SLO band (95% CI)

**Agent scorecard**

* TS%: **78.4** (↑5.3); BA: 0.96; RR: 0.07 (↓)
* Determinism@seed: 0.82 Jaccard; Replay: 0.97
* Safety: 0 leaks, 14 pre-execution blocks, 2 at-execution blocks

Each KPI tile links to **SEG slice** + **VIF packet** for audit.

**22.8 Promotion gates (ship/no-ship criteria)**

A candidate retrieval/orchestration stack **may ship** only if:

* **Lift\_eff ≥ θ** (org-set, e.g., +0.02 nDCG per 1k tokens) with CI > 0.
* **ECE ≤ band** (e.g., ≤ 0.03) **and** selective risk curve within SLO.
* **TS% non-degrading** on long-horizon suites; **BA ≥ 0.95**; **RR ≤ 0.1**.
* **Safety equal/better**: no new leak class; injection block rate ≥ baseline.
* VIF/SEG completeness ≥ 99% for all benchmark traces.

**22.9 Reproducibility & export**

Every run pins:

* **Model id**, **weights hash**, **prompt template hash**, **index snapshot id**, **policy-prefix version**, **tools & CapTok profiles**, **seeds**, **κ/UQ settings**.
* Stored as **VIF**; graph written to **SEG** with time-slice.
* **Export pack** (Part VIII) includes: metrics JSON, per-query traces, scorecards, and signatures.

**22.10 Quick checklists**

**Before running**

* Freeze snapshots; warm caches; declare budgets & κ/UQ knobs.
* Load gold/adversarial sets; confirm no contamination.
* Enable full VIF + OpenTelemetry.

**After running**

* Compute deltas + CI; inspect outlier queries.
* Review κ operating point vs. SLOs.
* Attach SEG slice + VIF packet; archive export pack.

**Part IX — Evaluation, Benchmarks, & Observability**

**Chapter 23. Observability & Costing — OpenTelemetry for agents, token/latency budgets, drift & prompt-rot monitors, SLOs and error budgets**

Aim. Give AIMOS/APOE **production-grade sight**: every plan, step, retrieval, tool call, HITL decision, and cost is **traced, measured, and governed**. Observability is not a dashboard—it’s the **nervous system** that enforces budgets, detects drift, and protects SLOs.

**23.0 Scope & outcomes**

**This chapter delivers**

* A complete **OpenTelemetry (OTel)** spec for agents, memory, retrieval, and tools (spans, metrics, logs, exemplars).
* **Token/latency/cost budgets** with admission control, backoff, and graceful degradation.
* **Drift & prompt-rot monitors** for data, behavior, and templates (with alerting).
* **SLOs & error budgets** (definitions, burn-rate alerts, runbooks).

**Principles**

1. **Everything is a step** → everything is observable.
2. **Budgets are contracts** → enforced at compile-time (APOE) and run-time (gates).
3. **Detection without lineage is noise** → all alerts attach a SEG slice & VIF packet.

**23.1 OpenTelemetry for agents (traces, metrics, logs)**

**23.1.1 Trace model (span taxonomy)**

**Root**

* apoe.plan — one per compiled plan execution (DAG).  
  Attributes: plan.id, plan.version, snapshot.id, policy.prefix.version, kappa.operating\_point, budget.tokens.max, budget.latency.ms.max.

**Children (typical)**

* apoe.step — per step (role-typed).
  + Attr: step.id, step.role ∈ {planner|retriever|reasoner|verifier|builder|critic|operator|witness},  
    budget.tokens.max, budget.latency.ms.max, inputs.hash, prompt.template.hash, model.id, weights.hash.
  + Events: gate.enter, gate.exit, abstain.kappa, redaction.applied, quarantine.start, rollback.
  + Status: OK|ERROR|ABSTAIN.
* cmc.write / cmc.read — memory atomization/index I/O.
  + Attr: atoms.count, ids.depth, dd.delta, snapshot.id.
* rag.query — HHNI/DVNS retrieval.
  + Attr: k, rs.mean, rs.p95, ids.mean, dd.mean, index.snapshot.id.
* tool.exec — sandboxed external call (browser, code runner, DB).
  + Attr: tool.name, capTok.id, arguments.hash, egress.bytes, sandbox.policy.id.
* hitl.review — human approval steps.
  + Attr: risk.score, two\_key.required, approver.role, decision.
* seg.write — evidence graph updates.
  + Attr: nodes.count, edges.count, contradictions.count.

**Correlation**

* Use trace\_id across spans; attach **VIF ids** (model/prompt/index hashes) to every boundary span.
* Include seg.slice.id on any span that asserts/derives a claim.

**23.1.2 Metrics (RED/USE + agent-specific)**

**Counters**

* tokens.in.total, tokens.out.total (by model.id, role, tenant).
* tool.exec.count{tool.name, outcome}.
* abstain.count{kappa.reason} (low RS, high UQ, policy).

**Histograms**

* latency.ms{role|tool|retrieval} (p50/p90/p95/p99).
* rs.score (retrieval score distribution).
* uq.confidence (post-calibration).

**Gauges**

* budget.tokens.remaining (plan & step).
* index.size.documents, mem.snapshot.bytes.
* cost.usd.current\_hour (derived; see §23.2).

**Exemplars**

* Attach trace\_id exemplars to latency and cost histograms so outliers hyperlink to traces.

**23.1.3 Logs (structured)**

* **Boundary logs** (JSON): inputs/outputs digests, κ decisions, redactions applied, gate verdicts.
* **Audit logs**: HITL actions, policy overrides, export pack creation.
* **Security logs**: injection triggers, exfiltration blocks (include payload hashes, not raw content).

**Retention**: per compliance policy (see Part VIII): e.g., 6–12 months with **privacy-first redaction** at ingestion.

**23.2 Token/latency budgets & costing**

**23.2.1 Budget objects (declared in ACL & enforced)**

**Per step**

budget:

tokens:

max: 12000

soft: 10000

backoff: summarize|truncate|route\_smaller\_model

latency\_ms:

max: 2500

soft: 1800

tools:

max\_calls: 3

**Per plan**  
Sum of step budgets with headroom factor (default 1.1).

Violation raises budget.exceeded event → see §23.2.3.

**23.2.2 Cost model**

Let p\_in, p\_out be USD per 1K tokens for the model.id.  
Per step cost:

Aggregate to cost.plan.usd. Emit to gauge cost.usd.current\_hour and counter cost.usd.total.

**Capacity planning (monthly)**

Segment by tenant/domain; alert on 20% deviation.

**23.2.3 Admission control & graceful degradation**

**At compile time (APOE)**

* Static checks ensure budgets exist for every step; reject plan if missing.

**At run time**

* If soft exceeded:
  1. **Summarize** context (dumbbell compression) and retry once.
  2. **Route** to a cheaper model with higher κ (more abstention).
  3. **Truncate** non-critical attachments (policy-aware).
* If max exceeded → abstain.kappa(budget) and escalate to hitl.review when configured.

**Pseudocode**

if tokens.used > soft:

context' = compress(context, policy="dumbbell")

rerun(step, context')

if tokens.used' > max:

emit(abstain.kappa, reason="budget")

route(HITL | cheaper\_model)

**23.3 Drift & prompt-rot monitors**

**Why**: HHNI/DVNS, models, and prompts evolve. We detect **data drift**, **behavior drift**, and **template rot** early—with lineage.

**23.3.1 Data drift (retrieval/memory)**

* **Embedding drift**: monitor centroid shift per corpus; alarm when > θ.
* **Distribution shift**: KL-divergence of RS histograms between windows.
* **Index health**: IDS (indexing depth score) mean/variance; drop indicates ingestion or tagging failure.
* **LIM sentinel**: p@k for mid-placement canaries (lost-in-the-middle probes).

**Alert**: drift.data with SEG slice listing affected indices/snapshots.

**23.3.2 Behavior drift (model & agent)**

* **Canary suite deltas**: weekly run of fixed gold sets; compare MAP/nDCG and κ coverage at same thresholds.
* **UQ calibration drift**: ECE delta > θ; triggers temperature re-fit.
* **Action profile shift**: step distribution (planner vs. retriever vs. verifier) changes (χ² test) → may indicate routing regression.

**Alert**: drift.behavior with VIF (model hash change?) and trace exemplars.

**23.3.3 Prompt-rot monitors (templates & policies)**

* **Template hash** mismatch (prompt.template.hash) vs. approved registry → block in staging, warn in prod.
* **Policy-prefix diff**: semantic distance > θ (embed-compare) → requires two-key approval.
* **Guardrail efficacy**: injection block rate ↓ while injection attempts ↑ → investigate.

**Alert**: rot.prompt → auto-create ADR stub & require HITL review.

**23.4 SLOs & error budgets**

**23.4.1 SLIs (service level indicators)**

**Availability**

* plan.success.rate = OK plans / total (exclude κ-abstentions with reason=policy if within spec).

**Quality**

* answer.accept.rate = plans passing acceptance tests (from Ch.22 gold sets).
* retrieval.ndcg.p95 by domain.

**Performance**

* latency.ms.p95 per plan class; tool.exec.p95.

**Cost**

* tokens.per.plan.median, cost.usd.per.plan.p95.

**Safety**

* injection.block.rate, exfiltration.prevented.rate.

**23.4.2 SLOs & budgets (examples)**

* Availability: **≥ 99.5%** monthly.
* Quality: **answer.accept.rate ≥ 80%** on tier-A suites.
* Performance: **p95 latency ≤ 2.5s** (query class A).
* Cost: **median tokens/plan ≤ 9k**.
* Safety: **injection.block.rate ≥ 99.9%**.

**Error budget** for Availability:

e.g., 0.5% of 30 days = 3h 36m downtime.

**23.4.3 Burn-rate alerting (multi-window)**

Use **multi-window multi-burn (MWMB)**:

* **Fast window**: 1 hour; **Slow**: 6 or 24 hours.
* Alert if

Tie alerts to runbooks with pre-approved mitigations.

**23.4.4 Runbooks (automations)**

* **Latency breach** → enable aggressive summarization, reduce k in RAG, switch model tier.
* **Cost breach** → raise κ (more abstentions), gate tool calls, enforce stricter soft budgets.
* **Quality breach** → pin model to last-known-good, roll back index snapshot, route to verifier-heavy plan.
* **Safety breach** → escalate to containment profile: tool off, only retrieval + summarize, HITL mandatory.

Each action emits a governance.change event with SEG export.

**23.5 Dashboards & exemplars (what to ship)**

**Top-level (Ops)**

* SLO tiles with burn-rate sparkline, error-budget remaining.
* p95 latency, cost p95, success rate, κ coverage.

**Agent (Plan)**

* Steps waterfall (trace), tokens in/out stacked by role, κ boundaries hit.
* RS distribution vs. threshold, IDS trend, DD (dependency delta) histogram.

**Retrieval**

* nDCG@10 over time, LIM sentinels, RS-lift vs. baseline (from Ch.22).

**Safety**

* Injection attempts vs. blocks, exfiltration prevented, redaction counts.

Each tile supports **click-through to exemplars** (OTel trace + SEG slice + VIF packet).

**23.6 Alerts, tickets, and exports**

* Alerts open tickets with **attached export packs**: metrics snapshot, top offending traces, SEG lineage, and the exact VIF (model/prompt/index hashes).
* Exports are **privacy-scrubbed** at source (PII redaction), signed, and stored with retention policy.

**23.7 Quick checklists**

**Instrumentation**

* All APOE steps emit apoe.step spans with budgets & VIF hashes.
* Retrieval emits RS/IDS/DD; memory emits snapshot ids.
* Tools carry capTok.id and sandbox policy labels.

**Budgets & Cost**

* Budgets declared in ACL for every step/plan.
* Admission control + graceful degradation configured.
* Cost forecasting vs. actual alerts enabled.

**Drift & Rot**

* Embedding & RS distribution monitors on; LIM sentinels active.
* Weekly canary suite; ECE drift alarms.
* Template/policy hashes pinned; diffs require two-key.

**SLOs**

* SLIs defined & computed; SLOs published per plan class.
* MWMB alerting wired to runbooks; governance events logged.

**23.8 Minimal reference (names & attributes)**

**Span names**: apoe.plan, apoe.step, cmc.write, cmc.read, rag.query, tool.exec, hitl.review, seg.write, governance.change.  
**Key attrs**: plan.id, step.role, budget.tokens.max, tokens.in/out, latency.ms, rs, uq, kappa, model.id, weights.hash, prompt.template.hash, index.snapshot.id, capTok.id.  
**Metrics**: tokens.in.total, tokens.out.total, latency.ms, rs.score, uq.confidence, cost.usd.\*, abstain.count, tool.exec.count.  
**Logs**: gate.verdict, abstain.kappa, redaction, quarantine, rollback, hitl.decision.

**Part X — Case Studies & Patterns**

**Chapter 24. Machine Communication — protocol translation, negotiation, device/file diplomacy (with traces, budgets, SEG lineage)**

Aim. Show AIMOS/APOE acting as a **machine-communication system**—bridging humans, documents, tools, and devices—with **memory-native retrieval (CMC/HHNI), compiled orchestration (APOE/ACL), verifiable lineage (VIF/SEG),** and safe evolution (SDF-CVF). Each case provides: scenario, APOE plan, budgets, OpenTelemetry trace, VIF packet, SEG slice, failure modes, and playbooks.

**24.0 Patterns at a glance**

* **Protocol translation:** “speak both ways” across formats and APIs; preserve semantics and provenance.
* **Negotiation & mediation:** structured bargaining under constraints; κ-aware abstention; HITL where risk>θ.
* **Device & file diplomacy:** hostile inputs, partial parses, capability tokens, and redaction by default.

All three reuse the same spine:

1. **Retrieve** the right memories/specs (RS↑) →
2. **Compile** a typed plan with budgets →
3. **Execute** with guarded tools →
4. **Witness** the lineage (VIF) →
5. **Write** SEG evidence →
6. **Evolve** atomically (SDF-CVF gates).

**24.1 Case Study A — Protocol Translation (CSV ledger → JSON API → signed PDF)**

**Scenario.** A finance team uploads a monthly CSV ledger and needs a JSON batch for a RegTech API plus a signed PDF summary. Constraints: PII masking, schema drift handling, and reproducible outputs.

**Inputs.**

* User artifact: ledger\_2025\_09.csv (unknown dialect).
* Target: REST /v2/transactions (OpenAPI v3).
* Policy: mask acct\_number except last 4; round totals to 2dp.

**Success criteria.**

* Schema fidelity ≥ 99.5% rows accepted by API sandbox.
* PII masked; SEG lineage resolvable from PDF to raw CSV.
* Cost ≤ $0.02 per 1k rows; p95 latency ≤ 2.5s for 50k rows.

**APOE plan (ACL sketch)**

pipeline translate\_ledger\_v2

input { file: csv\_blob, openapi: spec\_ref }

budget { tokens.max: 12000, latency\_ms.max: 2500, tools.max\_calls: 6 }

step detect\_dialect(role: retriever)

tool: csv\_sniffer

out: dialect\_report

budget { latency\_ms.max: 150 }

step map\_schema(role: reasoner)

in: dialect\_report, spec\_ref

out: field\_map, pii\_mask\_rules

gate: { policy: pii\_masking, schema: openapi\_conformance }

step transform(role: builder)

tool: csv\_to\_json

in: field\_map, pii\_mask\_rules

out: json\_batch

budget { tools.max\_calls: 2 }

step validate(role: verifier)

tool: openapi\_validator

in: json\_batch, spec\_ref

out: acceptance\_metrics

gate: { tests: acceptance >= 0.995 }

step render\_pdf(role: builder)

tool: pdf\_renderer

in: json\_batch, acceptance\_metrics

out: summary\_pdf

step witness(role: witness)

out: vif\_packet, seg\_edges

**Budgets (plan).** tokens.max=12k, latency\_ms.max=2500, tools.max\_calls=6. κ set to abstain if acceptance < 0.995 or UQ ECE>0.03.

**Trace (OTel excerpt)**

* apoe.plan translate\_ledger\_v2 [lat=1780ms] attrs: snapshot.id=S-83A, policy.prefix=v5, budget.tokens.max=12000.
  + rag.query HHNI k=12 RS.p50=0.82 (found prior mappings, PII policy)
  + tool.exec csv\_sniffer OK dialect=excel-semicolon
  + apoe.step map\_schema tokens.in=1.9k UQ.cal=0.014
  + tool.exec csv\_to\_json rows=50k egress=9.2MB
  + tool.exec openapi\_validator acceptance=0.9987
  + tool.exec pdf\_renderer pages=3
  + seg.write nodes=7 edges=9 (claim→source links)

**VIF (provenance packet, summary)**

model.id: apoe-compiler-02

weights.hash: sha256:edc1…

prompt.template.hash: sha256:aa74…

tools: [csv\_sniffer@1.3, csv\_to\_json@2.1, openapi\_validator@0.9, pdf\_renderer@1.0]

inputs: { csv.sha256: 7f21…, openapi.sha256: 9b10… }

snapshot.id: S-83A

uq: { ece: 0.014, ensemble.var: 0.007 }

**SEG lineage (JSON-LD slice)**

{

"@graph": [

{ "@id": "claim:pdf:2025-09", "type": "Claim",

"derivesFrom": ["artifact:json\_batch:7c1e", "metric:acceptance:0.9987"],

"createdAt": "2025-10-17T12:03:11Z" },

{ "@id": "artifact:json\_batch:7c1e", "type": "Artifact",

"supports": ["source:csv:7f21"], "tool": "csv\_to\_json@2.1" },

{ "@id": "source:csv:7f21", "type": "Source",

"hash": "sha256:7f21…", "piiMasking": "acct\_number: last4" }

]

}

**Failure modes & playbooks.**

* *Schema drift:* validator rejects >0.5% → κ abstain + planner regenerates field\_map from nearest prior mapping; HITL diff shown.
* *PII leakage:* gate fails → auto-redact + re-validate; repeated failure quarantines branch.
* *Cost spike:* increase batch size, reduce k in RAG, or route cheaper model for map\_schema.

**KPIs.** acceptance≥0.995, RS-lift vs. baseline KNN ≥ +0.06, p95 latency≤2.5s, cost p95 plan≤$0.98.

**24.2 Case Study B — Negotiation & Mediation (SLA slotting across constrained APIs)**

**Scenario.** A logistics broker negotiates pickup windows among three carriers’ APIs with conflicting constraints (weight limits, blackout dates, surge pricing). AIMOS/APOE mediates a Pareto-feasible schedule.

**Inputs.**

* 3 OpenAPI specs + policy clauses (penalties, PII rules).
* Shipment bundle (weights, origins, deadlines).

**Success criteria.** Feasible plan with **dominated-option pruning**, cost ≤ target, and **auditable counteroffer tree**.

**APOE plan (bidirectional protocol + bargaining)**

pipeline negotiate\_windows\_v1

input { carriers: [openapi: A,B,C], bundle: shipments }

budget { tokens.max: 18k, latency\_ms.max: 4500 }

step normalize\_specs(role: retriever) -> norm\_specs

step generate\_offers(role: planner)

in: norm\_specs, shipments

out: initial\_offers

gate: { coverage: >= 0.95 shipments }

step solicit\_quotes(role: operator)

tool: http\_client[signed]

in: initial\_offers -> carrier\_responses

budget { tools.max\_calls: 9 }

step reconcile(role: reasoner)

in: carrier\_responses, penalties

out: pareto\_front, counteroffers

gate: { dominance\_pruned: true }

step negotiate(role: operator)

tool: http\_client

in: counteroffers -> acceptances

policy: cap\_tokens, pii\_min

gate: { risk.score < 0.6 || two\_key\_approved }

step witness(role: witness) -> vif, seg

**Budgets.** tokens.max=18k, latency\_ms.max=4500, tools.max\_calls=9. κ abstain on risk.score≥0.8.

**Trace highlights.**

* rag.query retrieved prior **carrier quirks** (RS.p50=0.77).
* operator made 7 signed calls (capTok ids recorded).
* reconcile produced Pareto set size=4; **dominance\_pruned=true**.
* negotiate converged after 2 rounds; p95 roundtrip=620ms.

**VIF essentials.**

* Model/weights/prompt hashes for reconcile step included; risk model version noted.
* Tool identities + capTok ids; tariffs snapshot id.

**SEG slice (offer lineage).**

* Nodes: offer:A#102, offer:B#88, counter:C#44, decision:bundle#Q4.
* Edges: supports, dominates, accepts, rejects, with timestamps; **contradiction edges** retained to record declined offers (SEG design).

**Failure modes.**

* *Non-convergence:* κ abstain with proposal set + **HITL two-key** to select compromise.
* *API instability:* operator routes through cached spec; falls back to email template generator with VIF tag fallback.email.

**KPIs.** Dominance pruning ratio≥0.5; cost within target±3%; κ true-positive abstention≥95% (only when needed).

**24.3 Case Study C — Device & File Diplomacy (sensor logs → safe analytics)**

**Scenario.** Field sensors upload mixed archives (.zip with .bin, .csv, unknown .dat). Need: sanitize, parse partially, extract metrics, and publish a dashboard—**without exfiltrating secrets** and with **time-sliced evidence**.

**Inputs.** Device blobs + data dictionary (sometimes stale).  
**Success criteria.** Max salvage; zero secret leakage; reproducible metrics; SEG time slicing so analysts can “rewind.”

**APOE plan (hostile-input first)**

pipeline ingest\_device\_archive

input { archive: blob }

budget { tokens.max: 10k, tools.max\_calls: 8, latency\_ms.max: 3000 }

step fingerprint(role: operator)

tool: mime\_sniffer|yara\_scanner

out: manifest, risk.score

gate: { risk.score < 0.4 || contain\_only }

step unpack(role: operator)

tool: safe\_unzip[airgap]

out: file\_list

step parse\_known(role: builder)

tool: csv\_parser|bin\_decoder

in: file\_list

out: records[], rejects[]

gate: { pii\_redacted: true }

step partial\_parse(role: reasoner)

in: rejects[], prior\_decoders

out: provisional\_decoders, partial\_records

policy: no\_external\_calls

step metrics(role: verifier)

in: records + partial\_records

out: timeseries, qc\_report

gate: { qc.passed }

step witness(role: witness) -> vif, seg

**Budgets.** Constrained tools; *air-gapped* unpack; no external calls during partial\_parse.

**Trace.**

* fingerprint flagged 1 suspicious .dat (YARA rule hit) → **containment** applied; file quarantined, not parsed.
* partial\_parse used **pattern mining** on rejects with **no outbound I/O**; decoded 62% extra rows.
* metrics QC: gap rate 1.3% (<2% threshold).
* seg.write: contradiction edges for quarantined file (recorded, not deleted).

**VIF.**

* Tools with policy tags: safe\_unzip[airgap], bin\_decoder@0.3.
* UQ for inferred decoders: ensemble.var=0.11, ECE=0.028 → confidence band displayed.

**SEG (time-sliced).**

* source:zip:4e9a → artifact:records:day=2025-09-28
* contradicts edge to source:dat:9b17 (quarantine reason: YARA-sig-MQTTkey)
* decision:qc.ok supports claim:timeseries.published.

**Failure modes.**

* *Decoder hallucination risk:* if ECE>0.05, κ abstain from publishing; HITL required to approve provisional decoder with diff view.
* *Secret exfiltration attempt:* tool policy blocks; event logged; pipeline switches to **contain\_only**.

**KPIs.** Salvage ratio≥0.85, QC pass rate≥0.98, zero exfiltration incidents, reproducible rebuild from SEG slice.

**24.4 Reusable assets**

**ACL templates (starters).**

* translate\_protocol\_v2.acl — CSV/JSON/PDF chain with PII gates.
* negotiate\_bids\_v1.acl — multi-round mediator with dominance pruning.
* ingest\_device\_airgap.acl — hostile-input plan with quarantine/contain.

**Policy packs.**

* pii.masking.v5, capTok.carrier\_ops.v3, airgap.unpack.v2, containment.strict.v1.

**Dashboards.**

* Case-specific tiles (acceptance, dominance, salvage) + **exemplar links** (trace + SEG + VIF).

**24.5 What this demonstrates**

* **CMC/HHNI** keeps translation and negotiation grounded in **retrievable** history (mappings, quirks, policies).
* **APOE/ACL** compiles interactions into **deterministic, budgeted plans**, not ad-hoc chats.
* **VIF/SEG** makes every artifact **auditable**—from a signed PDF back to a masked CSV; from a counteroffer back to API quotes; from a timeseries back to quarantined files.
* **SDF-CVF** enforces **parity**: specs, code, tags, and traces change together—or not at all.

**Checklists**

**Before running a new “machine-communication” integration**

* Prior mappings and policies retrieved (RS≥0.75).
* Plan budgets present; κ thresholds defined.
* Tool capability tokens scoped; sandbox policies active.
* VIF fields (model/prompt/index hashes) wired at each boundary.
* SEG export pack enabled for the final artifact.

**Acceptance gates (per case)**

* Protocol translation: validator acceptance≥0.995; PII masked.
* Negotiation: Pareto set produced; risk<0.6 (or two-key).
* Device diplomacy: QC passed; secrets contained; ECE within band.

**Part X — Case Studies & Patterns**

**Chapter 25. Builder Programs — Greenfield • Brownfield • Compliance-Critical**

*(full APOE plans, budgets, and SEG-anchored release decisions)*

Aim. Show how AIMOS/APOE delivers **software that ships** under memory-native retrieval (CMC/HHNI), compiled orchestration (APOE/ACL), verifiable lineage (VIF/SEG), and safe evolution (SDF-CVF). Each program includes: objectives, inputs, success criteria, **ACL plan**, budgets, trace highlights, VIF/SEG artifacts, acceptance gates, failure playbooks, and KPIs.

**25.1 Greenfield App — “From blueprint to running product”**

**Objectives.** Take a structured blueprint → scaffold → code → test → deploy → observe, with **parity** between spec↔code↔evidence and **replayable** builds.

**Inputs.**

* blueprint.yaml (features, non-functionals, data contracts, policy packs).
* Component library constraints; cloud targets; cost/latency SLOs.

**Success criteria.**

* All features in blueprint implementable; p95 build ≤ X min; test coverage ≥ Y%; preview URL + SEG lineage; rollback snapshot available.

**APOE plan (ACL)**

pipeline greenfield\_v1

input { blueprint: yaml, policy\_packs: [pii, license], env: cloud }

budget { tokens.max: 45k, tools.max\_calls: 20, latency\_ms.max: 900000 }

step decompose\_spec(role: planner)

in: blueprint

out: work\_breakdown, domain\_models

gate: { completeness >= 0.95 }

step scaffold(role: builder)

tool: project\_scaffolder

in: work\_breakdown, domain\_models

out: repo, ci\_pipeline

gate: { license\_compliance.ok }

step codegen\_iter(role: builder)

tool: code\_generator

in: repo, component\_catalog

out: pr\_branch

policy: spec\_tags\_required

step tests(role: verifier)

tool: test\_runner

in: pr\_branch

out: coverage\_report, fail\_list

gate: { coverage >= target && fail\_list.empty? }

step deploy\_preview(role: operator)

tool: preview\_deployer

in: pr\_branch

out: preview\_url, telemetry\_sink

step observe(role: operator)

tool: trace\_collector|budget\_monitor

out: kpis, drift\_report

gate: { kpis.within\_slo }

step witness(role: witness)

out: vif\_packet, seg\_edges, release\_candidate

**Budgets.** tokens.max=45k, tools.max\_calls=20, build wall=15 min. κ-abstain if completeness<0.95, coverage<target, or SLO breach.

**Trace highlights (OTel).**

* rag.query(HHNI) seeds prior patterns (RS.p50=0.81).
* project\_scaffolder emitted repo@sha b81c…; CI seeded.
* test\_runner coverage=82.4%≥target; p95 unit time 410ms.
* preview\_deployer URL issued; **budget\_monitor**: token spend 38% of cap.

**VIF (summary).**  
model=apoe-compiler-02, weights.sha=…, prompt.sha=…, tools [scaffolder@1.6, codegen@3.1, test\_runner@2.4, deployer@1.2], snapshot=S-GF-12A, uq={ece:0.018, var:0.009}.

**SEG slice (JSON-LD, release candidate).**

{

"@graph": [

{"@id":"claim:release:rc-0.3","type":"Claim",

"derivesFrom":["artifact:repo:b81c","metric:coverage:0.824","kpi:slo\_ok"]},

{"@id":"artifact:repo:b81c","type":"Artifact","supports":["source:blueprint:sha7e"]},

{"@id":"decision:promote\_rc","type":"Decision","supports":["claim:release:rc-0.3"]}

]

}

**Acceptance gates.**

* **Parity:** blueprint sections tagged and reflected in code modules (SDF-CVF E-1).
* **Tests:** ≥ target, zero critical failures.
* **SLOs:** latency/cost within declared budgets.
* **VIF present:** model/tool/prompt hashes + snapshot.

**Failure playbooks.**

* *Coverage shortfall:* auto-generate unit/property tests; re-run; κ if still <target.
* *Budget overrun:* parameterize codegen (smaller model tier), compress RAG k, or split PR.
* *Spec drift:* quarantine branch; open ADR; require two-key.

**KPIs.** CFR<10%, MTTR<2h, RS-lift vs. baseline scaffold≥+0.05, preview replay fidelity=100%.

**25.2 Brownfield Refactor — “Stabilize, modernize, and keep parity”**

**Objectives.** Refactor legacy service without regressions; achieve **spec↔code** parity, add tests, reduce cost/latency.

**Inputs.**

* Legacy repo; prod logs; informal spec; error budget policy.

**Success criteria.**

* Behavior parity verified by **trace-based tests**; cost ↓ ≥ X%; p95 latency ↓; SEG shows old→new mapping.

**APOE plan (ACL)**

pipeline brownfield\_refactor\_v1

input { repo\_legacy: git, traffic\_samples: traces, target\_stack: template }

budget { tokens.max: 30k, tools.max\_calls: 18, latency\_ms.max: 600000 }

step mine\_contracts(role: retriever)

tool: log\_miner|schema\_infer

out: inferred\_spec, golden\_traces

gate: { coverage.traces >= 0.9 }

step propose\_design(role: planner)

in: inferred\_spec, target\_stack

out: migration\_plan, risk\_matrix

gate: { risk.total < 0.6 }

step refactor(role: builder)

tool: code\_modder|scaffolder

in: repo\_legacy, migration\_plan

out: repo\_modern

step parity\_tests(role: verifier)

tool: replay\_runner

in: repo\_modern, golden\_traces

out: parity\_report

gate: { parity.ok }

step perf\_budget(role: operator)

tool: load\_tester|cost\_profiler

out: perf\_report, cost\_report

gate: { perf.p95 <= target && cost.delta <= -X% }

step witness(role: witness) -> vif, seg, release\_candidate

**Budgets.** tokens.max=30k, κ on coverage.traces<0.9 or parity.ok=false.

**Trace highlights.**

* log\_miner synthesized 1.2k **golden traces** (RS.p50=0.79 to prior incidents).
* replay\_runner parity 99.2% endpoints; 6 diffs quarantined.
* load\_tester p95 ↓ 28%; cost\_profiler infra cost ↓ 22%.

**VIF.**  
Includes hashes for mined traces package; parity method; perf/cost tools versions; snapshot S-BF-09D.

**SEG mapping.**  
Edges derives from legacy endpoints → new handlers; contradiction edges retained for known, accepted behavior changes with ADR references.

**Acceptance gates.**

* Golden trace coverage≥0.9; parity≥0.99 or ADR-backed exceptions.
* Perf/cost gates pass; VIF + snapshot replayable.

**Failure playbooks.**

* *Parity diffs:* auto-patch with constrained code mods; if safety risk, HITL two-key.
* *Perf regression:* revert micro-opt path; re-tune cache/window; adjust DB plan.

**KPIs.** Parity≥0.99, cost-delta≤−X%, p95 delta≤−Y%, CFR<8%, MTTR<1h (canary).

**25.3 Compliance-Critical Workflow — “Ship under regulation by design”**

**Objectives.** Deliver/maintain a workflow in a **high-risk** domain (e.g., medical coding, financial reporting) with **native auditability** (VIF/SEG), **mandatory logging**, labeling, retention, and **HITL**.

**Inputs.**

* Policy pack: EU AI Act mappings; retention rules; labeling requirements.
* Gold sets; redaction schemas; HITL roster.

**Success criteria.**

* Every artifact bears VIF + machine-readable labels; logs retained; SEG exports pass audit; two-key approvals recorded.

**APOE plan (ACL)**

pipeline compliance\_critical\_v1

input { policy\_pack: compliance, goldsets: labeled, data\_stream: source }

budget { tokens.max: 25k, tools.max\_calls: 16, latency\_ms.max: 300000 }

step redact(role: operator)

tool: pii\_redactor

in: data\_stream

out: sanitized\_stream

gate: { leak.rate == 0 }

step infer(role: reasoner)

tool: domain\_model

in: sanitized\_stream

out: draft\_labels, uq\_vector

gate: { uq.calibrated && uq.ece <= 0.03 }

step verify(role: verifier)

tool: gold\_eval|rule\_checker

in: draft\_labels, goldsets

out: eval\_report

gate: { f1 >= policy.min\_f1 && violations == 0 }

step hitl(role: critic)

tool: review\_console

in: eval\_report, high\_risk\_cases

out: approvals, overrides

gate: { two\_key.approved }

step label\_outputs(role: builder)

tool: provenance\_labeler

in: approvals

out: labeled\_artifacts

step witness(role: witness)

out: vif\_packet, seg\_edges, export\_pack

**Budgets.** κ if ECE>0.03 or f1<policy.min; HITL mandatory for high-risk cases.

**Trace highlights.**

* pii\_redactor leak.rate=0; 3 attempted exfil blocks logged.
* gold\_eval f1=0.912≥min; violations=0.
* review\_console two-key approvals=27; response p95 4m.
* provenance\_labeler applied machine-readable synthetic content tags.

**VIF.**  
Model/tool/prompt hashes; UQ vector summary; HITL approver IDs (pseudonymized); snapshot=S-CC-31B.

**SEG & export.**

* Time-sliced chain: source→sanitized→infer→verify→HITL→label→publish.
* Export pack bundles JSON-LD + logs + label manifests + retention schedule.

**Acceptance gates.**

* **Zero PII leakage**, **UQ calibrated**, **f1≥min**, **violations=0**, **two-key approvals present**, **labels attached**, **export pack built**.

**Failure playbooks.**

* *Calibration drift:* run temperature sweep; re-fit calibrator; κ until ECE≤band.
* *Policy violation:* quarantine batch, open incident, attach review notes to SEG, notify stakeholders.
* *HITL backlog:* triage by risk; temporary throttling of intake with user messaging.

**KPIs.** Audit pass rate 100%, ECE≤0.03, time-to-approve p95≤SLA, retention compliance 100%, incident MTTR≤24h.

**25.4 Release decision rubric (SEG-anchored)**

A release can be promoted **only if**:

1. **Gates** for the program type pass (above),
2. **VIF** complete at all plan boundaries,
3. **SEG** shows unbroken lineage from sources to decision, with contradiction edges (not deletions) for changes, and
4. **SDF-CVF parity** holds (code/docs/tags/traces in the same commit).

**Decision record (SEG node).**

* decision:promote with attributes {risk\_score, κ\_count, gates\_passed[], exceptions[ADR#]} pointing to claim:release\_candidate and all supporting artifacts.

**25.5 Checklists**

**Before kickoff (all programs)**

* HHNI prior art retrieved (RS≥0.75).
* ACL plan compiled with budgets + κ thresholds.
* Tool capability tokens scoped; sandboxes configured.
* VIF wiring verified; snapshot IDs reserved.
* SDF-CVF gates enabled (parity, tests, trace).

**Before promote**

* Acceptance gates green.
* SEG export pack generated and archived.
* Rollback snapshot validated by replay.

**Part X — Case Studies & Patterns**

**Chapter 26. Ops & Incidents — Log Forensics, RCA with VIF/SEG, and Safe Autoremediation**

Aim. Turn **telemetry into evidence**, **evidence into decisions**, and **decisions into controlled change**—without violating SDF-CVF parity or compliance guarantees. This chapter defines the operating model, forensic pipeline, RCA methods over the **Shared Evidence Graph (SEG)**, and **autorem­ediation** playbooks with κ-gating, canaries, and two-key approval where risk demands it.

**26.0 Operating Model & Severity Taxonomy**

**Roles.**

* *Incident Commander (IC)* — owns timeline & decisions.
* *Resolver(s)* — service owners, on-call engineers.
* *Witness* — curates VIF packets / SEG links in real time.
* *Scribe* — maintains immutable timeline; exports post-hoc pack.
* *Comms* — stakeholder/customer comms.

**Severities.**

* **SEV-1** customer-visible outage or legal risk → HITL two-key required.
* **SEV-2** partial impact; degraded SLOs → guarded auto-remediation allowed.
* **SEV-3** limited scope; error budget burn detected.
* **SEV-4** near-miss, anomaly only; create watch & experiment.

**Golden clocks.**

* **MTTD** (detect), **MTTI** (IC engaged), **MTTC** (contain), **MTTR** (restore), **MTTP** (publish postmortem). Targets are declared per service.

**26.1 Signal Surface & Ingestion**

**Four signals** (OpenTelemetry aligned): **metrics, logs, traces, events**.  
**Required invariants:** time sync (NTP), **snapshot IDs** embedded in every span/log, PII-safe redaction at ingress.

**Forensic log schema (normalized).**

{

"ts": "2025-10-17T12:34:56.789Z",

"service": "apoe.orchestrator",

"env": "prod",

"level": "ERROR",

"snapshot\_id": "S-BF-09D",

"trace\_id": "9b7…",

"span\_id": "a2c…",

"actor": "step:retriever",

"msg": "prompt\_injection\_detected",

"kv": {

"run\_id": "R-00213",

"gate": "policy.redaction",

"risk\_score": 0.87,

"uq\_ece": 0.052,

"tool": "web.fetch",

"kappa\_abstain": true

}

}

**Ingest→Normalize→Redact→Index pipeline**

1. **Ingest** (syslog/OTLP) → 2) **Normalize** (JSON) → 3) **Redact** (PII, secrets) → 4) **Index** (columnar + vector) → 5) **Link** to **SEG** via snapshot\_id, trace\_id.  
   All stages emit **VIF** tuples (tool versions, hashes) to guarantee replay.

**26.2 Log Forensics: Patterns & Queries**

**Canonical patterns.**

* **Prompt-injection:** sudden spike in policy.violations + new referers.
* **Exfil/SSRF:** unusual egress domains, blocked capability tokens.
* **Cost runaway:** token spend derivative > threshold, RS-lift ↓.
* **Drift/prompt-rot:** κ-abstentions ↑ + ECE ↑ + RS distribution shift.
* **Data leakage near-miss:** redactor blocks > baseline p95.

**Query snippets (illustrative).**

*Time-boxed anomaly burst (SQL-ish):*

SELECT window\_start, COUNT(\*) AS inj\_attempts

FROM logs

WHERE msg='prompt\_injection\_detected'

AND ts BETWEEN now()-INTERVAL '30 minutes' AND now()

GROUP BY TUMBLE(ts, INTERVAL '5 minutes')

HAVING COUNT(\*) > baseline \* 2.5;

*Trace join to recent changes:*

SELECT l.trace\_id, d.pr\_number, d.change\_type

FROM logs l

JOIN deployment\_events d

ON l.snapshot\_id = d.snapshot\_id

WHERE l.level='ERROR' AND d.deployed\_at >= now()-INTERVAL '2h';

**26.3 RCA with the Shared Evidence Graph (SEG)**

**RCA goal.** Produce a **time-sliced subgraph** linking symptoms → spans/logs → changes → decisions → sources, then attach a **Decision** node with remediations.

**Minimal nodes.** Symptom, Span, Log, Metric, Change, Spec, Claim, Decision, Gate, ToolCall, Snapshot.

**Cypher (example) — “What changed before the first error?”**

MATCH (e:Log {level:'ERROR'})-[:IN\_TRACE]->(t:Trace)

WHERE e.ts >= $t0 AND e.ts <= $t1

MATCH (t)-[:HAS\_SNAPSHOT]->(s:Snapshot)<-[:DEPLOYED]-(c:Change)

WHERE c.at <= e.ts AND c.at >= e.ts - duration('PT2H')

RETURN e, t, s, c ORDER BY e.ts ASC LIMIT 50;

**SPARQL (lineage to data source).**

SELECT ?claim ?artifact ?source WHERE {

?claim a seg:Claim ;

seg:derivesFrom ?artifact .

?artifact seg:supports ?source .

FILTER (?claim = seg:Claim\_release\_rc\_0\_3)

}

**Eight-step RCA loop.**

1. **Freeze** time window; snapshot SEG slice.
2. **Cluster** symptoms (logs/metrics/traces).
3. **Correlate** to changes/spec diffs.
4. **Validate** with RS-lift & UQ deltas.
5. **Hypothesize** (explicit Claim).
6. **Experiment** in canary; capture VIF.
7. **Decide** (promote/revert/patch).
8. **Record** Decision & link all artifacts (no deletions—use contradicts edges).

**26.4 Incident Pipelines in ACL (APOE)**

**Detection→Triage→Containment→Recovery plan.**

pipeline incident\_sev2\_cost\_spike

input { window: time\_range, service: id }

budget { tokens.max: 8k, latency\_ms.max: 120000, tools.max\_calls: 10 }

step detect(role: operator)

tool: cost\_anomaly\_detector

out: anomaly\_report

gate: { anomaly.p\_value <= 0.01 }

step correlate(role: witness)

tool: seg\_query

in: anomaly\_report

out: change\_set, seg\_slice

gate: { change\_set.nonempty? }

step contain(role: operator)

tool: feature\_flags

in: change\_set

out: toggles\_off

policy: least\_disruption

step recover(role: builder)

tool: rollback\_manager

in: change\_set

out: rollback\_done

gate: { canary.ok && error\_budget.burn\_rate < 1.0 }

step witness(role: witness)

out: vif\_packet, decision\_node

**κ-gating.** If UQ.calibration deteriorates or RS-lift<0, **abstain** and escalate to HITL.

**26.5 Safe Autoremediation: Guardrails & Playbooks**

**Guardrails.**

* **Scope caps:** only toggle flags / revert *recent* change set.
* **Budget caps:** token/latency ceilings; auto-stop at 80% of cap.
* **Canaries:** ≤5% traffic; promote only if SLOs green for N intervals.
* **Two-key:** SEV-1 or *compliance-critical* flows require human approval.
* **Snap-first:** all actions bind to a snapshot\_id; rollback is deterministic.
* **Witnessing:** every step emits VIF & SEG edges.

**Playbook 1 — Cost Runaway (RAG).**

pipeline remediate\_cost\_runaway

step adjust\_rag(role: operator)

tool: config\_editor

in: { top\_k: current }

out: { top\_k: max(current-3, 3) }

gate: { rs\_lift >= -0.01 } # do-no-harm guard

step swap\_embedder(role: operator)

tool: model\_switch

in: { embedder: "v-large" }

out: { embedder: "v-medium" }

gate: { drift.delta <= 0.02 }

step canary(role: operator)

tool: canary\_runner

out: canary\_ok

gate: { slo.p95\_latency <= target && rs\_lift >= -0.005 }

**Playbook 2 — Prompt-Rot / Drift.**

pipeline remediate\_prompt\_rot

step calibrate(role: verifier)

tool: temp\_sweep

out: ece

gate: { ece <= 0.03 }

step template\_fallback(role: builder)

tool: prompt\_templates

in: { template: "main" }

out: { template: "fallback\_v2" }

gate: { goldset.f1 >= policy.min\_f1 }

step retrain\_calibrator(role: builder)

tool: calibrator\_fit

out: new\_calibrator

**Playbook 3 — Security Event (SSRF/Exfil).**

pipeline remediate\_security\_event

step revoke(role: operator) tool: token\_revoker

step isolate(role: operator) tool: sandbox\_enforcer

step blocklist(role: operator) tool: egress\_filter

step audit(role: witness) -> seg\_incident\_package

**26.6 Post-Incident: Evidence, Learning, and SDF-CVF**

**Postmortem (blameless, time-boxed).**

* **What/When:** timeline from SEG; attach spans/logs/decisions.
* **Why:** primary + contributing factors; model: **technical + organizational**.
* **So what:** user impact, legal/compliance notes; cost.
* **Now what:** *atomic* change set across **code/docs/tags/traces** (SDF-CVF), new tests, updated runbooks, monitors.

**SEG export pack contents.**

* Time-sliced subgraph (JSON-LD).
* VIF packet bundle (model/tool/prompt hashes).
* Redacted logs/metrics/traces.
* Decisions & ADRs.
* Retention schedule & labels.

**DORA-aligned updates.**

* CFR, MTTR, deployment frequency, lead time — tied to incident cohort.

**26.7 KPIs, SLOs, and Error Budgets (Ops)**

* **Detect:** MTTD p50/p95.
* **Contain:** MTTC; % incidents contained by canary.
* **Restore:** MTTR; rollback success rate.
* **Correctness:** RCA completeness (graph coverage ≥ X%), replay fidelity=100%.
* **Safety:** κ-abstention true-positive rate; **HITL latency** p95.
* **Cost:** token/latency budget adherence; **auto-remediation success** without regression ≥ Y%.
* **Quality:** RS-lift post-fix ≥ 0; ECE within band.

**26.8 End-to-End Example (Condensed Timeline)**

* **00:00** Pager triggers: cost anomaly (p=0.007).
* **00:02** IC engaged; κ gates new complex chains.
* **00:05** SEG query links spike to change\_set C-142 (RAG top-k=20).
* **00:09** Playbook “cost runaway” lowers top\_k to 8; canary 5% passes.
* **00:15** Promote canary to 50%; SLOs hold; RS-lift −0.003 (within guard).
* **00:20** Full traffic restored; auto-rollback prepared as shadow.
* **00:35** Decision node promote\_fix recorded; VIF bundle sealed.
* **02:00** Postmortem draft with SEG export pack; SDF-CVF commit adds tests to guard top-k regression.

**26.9 Checklists**

**During incident**

* IC named; comms set.
* Snapshot and time window frozen.
* Forensic pipeline redaction verified.
* SEG edges: symptoms↔changes linked.
* κ-gates active on high-risk steps.
* Canaries before production switches.
* VIF packets emitted at every boundary.

**After incident**

* Postmortem published; export pack archived.
* SDF-CVF atomic patch merged (code/docs/tags/traces).
* Monitors/runbooks updated; drills scheduled.
* Error-budget policy reviewed.

**Part XI — Reference Implementations**

**Chapter 27. Data Schemas — Atom/Edge (CMC/HHNI), SEG JSON-LD, and ACL AST**

Aim. Make **memory**, **evidence**, and **orchestration** concrete and interoperable. This chapter defines canonical schemas for (1) CMC/HHNI atoms & edges, (2) the Shared Evidence Graph (SEG) in JSON-LD, and (3) the APOE orchestration AST for ACL. All schemas are versioned, time-sliced, and provenance-aware (VIF).

**27.0 Conventions (applies to all schemas)**

* **ID**: uuidv7 (string).
* **Timestamps**: RFC 3339/ISO-8601 UTC (2025-10-17T12:34:56Z).
* **Versioning**: schema\_version uses semver; breaking changes bump MAJOR.
* **Hashing**: sha256\_hex of canonical bytes (UTF-8, normalized newlines).
* **Time-slicing**: valid\_from, valid\_to (null = open-ended).
* **Provenance** (VIF tuple): (model\_id, weights\_hash, prompt\_template\_id, tool\_ids[], snapshot\_id) recorded on any write.

**27.1 CMC/HHNI — Atom & Edge Schemas**

**27.1.1 Atom (memory node)**

**Purpose.** The smallest addressable unit of memory (or a container “molecule”). Every atom is embeddable, indexed (HHNI path), and snapshotted.

{

"$id": "https://aimos.dev/schema/atom.json",

"$schema": "https://json-schema.org/draft/2020-12/schema",

"title": "AIMOS Atom",

"type": "object",

"required": ["id","schema\_version","kind","modality","snapshot\_id","created\_at","content\_ref","embedding","hhni","vif"],

"properties": {

"id": { "type": "string", "pattern": "^[a-f0-9-]{36}$" },

"schema\_version": { "type": "string", "pattern": "^[0-9]+\\.[0-9]+\\.[0-9]+$" },

"kind": { "enum": ["atom","molecule"] },

"modality": {

"type": "string",

"description": "Free taxonomy with prefixes; examples below"

},

"modality\_meta": {

"type": "object",

"description": "Modality-specific fields",

"additionalProperties": true

},

"content\_ref": {

"type": "object",

"required": ["media\_type"],

"properties": {

"inline": { "type": "string", "description": "Optional small content (≤ 64KB)" },

"uri": { "type": "string", "description": "Content-addressed or object store URI" },

"bytes\_sha256": { "type": "string", "pattern": "^[a-f0-9]{64}$" },

"media\_type": { "type": "string" }

},

"oneOf": [

{ "required": ["inline"] },

{ "required": ["uri","bytes\_sha256"] }

]

},

"embedding": {

"type": "object",

"required": ["model\_id","dim"],

"properties": {

"model\_id": { "type": "string" },

"dim": { "type": "integer", "minimum": 1 },

"vector": {

"type": "array",

"items": { "type": "number" },

"minItems": 1

},

"vector\_ref": { "type": "string", "description": "If vector is stored in external vector index" }

},

"oneOf": [

{ "required": ["vector"] },

{ "required": ["vector\_ref"] }

]

},

"tags": {

"type": "array",

"items": {

"type": "object",

"required": ["key","weight"],

"properties": {

"key": { "type": "string" },

"value": { "type": "string" },

"weight": { "type": "number", "minimum": 0, "maximum": 1 }

}

}

},

"tpv": {

"type": "object",

"description": "Tag Priority Vector dynamics",

"properties": {

"priority": { "type": "number", "minimum": 0, "maximum": 1 },

"relevance": { "type": "number", "minimum": 0, "maximum": 1 },

"decay\_tau": { "type": "number", "minimum": 0 },

"last\_access\_at": { "type": "string" }

}

},

"hhni": {

"type": "object",

"required": ["path","dependency\_hash"],

"properties": {

"path": {

"type": "array",

"description": "Fractal path: system→section→paragraph→sentence→word→subword",

"items": { "type": "string" }

},

"parent\_id": { "type": "string" },

"children\_count": { "type": "integer", "minimum": 0 },

"dependency\_hash": { "type": "string", "pattern": "^[a-f0-9]{64}$" }

}

},

"metrics": {

"type": "object",

"properties": {

"qs": { "type": "number", "minimum": 0, "maximum": 1 },

"ids": { "type": "number", "minimum": 0, "maximum": 1 },

"dd": { "type": "number", "minimum": 0, "maximum": 1 },

"rs": { "type": "number", "minimum": 0, "maximum": 1 }

}

},

"snapshot\_id": { "type": "string" },

"created\_at": { "type": "string" },

"updated\_at": { "type": "string" },

"valid\_from": { "type": "string" },

"valid\_to": { "type": ["string","null"] },

"vif": {

"type": "object",

"required": ["model\_id","weights\_hash","prompt\_template\_id","tool\_ids","writer"],

"properties": {

"model\_id": { "type": "string" },

"weights\_hash": { "type": "string", "pattern": "^[a-f0-9]{64}$" },

"prompt\_template\_id": { "type": "string" },

"tool\_ids": { "type": "array", "items": { "type": "string" } },

"writer": { "type": "string", "description": "agent id or human id" }

}

},

"etag": { "type": "string" }

}

}

**Modality taxonomy (non-exhaustive).**

* text/plain, text/markdown
* code:ts, code:py, code:sql, code:dockerfile
* event:log, event:span, event:metric
* prompt:user, prompt:system, prompt:tool
* asset:image, asset:audio, asset:pdf

**Notes.**

* inline is capped for small payloads; large content goes via uri + bytes\_sha256.
* hhni.dependency\_hash is computed over (parent\_id, path, tags, content\_hash) to detect integrity breaks.
* Either embedding.vector or embedding.vector\_ref must exist (store vectors where your stack prefers).

**27.1.2 Edge (memory relation)**

**Purpose.** Typed, weighted links between atoms/molecules; time-sliced; provenance-stamped.

{

"$id": "https://aimos.dev/schema/edge.json",

"$schema": "https://json-schema.org/draft/2020-12/schema",

"title": "AIMOS Memory Edge",

"type": "object",

"required": ["id","schema\_version","source\_id","target\_id","rel","weight","snapshot\_id","created\_at","vif"],

"properties": {

"id": { "type": "string" },

"schema\_version": { "type": "string" },

"source\_id": { "type": "string" },

"target\_id": { "type": "string" },

"rel": {

"type": "string",

"enum": [

"contains","is\_part\_of","relates\_to","depends\_on","is\_summary\_of",

"derives","supports","contradicts","annotates","temporal\_next"

]

},

"weight": { "type": "number", "minimum": 0, "maximum": 1 },

"confidence": { "type": "number", "minimum": 0, "maximum": 1 },

"policy": { "type": "string", "description": "e.g., allowed, restricted, redacted" },

"reason": { "type": "string" },

"snapshot\_id": { "type": "string" },

"created\_at": { "type": "string" },

"valid\_from": { "type": "string" },

"valid\_to": { "type": ["string","null"] },

"vif": {

"type": "object",

"required": ["model\_id","weights\_hash","prompt\_template\_id","tool\_ids","writer"],

"properties": {

"model\_id": { "type": "string" },

"weights\_hash": { "type": "string" },

"prompt\_template\_id": { "type": "string" },

"tool\_ids": { "type": "array", "items": { "type": "string" } },

"writer": { "type": "string" }

}

}

}

}

**Edge semantics.**

* contains/is\_part\_of encode HHNI hierarchy.
* relates\_to (semantic similarity) is pruned by Adaptive Weight Pruning.
* depends\_on governs build/order constraints.
* derives/supports/contradicts/annotates mirror SEG semantics for cross-linking memory↔evidence.

**27.1.3 Snapshot (content-addressed, Merkle)**

{

"$id": "https://aimos.dev/schema/snapshot.json",

"$schema": "https://json-schema.org/draft/2020-12/schema",

"title": "AIMOS Snapshot",

"type": "object",

"required": ["id","schema\_version","created\_at","root\_hash","stats","signature"],

"properties": {

"id": { "type": "string" },

"schema\_version": { "type": "string" },

"created\_at": { "type": "string" },

"parent\_id": { "type": ["string","null"] },

"root\_hash": { "type": "string", "pattern": "^[a-f0-9]{64}$" },

"stats": {

"type": "object",

"properties": {

"atoms": { "type": "integer" },

"edges": { "type": "integer" }

}

},

"signature": {

"type": "object",

"required": ["algo","signed\_by","sig\_hex"],

"properties": {

"algo": { "enum": ["ed25519"] },

"signed\_by": { "type": "string" },

"sig\_hex": { "type": "string" }

}

},

"notes": { "type": "string" }

}

}

**27.2 SEG — Shared Evidence Graph (JSON-LD)**

**Purpose.** Interoperable, machine-readable lineage for claims, sources, tool calls, and decisions. Time-sliced, exportable, and queryable with graph tooling.

**27.2.1 JSON-LD @context**

{

"@context": {

"seg": "https://aimos.dev/seg#",

"prov": "http://www.w3.org/ns/prov#",

"xsd": "http://www.w3.org/2001/XMLSchema#",

"schema": "http://schema.org/",

"id": "@id",

"type": "@type",

"validFrom": { "@id": "seg:validFrom", "@type": "xsd:dateTime" },

"validTo": { "@id": "seg:validTo", "@type": "xsd:dateTime" },

"derivesFrom": { "@id": "seg:derivesFrom", "@type": "@id" },

"supports": { "@id": "seg:supports", "@type": "@id" },

"contradicts": { "@id": "seg:contradicts", "@type": "@id" },

"annotates": { "@id": "seg:annotates", "@type": "@id" },

"generatedAt": { "@id": "prov:generatedAtTime", "@type": "xsd:dateTime" },

"used": { "@id": "prov:used", "@type": "@id" },

"wasAssociatedWith": { "@id": "prov:wasAssociatedWith", "@type": "@id" },

"confidence": { "@id": "seg:confidence", "@type": "xsd:double" },

"uq": { "@id": "seg:uncertaintyVector" },

"vif": { "@id": "seg:provenanceTuple" }

}

}

**27.2.2 Core types**

* **seg:Claim** — a statement the system asserts.
* **seg:Artifact** — any intermediate (snippet, summary, code block).
* **seg:Source** — external reference (doc, URL, dataset).
* **seg:ToolCall** — invocation record (tool id, args hash, result hash).
* **seg:Decision** — adjudicated outcome linked to claims/evidence.
* **seg:Snapshot** — memory state reference that scoped the run.
* **seg:Span/Log/Metric** — operational evidence.

**27.2.3 Example (compact JSON-LD)**

{

"@context": "https://aimos.dev/seg/context.json",

"@graph": [

{

"id": "seg:claim:release\_rc\_0\_3",

"type": "seg:Claim",

"text": "Promote canary of RAG top\_k=8 to 100%.",

"confidence": 0.92,

"uq": { "ece": 0.018, "entropy": 0.23 },

"vif": {

"model\_id": "apoe-gptx-2025-09",

"weights\_hash": "9f2e…",

"prompt\_template\_id": "promote\_canary\_v2",

"tool\_ids": ["canaryRunner","segQuery"],

"snapshot\_id": "S-BF-09D"

},

"generatedAt": "2025-10-17T12:37:00Z",

"derivesFrom": ["seg:artifact:rs\_report","seg:tool:canary\_result"]

},

{

"id": "seg:artifact:rs\_report",

"type": "seg:Artifact",

"mediaType": "application/json",

"sha256": "2ab3…",

"validFrom": "2025-10-17T12:30:00Z",

"validTo": null,

"supports": ["seg:claim:release\_rc\_0\_3"]

},

{

"id": "seg:tool:canary\_result",

"type": "seg:ToolCall",

"toolId": "canaryRunner",

"argsHash": "a1b2…",

"resultHash": "c3d4…",

"wasAssociatedWith": "agent:operator",

"used": "seg:snapshot:S-BF-09D"

},

{

"id": "seg:decision:DR-778",

"type": "seg:Decision",

"text": "Approved by two-key HITL; promote canary.",

"validFrom": "2025-10-17T12:38:00Z",

"annotates": ["seg:claim:release\_rc\_0\_3"],

"supports": ["seg:claim:release\_rc\_0\_3"]

},

{

"id": "seg:snapshot:S-BF-09D",

"type": "seg:Snapshot",

"rootHash": "aa77…",

"generatedAt": "2025-10-17T12:29:55Z"

}

]

}

**Design notes.**

* All edges are **additive**; contradictions are explicit contradicts links (never destructive deletes).
* **Time-slice** by validFrom/validTo on any node/edge.
* The **VIF tuple** on claims and tool calls guarantees replayability.

**27.3 ACL — APOE Chain Language AST**

**Purpose.** A typed, versioned JSON AST for pipelines/steps/gates/budgets so plans are **diff-able, testable, and auditable**.

**27.3.1 Top-level schema**

{

"$id": "https://aimos.dev/schema/acl-pipeline.json",

"$schema": "https://json-schema.org/draft/2020-12/schema",

"title": "ACL Pipeline AST",

"type": "object",

"required": ["id","schema\_version","name","inputs","outputs","budgets","steps","flow","policies"],

"properties": {

"id": { "type": "string" },

"schema\_version": { "type": "string" },

"name": { "type": "string" },

"description": { "type": "string" },

"inputs": { "$ref": "#/$defs/ioSchema" },

"outputs": { "$ref": "#/$defs/ioSchema" },

"budgets": { "$ref": "#/$defs/budget" },

"policies": { "$ref": "#/$defs/policies" },

"steps": {

"type": "array",

"items": { "$ref": "#/$defs/step" },

"minItems": 1

},

"flow": {

"type": "array",

"description": "DAG edges: from → to",

"items": {

"type": "object",

"required": ["from","to"],

"properties": { "from": { "type": "string" }, "to": { "type": "string" } }

}

}

},

"$defs": {

"ioSchema": {

"type": "object",

"additionalProperties": {

"type": "object",

"required": ["type"],

"properties": {

"type": { "enum": ["string","number","boolean","object","array","artifact","evidence","any"] },

"required": { "type": "boolean" },

"description": { "type": "string" }

}

}

},

"budget": {

"type": "object",

"properties": {

"tokens\_max": { "type": "integer", "minimum": 0 },

"latency\_ms\_max": { "type": "integer", "minimum": 0 },

"tools\_max\_calls": { "type": "integer", "minimum": 0 },

"cost\_usd\_max": { "type": "number", "minimum": 0 },

"gpu\_class": { "type": "string" }

}

},

"policies": {

"type": "object",

"properties": {

"kappa\_abstain\_threshold": { "type": "number", "minimum": 0, "maximum": 1 },

"risk\_class": { "enum": ["low","medium","high","critical"] },

"hitl\_required": { "type": "boolean" }

}

},

"gate": {

"type": "object",

"required": ["id","kind","expr"],

"properties": {

"id": { "type": "string" },

"kind": { "enum": ["spec","test","safety","provenance","uq","cost","custom"] },

"expr": { "type": "string", "description": "CEL-like or JSONLogic expression" },

"severity": { "enum": ["warn","error","block"] },

"on\_fail": { "enum": ["abort","repair","quarantine","escalate"] }

}

},

"step": {

"type": "object",

"required": ["id","role","tool","in","out","budgets","gates","witness"],

"properties": {

"id": { "type": "string" },

"role": {

"enum": ["planner","retriever","reasoner","verifier","builder","critic","operator","witness"]

},

"tool": {

"type": "object",

"required": ["id"],

"properties": {

"id": { "type": "string" },

"args": { "type": "object", "additionalProperties": true },

"prompt\_template\_id": { "type": "string" }

}

},

"in": { "type": "object", "additionalProperties": true },

"out": { "type": "object", "additionalProperties": true },

"budgets": { "$ref": "#/$defs/budget" },

"gates": { "type": "array", "items": { "$ref": "#/$defs/gate" } },

"witness": {

"type": "object",

"properties": {

"record\_artifacts": { "type": "boolean" },

"record\_inputs": { "type": "boolean" },

"record\_outputs": { "type": "boolean" },

"vif\_emit": { "type": "boolean" }

}

},

"retries": {

"type": "object",

"properties": {

"max\_attempts": { "type": "integer", "minimum": 0 },

"backoff\_ms": { "type": "integer", "minimum": 0 }

}

},

"kappa\_overrides": {

"type": "object",

"properties": {

"abstain\_threshold": { "type": "number", "minimum": 0, "maximum": 1 }

}

}

}

}

}

}

**27.3.2 Example: *remediate\_cost\_runaway* (from Ch.26) — AST instance**

{

"id": "pipe:remediate\_cost\_runaway",

"schema\_version": "1.0.0",

"name": "Remediate Cost Runaway",

"description": "Lower RAG cost without harming retrieval quality.",

"inputs": { "window": { "type": "object", "required": true } },

"outputs": { "decision": { "type": "evidence" } },

"budgets": { "tokens\_max": 12000, "latency\_ms\_max": 180000, "tools\_max\_calls": 20, "cost\_usd\_max": 5.0 },

"policies": { "kappa\_abstain\_threshold": 0.35, "risk\_class": "medium", "hitl\_required": false },

"steps": [

{

"id": "adjust\_rag",

"role": "operator",

"tool": { "id": "config\_editor", "args": { "param": "rag.top\_k", "delta": -3 } },

"in": { "window": "${inputs.window}" },

"out": { "new\_config": "${result}" },

"budgets": { "tokens\_max": 2000, "latency\_ms\_max": 30000, "tools\_max\_calls": 2 },

"gates": [

{ "id": "do\_no\_harm", "kind": "custom", "expr": "metrics.rs\_lift >= -0.01", "severity": "block", "on\_fail": "abort" }

],

"witness": { "record\_artifacts": true, "vif\_emit": true }

},

{

"id": "canary",

"role": "operator",

"tool": { "id": "canary\_runner", "args": { "traffic\_pct": 5 } },

"in": { "config": "${steps.adjust\_rag.out.new\_config}" },

"out": { "canary\_report": "${result}" },

"budgets": { "latency\_ms\_max": 60000 },

"gates": [

{ "id": "slo\_ok", "kind": "test", "expr": "slo.p95\_latency <= target && rs\_lift >= -0.005", "severity": "block", "on\_fail": "rollback" }

],

"witness": { "record\_artifacts": true, "vif\_emit": true }

}

],

"flow": [

{ "from": "adjust\_rag", "to": "canary" }

]

}

**27.4 Integrity, Indexing, and Performance Notes**

* **Referential integrity**: Edges source\_id/target\_id must exist in the same snapshot\_id (or specify cross-snapshot links with explicit from\_snapshot/to\_snapshot fields).
* **Uniqueness**: (source\_id, target\_id, rel, valid\_from) must be unique.
* **Indices** (recommended):
  + Atoms: snapshot\_id, modality, hhni.path[\*], metrics.rs, tpv.priority.
  + Edges: snapshot\_id, rel, source\_id, target\_id, valid\_from.
  + Vectors: embedding.vector (cosine) or embedding.vector\_ref.
* **Concurrency**: optimistic writes with etag; server rejects on mismatch.
* **Compression**: store content\_ref.inline\_gzip for verbose text; keep bytes\_sha256 of raw.
* **Security**: redact PII at ingest; policy fields propagate to SEG exports.
* **Calibration**: only the **reader** recomputes rs = f(qs, ids, dd); writers should not set rs unless they also provide a VIF for the scoring step.

**27.5 Worked Example (Atom + Edge + SEG round-trip)**

1. **Write** a text atom (user prompt → atomized):

{

"id": "a-7c9c…",

"schema\_version": "1.1.0",

"kind": "atom",

"modality": "prompt:user",

"content\_ref": { "inline": "Summarize failed transfers by tip type last week.", "media\_type": "text/plain" },

"embedding": { "model\_id": "embed-004", "dim": 768, "vector\_ref": "vec:a-7c9c…" },

"tags": [{ "key": "topic", "value": "payments", "weight": 0.9 }],

"tpv": { "priority": 0.8, "relevance": 0.87, "decay\_tau": 604800 },

"hhni": { "path": ["system:aiops","section:queries","sentence:12"], "dependency\_hash": "5fe1…" },

"metrics": { "qs": 0.82, "ids": 0.76, "dd": 0.03 },

"snapshot\_id": "S-BF-09D",

"created\_at": "2025-10-17T12:30:01Z",

"updated\_at": "2025-10-17T12:30:01Z",

"valid\_from": "2025-10-17T12:30:01Z",

"valid\_to": null,

"vif": {

"model\_id": "apoe-gptx-2025-09",

"weights\_hash": "9f2e…",

"prompt\_template\_id": "atomize\_v3",

"tool\_ids": ["embedder-004"],

"writer": "agent:archivist"

},

"etag": "W/\"e1f-7c3\""

}

1. **Relate** it to a retrieval artifact with an edge:

{

"id": "e-22a1…",

"schema\_version": "1.0.0",

"source\_id": "a-7c9c…",

"target\_id": "a-42bd…",

"rel": "relates\_to",

"weight": 0.87,

"confidence": 0.9,

"policy": "allowed",

"reason": "semantic\_cosine>=0.85",

"snapshot\_id": "S-BF-09D",

"created\_at": "2025-10-17T12:30:05Z",

"valid\_from": "2025-10-17T12:30:05Z",

"valid\_to": null,

"vif": { "model\_id":"reranker-02","weights\_hash":"1cc…","prompt\_template\_id":"rerank\_v2","tool\_ids":["vecIndex"],"writer":"agent:retriever" }

}

1. **Emit SEG** for a claim derived from these atoms; export the JSON-LD bundle as part of the run’s VIF.

**27.6 Validation & Tooling**

* **Static validation**: JSON Schema validators (draft 2020-12).
* **Semantic validation**:
  + Atom: kind=atom ⇒ hhni.parent\_id==null or points to molecule.
  + Edge: rel in {"contains","is\_part\_of"} ⇒ (source.kind, target.kind) is lawful (molecule↔atom rules).
  + Time: valid\_from <= valid\_to || valid\_to==null.
* **Canonicalization for hashing**: RFC 8785 (JSON Canonicalization Scheme) before computing bytes\_sha256 or root\_hash.
* **Migrations**: store schema\_version; supply mappers vN→vN+1 with SEG-logged Decisions.

**27.7 Minimal APIs (shape only)**

* **Memory API**
  + POST /atoms (validate + snapshot bind)
  + POST /edges
  + GET /atoms/{id}?as\_of=… (time-slice)
  + POST /query/vector (kNN over embedding)
* **Evidence API**
  + POST /seg/bundle (ingest JSON-LD)
  + GET /seg/graph?as\_of=… (subgraph extract)
* **Orchestration API**
  + POST /acl/pipelines (AST)
  + POST /acl/run → emits Witness/VIF + SEG links

All **read-only** export endpoints include X-Snapshot-Id in headers for replay.

**27.8 Checklists**

**Atom write**

* Inline vs URI chosen; hash computed.
* Embedding present (vector or ref).
* HHNI path + dependency hash set.
* VIF tuple filled; snapshot bound.

**Edge write**

* Relation lawful; self-edge forbidden.
* Weight/confidence in [0,1].
* Time-slice fields set; VIF present.

**SEG export**

* Context URL resolves locally or is bundled.
* All referenced nodes reachable.
* ValidFrom/ValidTo reflect change window.
* Decision nodes annotate and/or support claims.

**Part XI — Reference Implementations**

**Chapter 28. APIs — Memory, Orchestration, and Audit (read-only)**

Aim. Expose AIMOS/APOE as **clean, versioned, and provable** services. This chapter defines endpoint shapes, auth scopes, pagination, filtering, error models, idempotency, and examples. It aligns with the schemas in Ch.27 (Atom/Edge, SEG JSON-LD, ACL AST) and the invariants CMC/APOE/VIF/SEG/SDF-CVF.

**28.0 Design goals**

* **Deterministic contracts** (APOE): typed I/O, explicit budgets, reproducible runs.
* **Memory-first** (CMC): snapshot-bound reads/writes; time-sliced queries.
* **Provenance by default** (VIF): every mutating call emits a VIF tuple.
* **Graph substrate** (SEG): lineage is queryable; contradictions are first-class.
* **Atomic evolution** (SDF-CVF): parity gates enforced at API boundaries.

**28.1 Base, versions, and auth**

* **Base URL**: https://api.aimos.dev/v1/ (semantic, additive minor versions; breaking changes => v2).
* **Auth**: OAuth2/OIDC bearer JWT **or** mTLS.
* **Tenancy**: X-Tenant: {org\_id} (required).
* **Scopes** (minimum set):

| **Scope** | **Purpose** |
| --- | --- |
| aimos.memory.read | Read atoms/edges/snapshots, vector queries |
| aimos.memory.write | Create/update atoms/edges, open snapshots |
| aimos.orch.read | Read pipelines, runs, traces |
| aimos.orch.write | Create/update pipelines, start/cancel runs |
| aimos.audit.read | Read SEG bundles, VIF tuples, replays (read-only) |

**28.2 Cross-cutting conventions**

* **Content-Type**: application/json; charset=utf-8 (SEG bundles: application/ld+json).
* **Idempotency**: Idempotency-Key (UUID v4); server stores 24h hash of request → response.
* **ETags/optimistic control**: If-Match: W/"etag" on mutable resources.
* **Time-slicing**: ?as\_of=2025-10-17T12:34:56Z or ?from=…&to=….
* **Pagination**: cursor model — limit (≤ 500), cursor; response includes next\_cursor.
* **Rate limits**: headers X-RateLimit-Limit/Remaining/Reset.
* **Correlation**: X-Request-Id echoed; also in error bodies and SEG ToolCall nodes.

**Error model (uniform):**

{

"error": {

"code": "MEMORY\_INVALID\_HHNI",

"message": "hhni.path must be non-empty and canonical.",

"status": 400,

"retryable": false,

"correlation\_id": "req\_9b2f…",

"details": { "field": "hhni.path[0]" }

}

}

Common codes (subset):

* UNAUTHENTICATED, PERMISSION\_DENIED, NOT\_FOUND, ALREADY\_EXISTS,  
  FAILED\_PRECONDITION, INVALID\_ARGUMENT, RESOURCE\_EXHAUSTED,  
  ABORTED, CONFLICT, TOO\_MANY\_REQUESTS, INTERNAL.

**28.3 Memory API**

**28.3.1 Atoms**

* **POST /atoms** — create atom (Ch.27.1.1).
  + Scopes: aimos.memory.write
  + Headers: Idempotency-Key, X-Snapshot-Id *(optional; if absent, server opens one)*
  + On success: 201 Created + body; headers: X-Snapshot-Id, ETag
* **GET /atoms/{id}** — fetch one (time-slice via as\_of).
  + Scopes: aimos.memory.read
* **GET /atoms** — list/filter.
  + Query: modality=code:ts, tag=topic:payments, path=system:…, as\_of=…, limit, cursor
* **POST /query/vector** — kNN over embeddings (returns atom ids + scores).
  + Body: { "model\_id":"embed-004", "k":10, "vector":[…], "filters":{ "modality":"text/plain" } }

**Create atom — example**

POST /v1/atoms

Authorization: Bearer …

Content-Type: application/json

Idempotency-Key: 9f7d6c8e-…

{

"schema\_version": "1.1.0",

"kind": "atom",

"modality": "prompt:user",

"content\_ref": { "inline": "Summarize failed transfers by tip type last week.", "media\_type": "text/plain" },

"embedding": { "model\_id": "embed-004", "vector": [0.12, -0.03, …], "dim": 768 },

"tags": [ { "key": "topic", "value": "payments", "weight": 0.9 } ],

"hhni": { "path": ["system:aiops","section:queries","sentence:12"], "dependency\_hash": "5fe1…" },

"tpv": { "priority": 0.8, "relevance": 0.87, "decay\_tau": 604800 }

}

**200/201 response (abridged)**

{ "id": "a\_7c9c…", "snapshot\_id": "S\_BF\_09D", "etag": "W/\"e1f-7c3\"", "created\_at": "2025-10-17T12:30:01Z", "vif": { "model\_id":"apoe-gptx-2025-09", "weights\_hash":"9f2e…", "prompt\_template\_id":"atomize\_v3", "tool\_ids":["embedder-004"], "writer":"agent:archivist" } }

**28.3.2 Edges**

* **POST /edges** — create edge (Ch.27.1.2).
  + Validates relation lawfulness, time-slice, and existence in snapshot.
* **GET /edges/{id}**, **GET /edges** — filter by rel, source\_id, target\_id, as\_of.

**28.3.3 Snapshots**

* **POST /snapshots** — close/open snapshot (Merkle root computed).
  + Body: { "action":"close" | "open", "notes":"…" }
  + Returns closed snapshot metadata (Ch.27.1.3).
* **GET /snapshots/{id}**, **GET /snapshots** — list, filter by time window, stats.

**28.4 Orchestration API (APOE/ACL)**

**28.4.1 Pipelines (ACL AST)**

* **POST /acl/pipelines** — create/update pipeline definition.
  + Scopes: aimos.orch.write (ETag required for updates).
  + Validates AST against schema (Ch.27.3) and static gates.
* **GET /acl/pipelines/{id}** — fetch one (versioned).
* **GET /acl/pipelines** — list; filters: risk\_class, updated\_from, owner.

**28.4.2 Runs**

* **POST /acl/runs** — start a run from pipeline.
  + Body (shape):
  + {
  + "pipeline\_id": "pipe:remediate\_cost\_runaway",
  + "inputs": { "window": { "from": "2025-10-10", "to": "2025-10-17" } },
  + "budgets\_override": { "tokens\_max": 14000 },
  + "snapshot\_id": "S\_BF\_09D",
  + "policies": { "hitl\_required": false }
  + }
  + Returns run\_id, initial state, and **witness channel**.
* **GET /acl/runs/{run\_id}** — status (pending|running|abstained|failed|succeeded|quarantined|rolled\_back).
* **GET /acl/runs** — filter by pipeline\_id, status, from/to, actor.
* **POST /acl/runs/{run\_id}:cancel** — best-effort cancel.
* **GET /acl/runs/{run\_id}/trace** — **witness trace**: steps, gates, UQ, artifacts (redacted by policy).
* **GET /acl/runs/{run\_id}/budget** — live budget ledger (tokens/latency/tools/cost).

**28.4.3 HITL actions (if enabled by risk)**

* **POST /acl/runs/{run\_id}/hitl/approve** — two-key approval (requires second human approval; enforced role).
* **POST /acl/runs/{run\_id}/hitl/reject** — rejection with rationale (SEG decision emitted).

**28.5 Audit API (read-only)**

**Immutable & replayable.** Any POST/PUT/PATCH/DELETE to /audit/\* returns 405 Method Not Allowed.

* **GET /audit/seg/bundles/{id}** — retrieve a JSON-LD SEG bundle (Ch.27.2) for a run/claim.
  + Query: format=compact|expanded, as\_of, includeArtifacts=true|false.
* **GET /audit/seg/lineage** — parametric lineage query.
  + Query: claim\_id=… or artifact\_sha256=… or decision\_id=…; server returns subgraph.
* **GET /audit/vif/{artifact\_id}** — VIF tuple for any stored artifact.
* **GET /audit/replay/{run\_id}** — dry replay plan (no external side-effects); returns **diff** of outputs, calibration deltas, and a replay SEG.
* **GET /audit/snapshots/{id}** — snapshot metadata + root hash, signer, parent link.
* **GET /audit/runs/{run\_id}/witness** — read-only witness channel export (events, UQ, gate outcomes).

**28.6 Filtering, sorting, and pagination (examples)**

* **List atoms (latest), payments topic, 200/page**

GET /v1/atoms?tag=topic:payments&limit=200

Response:

{ "items":[{…},{…}], "next\_cursor":"eyJwYWdlIjoyfQ==" }

* **Lineage for claim (time-slice to last week)**

GET /v1/audit/seg/lineage?claim\_id=seg:claim:release\_rc\_0\_3&from=2025-10-10T00:00:00Z&to=2025-10-17T00:00:00Z

* **Vector query within code only**

POST /v1/query/vector

{

"model\_id":"embed-004",

"k":25,

"vector":[…],

"filters":{ "modality":"code:ts", "tag":"repo:checkout" }

}

**28.7 Idempotency, concurrency, and retries**

* **Idempotency-Key**: identical POST /atoms within 24h returns same 201 body; later **updates** require PATCH with If-Match.
* **Optimistic updates**: If-Match required for PATCH /atoms/{id} (if enabled); 412 Precondition Failed on ETag mismatch.
* **Safe retries**: servers mark **retryable** in error model; clients backoff with jitter when receiving 429/503/504.

**28.8 Budget & policy headers**

* Requests MAY provide **intent** headers to aid routing/quotas:
  + X-Intent: retrieval|orchestration|export
  + X-Risk-Class: low|medium|high|critical
* Responses include **budget usage** snapshot:
  + X-Budget-Tokens-Used, X-Budget-Latency-Ms, X-Budget-Tools-Calls, X-Budget-Cost-Usd.

**28.9 Example flows**

**A) Write → Snapshot → Query → Lineage**

1. Create atom → 201 (VIF emitted, snapshot bound S\_BF\_09D)
2. Close snapshot:

POST /v1/snapshots

{ "action":"close", "notes":"ingest batch 42" }

1. Vector search → collect related atoms
2. Emit claim via pipeline run; fetch audit bundle:

GET /v1/audit/seg/bundles/claim:release\_rc\_0\_3?format=compact

**B) Start an APOE run and stream witness**

1. Start:

POST /v1/acl/runs

{ "pipeline\_id":"pipe:remediate\_cost\_runaway", "inputs":{…}, "snapshot\_id":"S\_BF\_09D" }

1. Poll or subscribe (SSE/WebSocket optional) to /v1/acl/runs/{id} and /trace
2. If HITL required, POST /hitl/approve twice with distinct human actors
3. Read-only audit:

GET /v1/audit/replay/{run\_id}

**28.10 OpenAPI (excerpt, abridged)**

openapi: 3.1.0

info: { title: AIMOS/APOE API, version: v1 }

servers: [ { url: https://api.aimos.dev/v1 } ]

components:

securitySchemes:

oauth2:

type: oauth2

flows:

clientCredentials:

tokenUrl: https://auth.aimos.dev/oauth/token

scopes:

aimos.memory.read: Read memory

aimos.memory.write: Write memory

aimos.orch.read: Read orchestration

aimos.orch.write: Write orchestration

aimos.audit.read: Read audit

security:

- oauth2: [ aimos.memory.read ]

paths:

/atoms:

post:

security: [ { oauth2: [ aimos.memory.write ] } ]

requestBody: { content: { application/json: { schema: { $ref: '#/components/schemas/Atom' } } } }

responses:

'201': { description: Created, headers: { X-Snapshot-Id: { schema: { type: string } }, ETag: { schema: { type: string } } } }

'400': { $ref: '#/components/responses/Error' }

/acl/runs:

post:

security: [ { oauth2: [ aimos.orch.write ] } ]

requestBody: { content: { application/json: { schema: { $ref: '#/components/schemas/StartRun' } } } }

responses:

'201': { description: Run started }

/audit/seg/lineage:

get:

security: [ { oauth2: [ aimos.audit.read ] } ]

parameters:

- { name: claim\_id, in: query, schema: { type: string } }

- { name: artifact\_sha256, in: query, schema: { type: string } }

- { name: from, in: query, schema: { type: string, format: date-time } }

- { name: to, in: query, schema: { type: string, format: date-time } }

responses:

'200': { description: SEG subgraph, content: { application/ld+json: {} } }

components:

responses:

Error:

description: Error

content:

application/json:

schema:

type: object

properties:

error:

type: object

properties:

code: { type: string }

message: { type: string }

status: { type: integer }

retryable: { type: boolean }

correlation\_id: { type: string }

schemas:

Atom: { $ref: 'https://aimos.dev/schema/atom.json' }

StartRun:

type: object

required: [ pipeline\_id, inputs, snapshot\_id ]

properties:

pipeline\_id: { type: string }

inputs: { type: object, additionalProperties: true }

budgets\_override: { type: object }

snapshot\_id: { type: string }

**28.11 Security & policy notes**

* **PII redaction at ingest** (server-side policy pack); violations → FAILED\_PRECONDITION with policy\_violation=true.
* **Tool sandboxes & capability tokens**: orchestration calls to tools are proxied; tokens are least-privilege and time-boxed; all tool calls are SEG-logged.
* **Synthetic labeling**: any generated artifact includes machine-readable label fields in SEG (meets synthetic-content labeling duties).

**28.12 Checklists**

**Client integration**

* Set X-Tenant, X-Request-Id, Idempotency-Key on POST.
* Handle next\_cursor; stop on absence.
* Respect retryable and Backoff-Jitter on 429/503/504.
* Send If-Match for updates; handle 412.

**Server readiness**

* Emit VIF tuples for all writes and runs.
* Bind writes to a snapshot; closing snapshot computes Merkle root and signature.
* Enforce scopes per route; block write verbs on /audit/\*.
* Uniform error body across services.

**Part XI — Reference Implementations**

**Chapter 29. SDKs & Clients — JS/TS, Python, and CLI**

Aim. Give builders **production-ready client patterns** for AIMOS/APOE: strict typing, auth, retries with jitter, pagination cursors, idempotency keys, budget headers, and first-class handling of **VIF/SEG** artifacts. All examples conform to the API contracts defined in Ch.28.

**29.0 Design goals (all SDKs)**

* **Typed by default.** Request/response models map 1:1 to the schemas in Ch.27/28.
* **Deterministic retries.** Exponential backoff with jitter on 429/503/504 and error.retryable=true.
* **Idempotency.** Idempotency-Key for all mutating POSTs; safe to retry.
* **Pagination.** Cursor pattern with transparent iterators.
* **Time-slicing.** as\_of / from / to helpers.
* **Budgets & observability.** Surface X-Budget-\* response headers.
* **Tenant-scoped.** Always send X-Tenant.
* **Error model.** Uniform mapping to rich exceptions: code, status, correlation\_id, details.
* **Security.** OAuth2 bearer (or mTLS) injection, zero secret logging.

Environment variables used across examples:

AIMOS\_BASE\_URL=https://api.aimos.dev/v1

AIMOS\_TENANT=acme-co

AIMOS\_TOKEN=eyJhbGciOi...

**29.1 JavaScript / TypeScript SDK**

**29.1.1 Installation & structure**

npm i @aimos/sdk

# or: pnpm add @aimos/sdk

Recommended package layout:

src/

http.ts // fetch wrapper, retries, auth, headers

types.ts // Atom/Edge/SEG/ACL types (generated from OpenAPI)

memory.ts // atoms/edges/snapshots

orch.ts // pipelines/runs

audit.ts // SEG/VIF/replay

paginators.ts // async iterators

index.ts

**29.1.2 Core HTTP with retries & idempotency**

// src/http.ts

import { setTimeout as sleep } from 'node:timers/promises';

import { randomUUID } from 'node:crypto';

export class AimosHTTP {

constructor(

private base = process.env.AIMOS\_BASE\_URL!,

private token = process.env.AIMOS\_TOKEN!,

private tenant = process.env.AIMOS\_TENANT!,

private userAgent = 'aimos-ts/1.0.0'

) {}

private async request<T>(path: string, init: RequestInit & { idempotency?: boolean } = {}): Promise<{ data: T; headers: Headers }> {

const url = `${this.base}${path}`;

const headers = new Headers(init.headers ?? {});

headers.set('Authorization', `Bearer ${this.token}`);

headers.set('X-Tenant', this.tenant);

headers.set('X-Request-Id', crypto.randomUUID());

headers.set('Content-Type', headers.get('Content-Type') || 'application/json');

headers.set('User-Agent', this.userAgent);

if (init.idempotency) headers.set('Idempotency-Key', randomUUID());

let attempt = 0;

/\* retry: 5 attempts, exp backoff + jitter \*/

while (true) {

const res = await fetch(url, { ...init, headers });

const text = await res.text();

const contentType = res.headers.get('Content-Type') || '';

const body = contentType.includes('json') && text ? JSON.parse(text) : text;

if (res.ok) return { data: body as T, headers: res.headers };

const retryable = [429, 503, 504].includes(res.status) || body?.error?.retryable;

if (retryable && attempt < 5) {

attempt++;

const base = Math.min(1000 \* 2 \*\* attempt, 8000); // exp backoff

const jitter = Math.floor(Math.random() \* 250); // jitter

await sleep(base + jitter);

continue;

}

const err = new AimosError(

body?.error?.message || `HTTP ${res.status}`,

body?.error?.code || 'HTTP\_ERROR',

res.status,

body?.error?.correlation\_id || headers.get('X-Request-Id') || ''

);

(err as any).details = body?.error?.details;

throw err;

}

}

get<T>(p: string, q?: Record<string,string|number|boolean|undefined>) {

const qs = q ? '?' + new URLSearchParams(Object.entries(q).filter(([,v]) => v!==undefined).map(([k,v]) => [k,String(v)])).toString() : '';

return this.request<T>(`${p}${qs}`, { method: 'GET' });

}

post<T>(p: string, b?: unknown, idempotency = false) {

return this.request<T>(p, { method: 'POST', body: b ? JSON.stringify(b) : undefined, idempotency });

}

patch<T>(p: string, b?: unknown, etag?: string) {

const headers = etag ? { 'If-Match': etag } : undefined;

return this.request<T>(p, { method: 'PATCH', body: b ? JSON.stringify(b) : undefined, headers });

}

}

export class AimosError extends Error {

constructor(

message: string,

public code: string,

public status: number,

public correlationId: string

) { super(message); }

}

**29.1.3 Typed models & helpers**

*(Types generated from OpenAPI/Ch.28; excerpt)*

// src/types.ts

export type Atom = {

id: string; modality: string;

content\_ref: { inline?: string; uri?: string; media\_type: string };

embedding?: { model\_id: string; dim: number; vector: number[] };

tags?: { key: string; value: string; weight?: number }[];

hhni?: { path: string[]; dependency\_hash?: string };

tpv?: { priority: number; relevance: number; decay\_tau?: number };

snapshot\_id: string; created\_at: string; vif: VIF;

};

export type VIF = {

model\_id: string; weights\_hash: string;

prompt\_template\_id?: string;

tool\_ids?: string[]; writer: string;

};

export type VectorQueryReq = {

model\_id: string; k: number; vector: number[];

filters?: Record<string,string>;

};

export type VectorQueryRes = { results: { id: string; score: number }[] };

**29.1.4 Memory client, paginators, and budgets**

// src/memory.ts

import { AimosHTTP } from './http';

import { Atom, VectorQueryReq, VectorQueryRes } from './types';

export class MemoryClient {

constructor(private http: AimosHTTP) {}

async createAtom(input: Omit<Atom, 'id'|'snapshot\_id'|'created\_at'|'vif'>) {

const { data, headers } = await this.http.post<Atom>('/atoms', input, true);

return { atom: data, snapshotId: headers.get('X-Snapshot-Id')!, etag: headers.get('ETag') };

}

async \*listAtoms(params: { tag?: string; modality?: string; limit?: number; as\_of?: string } = {}) {

let cursor: string | undefined;

do {

const { data } = await this.http.get<{ items: Atom[]; next\_cursor?: string }>('/atoms', { ...params, cursor });

for (const a of data.items) yield a;

cursor = data.next\_cursor;

} while (cursor);

}

async vectorQuery(req: VectorQueryReq) {

const { data } = await this.http.post<VectorQueryRes>('/query/vector', req);

return data.results;

}

async closeSnapshot(notes?: string) {

const { data } = await this.http.post('/snapshots', { action: 'close', notes }, true);

return data;

}

}

**29.1.5 Orchestration & witness trace**

// src/orch.ts

export class OrchestrationClient {

constructor(private http: AimosHTTP) {}

startRun(input: {

pipeline\_id: string; inputs: Record<string,unknown>;

snapshot\_id: string; budgets\_override?: Record<string,number>; policies?: { hitl\_required?: boolean }

}) {

return this.http.post<{ run\_id: string; status: string; started\_at: string }>('/acl/runs', input, true);

}

getRun(runId: string) {

return this.http.get<{ run\_id: string; status: string; budget: Record<string,number> }>(`/acl/runs/${runId}`);

}

getTrace(runId: string) {

return this.http.get<{ steps: unknown[]; gates: unknown[]; artifacts: unknown[] }>(`/acl/runs/${runId}/trace`);

}

/\* polling helper \*/

async \*tail(runId: string, intervalMs = 1200) {

let status: string | undefined;

do {

const { data } = await this.getRun(runId);

status = data.status;

yield data;

if (['succeeded','failed','abstained','quarantined','rolled\_back','cancelled'].includes(status)) break;

await new Promise(r => setTimeout(r, intervalMs));

} while (true);

}

}

**29.1.6 End-to-end example (TS)**

import { AimosHTTP } from './http';

import { MemoryClient } from './memory';

import { OrchestrationClient } from './orch';

const http = new AimosHTTP();

const memory = new MemoryClient(http);

const orch = new OrchestrationClient(http);

(async () => {

// 1) Create an atom (semantic prompt), idempotent

const { atom, snapshotId } = await memory.createAtom({

modality: 'prompt:user',

content\_ref: { inline: 'Summarize failed transfers by tip type last week.', media\_type: 'text/plain' },

embedding: { model\_id: 'embed-004', dim: 768, vector: new Array(768).fill(0) },

tags: [{ key:'topic', value:'payments', weight:0.9 }],

hhni: { path: ['system:aiops','section:queries','sentence:12'] },

tpv: { priority: 0.8, relevance: 0.87, decay\_tau: 604800 }

});

console.log('Atom', atom.id, 'snapshot', snapshotId);

// 2) Close snapshot

await memory.closeSnapshot('ingest batch 42');

// 3) Start a run

const run = await orch.startRun({

pipeline\_id: 'pipe:payments\_weekly\_report',

inputs: { lookback\_days: 7 },

snapshot\_id: snapshotId!

});

const runId = run.data.run\_id;

// 4) Tail status

for await (const frame of orch.tail(runId)) {

console.log('status:', frame.status, 'budget:', frame.budget);

}

// 5) Fetch witness trace

const trace = await orch.getTrace(runId);

console.log('trace steps:', trace.data.steps.length);

})();

**29.2 Python SDK**

**29.2.1 Install & layout**

pip install aimos-sdk

aimos\_sdk/

http.py # session, retries, auth, idempotency

types.py # pydantic models

memory.py

orch.py

audit.py

paginators.py

**29.2.2 Core HTTP with backoff & rich errors**

# aimos\_sdk/http.py

import os, time, uuid, json, random, typing as t

import requests

from dataclasses import dataclass

@dataclass

class AimosError(Exception):

message: str

code: str

status: int

correlation\_id: str

details: t.Optional[dict] = None

class AimosHTTP:

def \_\_init\_\_(self, base=None, token=None, tenant=None, user\_agent='aimos-py/1.0.0', timeout=30):

self.base = base or os.environ['AIMOS\_BASE\_URL']

self.token = token or os.environ['AIMOS\_TOKEN']

self.tenant = tenant or os.environ['AIMOS\_TENANT']

self.timeout = timeout

self.session = requests.Session()

self.session.headers.update({

'Authorization': f'Bearer {self.token}',

'X-Tenant': self.tenant,

'User-Agent': user\_agent,

'Content-Type': 'application/json'

})

def request(self, method: str, path: str, \*, params=None, json\_body=None, idempotency=False):

url = f'{self.base}{path}'

headers = {'X-Request-Id': str(uuid.uuid4())}

if idempotency:

headers['Idempotency-Key'] = str(uuid.uuid4())

attempt = 0

while True:

res = self.session.request(method, url, params=params, json=json\_body, headers=headers, timeout=self.timeout)

content\_type = res.headers.get('Content-Type','')

body = res.json() if 'json' in content\_type and res.text else {'raw': res.text}

if res.ok:

return body, res.headers

retryable = res.status\_code in (429, 503, 504) or (body.get('error',{}).get('retryable'))

if retryable and attempt < 5:

attempt += 1

backoff = min(1.0 \* 2\*\*attempt, 8.0) + random.uniform(0,0.25)

time.sleep(backoff); continue

err = body.get('error', {})

raise AimosError(

err.get('message', f'HTTP {res.status\_code}'),

err.get('code', 'HTTP\_ERROR'),

res.status\_code,

err.get('correlation\_id', headers['X-Request-Id']),

err.get('details')

)

def get(self, p, \*\*q): return self.request('GET', p, params={k:v for k,v in q.items() if v is not None})

def post(self, p, body=None, idempotency=False): return self.request('POST', p, json\_body=body, idempotency=idempotency)

def patch(self, p, body=None, etag=None):

h = self.session.headers.copy()

if etag: self.session.headers['If-Match'] = etag

try:

return self.request('PATCH', p, json\_body=body)

finally:

self.session.headers = h

**29.2.3 Models & iterators**

# aimos\_sdk/types.py

from pydantic import BaseModel

from typing import List, Optional, Dict

class VIF(BaseModel):

model\_id: str

weights\_hash: str

prompt\_template\_id: Optional[str] = None

tool\_ids: Optional[List[str]] = None

writer: str

class Atom(BaseModel):

id: str

modality: str

content\_ref: Dict[str, str]

embedding: Optional[Dict] = None

tags: Optional[List[Dict]] = None

hhni: Optional[Dict] = None

tpv: Optional[Dict] = None

snapshot\_id: str

created\_at: str

vif: VIF

# aimos\_sdk/memory.py

from .http import AimosHTTP

from .types import Atom

from typing import Iterator

class MemoryClient:

def \_\_init\_\_(self, http: AimosHTTP): self.http = http

def create\_atom(self, payload: dict):

data, headers = self.http.post('/atoms', payload, idempotency=True)

return Atom(\*\*data), headers.get('X-Snapshot-Id'), headers.get('ETag')

def list\_atoms(self, \*, tag=None, modality=None, limit=200, as\_of=None) -> Iterator[Atom]:

cursor = None

while True:

params = { 'tag':tag, 'modality':modality, 'limit':limit, 'cursor':cursor, 'as\_of':as\_of }

data, \_ = self.http.get('/atoms', \*\*params)

for item in data.get('items', []):

yield Atom(\*\*item)

cursor = data.get('next\_cursor')

if not cursor: break

def close\_snapshot(self, notes: str = None):

return self.http.post('/snapshots', { 'action':'close', 'notes': notes }, idempotency=True)[0]

# aimos\_sdk/orch.py

class OrchestrationClient:

def \_\_init\_\_(self, http: AimosHTTP): self.http = http

def start\_run(self, body: dict):

return self.http.post('/acl/runs', body, idempotency=True)[0]

def get\_run(self, run\_id: str):

return self.http.get(f'/acl/runs/{run\_id}')[0]

def get\_trace(self, run\_id: str):

return self.http.get(f'/acl/runs/{run\_id}/trace')[0]

def tail(self, run\_id: str, interval\_sec=1.2):

import time

while True:

frame = self.get\_run(run\_id)

yield frame

if frame['status'] in ['succeeded','failed','abstained','quarantined','rolled\_back','cancelled']:

break

time.sleep(interval\_sec)

**29.2.4 End-to-end example (Python)**

from aimos\_sdk.http import AimosHTTP

from aimos\_sdk.memory import MemoryClient

from aimos\_sdk.orch import OrchestrationClient

http = AimosHTTP()

mem = MemoryClient(http)

orch = OrchestrationClient(http)

# 1) Create atom

payload = {

"modality":"prompt:user",

"content\_ref": { "inline":"Summarize failed transfers by tip type last week.", "media\_type":"text/plain" },

"embedding": { "model\_id":"embed-004", "dim":768, "vector":[0.0]\*768 },

"tags":[{"key":"topic","value":"payments","weight":0.9}],

"hhni": { "path":["system:aiops","section:queries","sentence:12"] },

"tpv": { "priority":0.8, "relevance":0.87, "decay\_tau":604800 }

}

atom, snap, \_ = mem.create\_atom(payload)

print('atom', atom.id, 'snap', snap)

# 2) Close snapshot

mem.close\_snapshot('ingest batch 42')

# 3) Start pipeline run

run = orch.start\_run({

"pipeline\_id":"pipe:payments\_weekly\_report",

"inputs":{"lookback\_days":7},

"snapshot\_id": snap

})

run\_id = run['run\_id']

# 4) Tail

for frame in orch.tail(run\_id):

print('status', frame['status'], 'budget', frame.get('budget'))

# 5) Trace

trace = orch.get\_trace(run\_id)

print('steps', len(trace.get('steps', [])))

**29.3 CLI (Node/TS example)**

**29.3.1 Install & entrypoint**

npm i -g @aimos/cli

aimos --help

**29.3.2 Commands**

* aimos atoms:create --modality prompt:user --text "..." --tag topic:payments
* aimos snapshots:close --notes "ingest batch 42"
* aimos runs:start --pipeline pipe:payments\_weekly\_report --inputs lookback\_days=7 --snapshot S\_ABC123
* aimos runs:tail RUN\_ID
* aimos audit:lineage --claim seg:claim:release\_rc\_0\_3 --from 2025-10-10 --to 2025-10-17

**29.3.3 Implementation sketch**

#!/usr/bin/env node

import { Command } from 'commander';

import { AimosHTTP } from '@aimos/sdk/http';

import { MemoryClient } from '@aimos/sdk/memory';

import { OrchestrationClient } from '@aimos/sdk/orch';

const program = new Command();

program.name('aimos').description('AIMOS/APOE CLI').version('1.0.0');

program.command('atoms:create')

.requiredOption('--modality <m>')

.option('--text <t>')

.option('--tag <k:v>')

.action(async (opts) => {

const http = new AimosHTTP(); const mem = new MemoryClient(http);

const tags = opts.tag ? [{ key: opts.tag.split(':')[0], value: opts.tag.split(':')[1], weight: 0.9 }] : [];

const { atom, snapshotId } = await mem.createAtom({

modality: opts.modality,

content\_ref: { inline: opts.text, media\_type: 'text/plain' },

tags

} as any);

console.log(JSON.stringify({ id: atom.id, snapshotId }, null, 2));

});

program.command('snapshots:close')

.option('--notes <n>')

.action(async (o) => {

const http = new AimosHTTP(); const mem = new MemoryClient(http);

const res = await mem.closeSnapshot(o.notes);

console.log(JSON.stringify(res, null, 2));

});

program.command('runs:start')

.requiredOption('--pipeline <id>')

.requiredOption('--snapshot <sid>')

.option('--inputs <k=v...>')

.action(async (o) => {

const http = new AimosHTTP(); const orch = new OrchestrationClient(http);

const inputs = Object.fromEntries(String(o.inputs || '').split(',').filter(Boolean).map(pair => pair.split('=')));

const run = await orch.startRun({ pipeline\_id: o.pipeline, inputs, snapshot\_id: o.snapshot });

console.log(JSON.stringify(run, null, 2));

});

program.command('runs:tail')

.argument('<runId>')

.action(async (runId) => {

const orch = new OrchestrationClient(new AimosHTTP());

for await (const f of orch.tail(runId)) console.log(JSON.stringify(f, null, 2));

});

program.parse();

**29.4 Testing, mocks, and resilience**

* **JS/TS**: use nock to mock endpoints; assert retry behavior on 429 bursts; verify Idempotency-Key present for POSTs.
* **Python**: use responses or pytest-httpserver; simulate 503→503→200 sequences; assert backoff timing within tolerance.
* **Determinism**: seed any randomness (e.g., UUID strategy for e2e tests use fixed keys) when validating idempotency caching.
* **Budget headers**: capture and expose X-Budget-Tokens-Used, etc., for observability tests.

**29.5 Security & compliance hooks**

* **Token providers.** Allow injection (rotating tokens, STS). Never log tokens; scrub on exceptions.
* **PII redaction.** Pre-flight local redaction before POST /atoms; respect server FAILED\_PRECONDITION policy\_violation=true.
* **mTLS.** Optional constructor override for HTTP layer with client certs.
* **Provenance.** Surface returned vif on all write responses; provide helpers to attach VIF to downstream artifacts.

**29.6 Streaming & large artifacts**

* **Witness “tail”.** Default pattern uses polling (see 29.1.5/29.2.4). If SSE/WebSocket is enabled by the server, expose runs.subscribe(runId, onEvent) helper that falls back to polling.
* **Chunked uploads.** For large content\_ref.uri artifacts, prefer object storage + content\_ref.uri rather than inlining.

**29.7 Cookbook (common tasks)**

* **Vector search within a modality (TS):**

await memory.vectorQuery({

model\_id: 'embed-004', k: 25,

vector: myVec, filters: { modality: 'code:ts', 'tag.repo': 'checkout' }

});

* **Time-slice read (Py):**

latest = http.get('/atoms', as\_of='2025-10-17T00:00:00Z')[0]

* **Idempotent batch ingest (TS):**

await Promise.all(payloads.map(p => memory.createAtom(p)));

await memory.closeSnapshot('batch 17');

* **Lineage export for a claim (Py):**

bundle, \_ = http.get('/audit/seg/lineage', claim\_id='seg:claim:release\_rc\_0\_3',

\_from='2025-10-10T00:00:00Z', to='2025-10-17T00:00:00Z')

**29.8 Versioning & compatibility**

* **Semver packages** (@aimos/sdk, aimos-sdk).
* **API drift management.** SDKs carry generated types from the published OpenAPI (see Ch.28.10). Minor server changes are additive; breaking changes bump v2 base URL and @aimos/sdk@2.x.
* **Feature flags.** Expose booleans for optional endpoints (e.g., SSE witness) with graceful degradation.

**29.9 Client checklists**

**Initialization**

* AIMOS\_BASE\_URL, AIMOS\_TENANT, AIMOS\_TOKEN configured.
* Per-service timeouts set (connect/read).
* User agent string identifies your app.

**Reliability**

* Retries on 429/503/504 + retryable=true.
* Idempotency keys on POST.
* If-Match on PATCH (when used), handle 412.

**Security**

* No secrets in logs.
* PII pre-redaction before ingest.
* Token rotation hooks present.

**Observability**

* Capture X-Budget-\* headers.
* Log correlation\_id on errors.

**Part XII — Mathematics & Formal Methods**

**Chapter 30. Retrieval Math — RS/QS/IDS/DD definitions, normalization, calibration, and UQ theory**

Aim. Specify the retrieval scoring family used by AIMOS/APOE; prove its monotonicity/bounds; define normalization and calibration to map scores to probabilities; and formalize uncertainty quantification (UQ) for gating (κ) and orchestration budgets.

**30.0 Notation (recap)**

* indexes a candidate memory atom/molecule; is the query/task context.
* : quality score for relative to .
* : indexing-depth score for in HHNI.
* : dependency-delta risk incurred by using .
* : retrieval score for .
* : calibrated probability .
* : abstention threshold; abstain when .
* : time constant for decay; are positive weights.
* : logistic; : min/max clamp.

**30.1 RS/QS/IDS/DD — definitions & bounds**

**30.1.1 Quality score**

Decomposed into **completeness**, **specificity**, and **semantic alignment**, with temporal hygiene:

* : fraction of query intents covered by (from intent graph).
* : inverse breadth (e.g., ).
* : normalized similarity (e.g., cosine of query/atom embeddings, reranked by cross-encoder if available).
* , . Clip to .

*Monotonicity*: increases with , , ; non-increasing in .

**30.1.2 Indexing-depth score**

Measures **how deeply and cleanly** the content is positioned in HHNI:

* : number of resolvable sub-indices (system→…→sub-word) beneath/around .
* : local connection density quality (weighted by tag precision, not raw degree).
* : share of the relevant section hierarchy with valid links.
* : inconsistency penalty (dangling hashes, orphan edges).
* . Output is in via .

*Intuition*: Deep, well-connected, and consistent items score higher.

**30.1.3 Dependency-delta**

Risk that using **breaks or contradicts** existing commitments:

* : neighboring edges/nodes (dependencies) in SEG/HHNI.
* : change in key tag vectors vs. last accepted snapshot.
* Hard contradictions raise sharply (binary bump).
* is perfectly safe; is unusable without quarantine/HITL.

*Monotonicity*: increases with conflicts and drift.

**30.1.4 Retrieval score**

**Compositional** and **bounded**:

* **Bounds**: product of three terms ⇒ .
* **Sensitivity**: ;  
  ;  
  .
* **Interpretation**: high quality + deep index **unless** the dependency risk is high.

**30.2 Normalization & calibration**

We must convert raw scores to **well-calibrated probabilities** usable for κ-gating and budgets.

**30.2.1 Score normalization**

Bring heterogenous components to a comparable scale before product:

* **Z-score within slice** (modality/tenant/time):  
  .
* **Quantile map** to : .
* **Affine clip** for stability: .

We normalize into before composing .

**30.2.2 Probability calibration**

Map (or a feature vector) to with held-out labels .

**a) Platt / Logistic calibration (binary):**

Fit on validation data minimizing log-loss; optionally L2-penalize.

**b) Isotonic regression (non-parametric):**  
Monotone stepwise function s.t. . Better when score–probability relation is non-linear.

**c) Multi-feature calibration:**

Use **slice-aware** models (per modality/domain) to avoid leakage and improve fairness.

**d) Temperature scaling (for classifier logits):**  
For cross-encoder logits : , fitted on a dev set.

**Calibration metrics:** Expected Calibration Error (ECE), Brier score, reliability curves.  
**Acceptance**: ECE ≤ 0.04 on primary slices; Brier ≤ baseline-5%.

**30.3 Uncertainty theory (UQ) for retrieval & chains**

We quantify **aleatoric** (inherent ambiguity) and **epistemic** (model ignorance).

**30.3.1 Pointwise uncertainty**

* **Predictive entropy**: .
* **Ensemble variance**: multiple models or seeds ,  
  .
* **Temperature sweep stability**: stddev across .

Combine into a **UQ vector** and a scalar risk index:

**30.3.2 Conformal coverage (optional guarantees)**

Split-conformal for retrieval thresholds:

* Define nonconformity .
* On calibration set , pick quantile of .
* At inference, accept if ⇒ marginal coverage .

This yields **finite-sample** control of false accepts under exchangeability.

**30.3.3 Chain-level UQ (APOE)**

For a chain using items with pairwise dependence small:

* **Law of total variance** aggregation (approximate):
* **Min-confidence heuristic** for safety gates:

Gate if or .

**30.4 Algorithms (reference)**

**30.4.1 Compute for a candidate**

1. **Features**: .
2. **Normalize** features to .
3. **Compose** via §30.1.
4. **Product** .
5. **Clip** to .

*Complexity*: per candidate (features precomputed in ingestion).

**30.4.2 Calibrate to**

* Maintain per-slice calibrators (Platt or isotonic).
* Periodically refit on labeled interactions (click/accept, HITL outcomes).
* Emit and UQ vector .

**30.4.3 κ-gating & selection**

* If ⇒ abstain/seek clarification.
* Else select top- by subject to diversity & policy filters; penalize high .

**30.4.4 Online adaptation (drift)**

* Track slice ECE; if > threshold for windows ⇒ refit calibrators.
* Update by bandit on downstream success (A/B across traffic shards).

**30.5 Acceptance criteria & dashboards**

**Correctness**

* AUC/PR on relevance ≥ baseline +3–5%.
* RS-lift in offline recall@k ≥ +8% on HHNI-deep queries.

**Calibration**

* Global ECE ≤ 0.04; worst-slice ECE ≤ 0.06.
* Brier ≤ baseline-5%.

**Safety**

* κ-abstentions: ≥ 70% precision when abstaining (i.e., abstentions correlate with true ambiguity).
* Conflict usage: fraction of selected items with ≤ 2%.

**Latency/Cost**

* RS+calibration overhead ≤ 5 ms @ p95 (pre-computed features).
* Top-k rerank with cross-encoder ≤ 60 ms @ p95 (optional).

**Observability (to SEG/VIF)**

* Log with model/calibrator ids, data slice, and snapshot id.
* Reliability curves per slice; κ threshold overlays; drift alerts.

**30.6 Implementation notes (defaults)**

* **Weights**: (adjust by domain).
* **IDS**: ; clip to .
* **DD**: ; hard contradictions set .
* **κ**: start 0.65; tune to keep abstention rate 5–15% with high utility.
* **Calibration**: isotonic where enough data ( labels/slice), else Platt; re-fit weekly or on ECE breach.
* **Ensembles**: 3 seeds or 2 model families for variance when available.

**30.7 Worked micro-example**

Suppose for candidate : .

* .  
  Time decay ⇒ .
* with ⇒ .
* (minor drift, no conflict).
* .
* Calibrator (Platt): .
* Entropy bits; ensemble var .
* If , accept; else abstain/seek more context.

**30.8 Interface contracts (summary)**

* **Write path** (Ch.28): /atoms must store components and the **calibration model id** used at read time.
* **Read path**: /query/vector returns per hit; SEG/VIF embeds the parameters.

**Outcome.** This chapter fixes the retrieval scoring semantics (RS), ties them to HHNI structure (IDS) and safety (DD), and delivers a **probabilistic, calibrated interface** with auditable UQ for κ-gating—foundational for APOE routing and for satisfying the Witness/Audit invariants.

**Part XII — Mathematics & Formal Methods**

**Chapter 31. Dynamics — GODN potentials & forces, convergence sketches, stability regions**

**Scope.** This chapter specifies the continuous and discrete-time dynamics used by the Graviton Organic Dynamics Network (GODN): the potential energy model, force laws, integrators, convergence conditions, stability regions, and practical parameter regimes. These definitions are aligned with the GODN reference you provided and are specialized here for AIMOS/APOE (DVNS/HHNI) retrieval and orchestration.

Graviton Organic Dynamic Network

**31.1 State, potentials, and units**

**State variables (per node *i*):** position , velocity , mass . Distances , unit vector .

**Total energy.**

with

Parameters: gravitational constant , barrier stiffness , bond stiffness , perimeter distance , bond equilibrium , bond set . (Matches the reference’s gravity, elastic perimeter, holding terms.)

Graviton Organic Dynamic Network

**Damping (Rayleigh).** Linear velocity damping with coefficient .

**Non-dimensionalization (for tuning):** choose a length scale , time scale , mass scale . Define , , . Natural dimensionless groups:

* **Gravity–elastic ratio** (collapse tendency).
* **Bond–barrier stiffness ratio** (cluster rigidity).
* **Damping ratio** (effective local second-order behavior).

**31.2 Forces and equations of motion**

For each pair :

Net force: . Dynamics: . (As summarized in the GODN framework.)

Graviton Organic Dynamic Network

**31.3 Integrators (energy behavior & recommendations)**

**Symplectic Euler (semi-implicit).**

Pros: simple, stable for stiff contact-like terms, modest energy drift. Good default for online DVNS refinement.

**Velocity-Verlet.**

Pros: second-order, near-symplectic; use when energy fidelity matters (e.g., precomputation/offline).

**Adaptive time-step.**  
If small (near equilibrium), increase step (fast-forward); near high-curvature events (close approaches, bond snaps), decrease step. (Matches GODN’s time scaling guidance.)

Graviton Organic Dynamic Network

**Guideline:** begin with symplectic Euler, clamp by local Lipschitz estimate of forces; switch to Velocity-Verlet in offline precompute of super-index fields.

**31.4 Convergence sketches (with damping and gates)**

**Energy descent with damping.** For fixed and , continuous-time dynamics satisfy

Thus is a Lyapunov function and decreases monotonically to a (possibly local) minimum. In discrete time, if is small enough (CFL-like bound below), the numerical scheme preserves practical descent (non-increasing except for jitter). (Consistent with the doc’s energy-minimization loop.)

Graviton Organic Dynamic Network

**Stopping conditions (APOE gate compatible).**

* for consecutive steps, **and**
* max speed , **and**
* topology stable (no bond/perimeter events over steps).

**Abstention trigger (κ):** if energy fails to descend or oscillates beyond iterations, emit and route to a safer parameter regime (reduce , increase , or raise ).

**31.5 Stability regions & parameter regimes**

We characterize qualitative regimes by three axes:

1. **Collapse vs. dispersion (gravity vs. barrier).**  
   Define .

* **Dispersion region:** → repulsion dominates, nodes maintain spacing; good for **policy-aware retrieval** (no cluster collapse).
* **Balanced clustering:** – → soft clusters, emergent paths without singular collapse (DVNS sweet spot).
* **Collapse-prone:** → dense cores; only safe with strong and nonzero bonds.

1. **Rigidity vs. fluidity (bond vs. barrier).**  
   .

* Low : fluid clusters, easier path re-routing, risk of jitter.
* Mid (0.5–5): elastic but stable clusters; recommended.
* High : rigid lattices; can trap in suboptimal shapes.

1. **Damping ratio (oscillation control).**  
   Local second-order modes exhibit under/critical/over-damped behavior governed by .

* : faster convergence but overshoot/oscillation risk.
* : robust, monotone settling; recommended for online.
* : slow but safest near constraints.

**Discrete-time CFL-like bound (rule of thumb).**

Pick tighter step if close approaches are frequent (TSP cluster stitching, protein contacts). These prescriptions follow the reference’s “stability and scalability” and “dynamic parameter” guidance.

Graviton Organic Dynamic Network

**31.6 Fast force evaluation (scaling)**

Naïve pairwise forces are . Use:

* **Cutoffs & neighbor lists** for repulsion/holding (short-range).
* **Barnes–Hut / FMM** for gravity (approximate far-field, ).
* **Uniform grids** in 2D/3D TSP-like tasks for near-field search.

These preserve the emergent behavior while scaling to large HHNI/DVNS corpora. (The GODN doc emphasizes scalability via localized computations.)

Graviton Organic Dynamic Network

**31.7 Mapping to DVNS/HHNI retrieval**

**Positions from embeddings.** Let be normalized embedding vectors. Use angular distance (or ). Forces then act along (the geodesic gradient direction). Practical simplification: embed to , normalize, treat distances as Euclidean locally.

**Policy-aware geometry.** Mark forbidden regions (privacy, license) as **hard perimeters**: set a high and small buffer around restricted sets; DVNS will not cross them (forces push trajectories away).

**Adaptive masses.** Let (from CMC/HHNI). Use decay for stale nodes (the doc’s “mass adjustment”).

Graviton Organic Dynamic Network

**31.8 Default parameter packs (online vs. offline)**

| **Mode** |  |  |  |  |  |  |  | **Notes** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Online refine** | mid | high | mid | small | medium | high | small | Stable, monotone; policy-safe |
| **Exploration** | mid-high | mid | low | small | large | mid | small-mid | Faster clustering, may oscillate |
| **Precompute** | mid | high | mid | small | medium | mid-high | adaptive | Use Velocity-Verlet, adaptive dt |

Tighten and when running with strong gravity or noisy embeddings.

**31.9 Failure modes & mitigations**

* **Collapse (core singularity):** too large or too small → raise , increase , clamp min .
* **Oscillation/limit cycles:** with large → increase , reduce , switch to symplectic Euler.
* **Fragmentation (no paths):** barriers too strong or too weak → lower or raise /.
* **Stall (no convergence):** poor step control near contacts → adaptive , neighbor cutoffs, reorder force eval.
* **Policy breach risk:** ensure forbidden regions have **hard perimeters** and that APOE gates check crossings before commit.

**31.10 Algorithm sketch (discrete iteration)**

given nodes {x\_i, v\_i, m\_i}, parameters (G, k\_b, k\_h, d\_p, d\_c, c\_d), bonds B

repeat

// neighbor discovery (short-range) and far-field approx (gravity)

for i: F\_i = -c\_d \* v\_i

for pairs (i,j) in near-list:

F\_i += F\_grav(i,j) + F\_rep(i,j) + F\_hold(i,j)

F\_j -= (same)

for far clusters: add Barnes–Hut gravity contributions

// integrate (symplectic Euler)

v\_i += (F\_i / m\_i) \* Δt

x\_i += v\_i \* Δt

// adaptive dt & event handling

if (close-approach or bond event) reduce Δt; else maybe increase

// check convergence gates (ΔE, max|v|, topology stable)

until gates pass or κ-abstain

**31.11 Worked intuitions (three canonical tasks)**

* **TSP-like retrieval:** gravity pulls semantically close cities into clusters; barrier keeps tour from self-intersecting; bonds stitch cluster frontiers; damping yields a clean loop. (Doc’s TSP example.)

Graviton Organic Dynamic Network

* **Protein-like folding:** hydrophobic cores via gravity, steric clash avoided by perimeters, secondary-structure bonds as springs; adaptive mass highlights catalytic residues.

Graviton Organic Dynamic Network

* **Galaxy-like super-indexing:** super-dense knowledge regions act as high-mass attractors; arms form along semantic manifolds; dynamic focus zooms dense cores.

Graviton Organic Dynamic Network

**31.12 What to log (for VIF/SEG)**

Per run: , integrator, neighbor/cutoff scheme, seeds, energy trace , gate outcomes, κ events, and the mapping from nodes to artifacts (snapshot ids). This makes each emergent configuration **witnessable** and **replayable** inside SEG.

**Takeaway.** GODN’s simple potential—gravity + elastic perimeter + holding—combined with damping and careful integration yields a robust, tunable engine for emergent organization. With the stability map , adaptive time/mass, and policy-aware perimeters, AIMOS/APOE can run DVNS at scale while remaining verifiable and safe.

Graviton Organic Dynamic Network

**Part XII — Mathematics & Formal Methods**

**Chapter 31. Dynamics (Focused Addendum)**

Below is a deep, self-contained expansion of the three items you asked for: **31.2 Forces**, **31.3 Convergence Proof Sketches**, and **31.4 Stability Regions**. It refines and extends the earlier high-level write-up with tighter definitions, optional smoothings, and explicit conditions you can implement and test.

**31.2 Forces — full specification & implementation notes**

**31.2.1 State and notation**

* Nodes with position , velocity , mass .
* Pair geometry: , (for ).
* Parameter set :
  + : gravity strength.
  + : barrier/perimeter stiffness (short-range repulsion).
  + : bond stiffness for pairs in (the “holding set”).
  + : perimeter distance (minimum comfortable separation).
  + : bond equilibrium distance.
  + : linear (Rayleigh) damping.

**31.2.2 Potentials and canonical forces (piecewise)**

Total energy

with components

Conservative pair forces are (equal and opposite on ):

Damping: .  
Net force: .

**Equations of motion (continuous time).**

**31.2.3 Smooth, differentiable alternatives (for robust autodiff)**

The piecewise barrier can introduce kinks at . For differentiable pipelines (e.g., learning or embedding–force co-training), prefer a smooth approximation:

* **Soft barrier (softplus-squared):**  
  Let . Define .  
  Then with .
* **Softened gravity (Plummer-like):**  
  Replace with (small ), limiting forces at very small .
* **Bond windowing:**  
  Multiply the bond term by a smooth gate to avoid stiff snaps when bonds appear/disappear.

These smoothings preserve intent (attraction/repulsion/holding) while improving gradient stability.

**31.2.4 Embedding-space forces for DVNS/HHNI**

In DVNS retrieval, positions come from normalized embeddings . Two pragmatic constructions:

1. **Euclidean proxy.** Work in with normalized, distance . Forces as above with defined in the ambient space.
2. **Angular metric.** Use as “distance”. For a potential , the force in the tangent space is:

with . In practice, we often adopt the Euclidean proxy for speed and simplicity, reserving angular corrections for offline precomputation.

**31.3 Convergence proof sketches (continuous & discrete; with events)**

**31.3.1 Continuous-time gradient–damped system**

Define the conservative force . With damping (), the dynamics are

**Lyapunov descent.** Consider . Then

Hence is non-increasing, bounded below (by construction), and thus converges. By LaSalle’s invariance principle, the trajectories approach the largest invariant set where and , i.e., a (possibly local) equilibrium of .

**With smooth barriers/bonds.** If the potentials are (soft variants above), the same Lyapunov argument holds globally.

**With piecewise barriers/bonds.** The system is a **hybrid** (switched) dynamical system with discrete “events” (enter/exit perimeter, bond on/off). Between events, the Lyapunov argument holds. At an event, energy may jump **up** or **down** depending on how the event function is implemented:

* If bonds are activated only when it *reduces* energy (or under a “no-increase” policy), remains non-increasing.
* If bonds/perimeters enforce **projections** that do not increase (e.g., snapping to the nearest feasible manifold), remains non-increasing.

**Conclusion (sketch).** With and a “no-increase” event policy, the continuous system converges to a closed set of equilibria; generically, to a local minimum (modulo plateaus/saddles).

**31.3.2 Discrete-time integration (semi-implicit Euler / Velocity-Verlet)**

Let be the step.

**Semi-implicit (symplectic) Euler**

For a quadratic well, this scheme is energy-stable for sufficiently small and . In our mixed potentials, **practical descent** holds if:

* satisfies a CFL-like bound w.r.t. local stiffness (below),
* damping is non-zero,
* events are handled by “no-increase” policies.

**Velocity-Verlet** is second-order and near-symplectic. With linear damping (handled via splitting or exponential integrators), it exhibits smaller long-term drift. Use it in offline precompute or when energy fidelity matters.

**Step-size bound (rule-of-thumb).** Let . A stable satisfies

where is a characteristic length and a characteristic mass. In practice, we **adapt** downwards when (i) any is near a barrier or (ii) a bond event is imminent.

**Stopping gates (deterministic APOE-compatible).** Stop when

If not satisfied by iterations, **abstain** () and re-run with safer parameters (↓, ↑, ↑).

**31.3.3 Local optimality and saddles**

At a fixed event pattern (no switching), equilibria satisfy . The Hessian determines local type:

* **Positive-definite :** strict local minimum (preferred).
* **Indefinite :** saddle; small perturbations or stochasticity help escape.
* **Near-singular :** flat directions; increase or slightly to regularize.

With damping, noisy embeddings, or small stochastic kicks, systems typically leave saddles and settle into local minima.

**31.4 Stability regions — maps, regimes, and tuning playbook**

**31.4.1 Dimensionless groups (for portable tuning)**

Choose scales (length), (mass). Define:

* **Gravity–barrier ratio**

Small → barrier dominates (spread out); large → collapse risk.

* **Bond–barrier stiffness ratio**

Low → fluid clusters (easy re-route), high → rigid clusters (may trap).

* **Damping ratio** (local second-order surrogate)

with from local curvature (barrier/bond) and a typical mass.

**31.4.2 Qualitative phase map**

* **Dispersion region** ():  
  Repulsion dominates; nodes maintain spacing; good for **policy-aware retrieval** and safe navigation (no collapses), but can fragment (add mild or bonds to “thread” paths).
* **Balanced clustering** ():  
  Soft clusters and emergent “lanes”. **DVNS sweet spot** for forming stable retrieval paths without singular cores.
* **Collapse-prone** ():  
  Strong attraction; requires higher , hard perimeters, and bond guidance. Use only for “galaxy” super-indexing or precompute, not online navigation.

Stiffness ratio :

* : elastic and steerable clusters (recommended).
* : jitter; increase damping or barrier.
* : rigid lattices; risk of suboptimal traps.

Damping :

* : monotone settling (recommended online).
* : faster but oscillatory; combine with adaptive .
* : very safe, may be slow; good for sensitive policy zones.

**31.4.3 Numerical stability (time-step and events)**

* Use **adaptive** with multiplicative factors (e.g., ×0.5 on close approach; ×1.1 when calm).
* Enforce **minimum separation** (tiny ) or softened gravity to avoid blow-ups.
* Handle **events** (perimeter/bond transitions) with a **no-increase** energy policy; if violated, roll back one substep and re-attempt with smaller or softened parameter locally.

**31.4.4 Default parameter packs (online vs. offline)**

| **Mode** |  |  |  |  | **Use** |
| --- | --- | --- | --- | --- | --- |
| **Online navigation** | – | 1–3 | ~1 | small, adaptive | Safe DVNS refinement on top-k |
| **Exploratory clustering** | – | 0.5–2 | 0.6–0.9 | small–mid | Rapid structure discovery; watch oscillations |
| **Offline precompute** | – | 1–5 | 0.8–1.2 | adaptive (Verlet) | Super-index field building |

**31.4.5 Tuning playbook (actionable)**

1. **You see collapse (cores):** ↑, ↑, ↓; clamp ; soften gravity near .
2. **You see oscillations:** ↑; ↓; if needed, raise modestly.
3. **You see fragmentation:** ↑ slightly or add/strengthen bonds ().
4. **Slow convergence:** mild ↑ in calm regions; switch to Velocity-Verlet offline.
5. **Policy breach risk:** define forbidden sets as **hard perimeters** (large , small ), and add pre-step APOE gates to reject crossings.

**Minimal checklist for implementation**

* **Forces**: implement conservative forces + damping; optional smooth variants.
* **Integrator**: start with semi-implicit Euler + adaptive ; consider Velocity-Verlet offline.
* **Gates**: energy descent, velocity cap, topology stability; κ-abstain on failure.
* **Params**: target for online DVNS.
* **Events**: “no-increase” policy; rollback on violation.
* **Safety**: hard perimeters for policy zones; minimum separation; softened gravity.

This addendum gives you the mathematically disciplined backbone you need to run GODN inside DVNS/HHNI confidently—stable, tunable, and auditable.

**Part XII — Mathematics & Formal Methods**

**Chapter 31. Dynamics**

**31.1 GODN Potentials & Forces (recap + formalization)**

We model each node (atom, tag, step, document, etc.) as a particle with mass , position , and velocity . Pairs interact through **attractive** (semantic “gravity”), **repulsive/elastic** (collision & structure), and **holding** (soft constraints) terms; plus **viscous damping** for dissipation.

Let , , with a small to avoid division by zero.

**Potential energy (pairwise + unary):**

* : set of **structural** edges we want to keep near rest-length (e.g., hierarchy or typed dependency).
* : set of **forbidden** sets (guardrails), enforced by convex barriers (e.g., large penalty near disallowed subspaces).
* Softened gravity prevents singular collapse; hard-core repulsion prevents overlap; springs preserve HHNI/SEG geometry.

**Forces (negative gradient):**

with the viscous damping. Expanded pairwise terms (omitting indicators and barriers for brevity):

**Continuous-time dynamics (second order with damping):**

**Discrete-time integrators** (used by DVNS/online re-ranking):

* **Symplectic/Velocity-Verlet (recommended):** stable for stiff springs if respects CFL bounds (below).
* **Explicit Euler:** simple but small region; we use it only for analysis/fallback.
* **Projected step:** after each update, project onto feasible set to respect policy geometry .

**31.2 Convergence — proof sketches**

We establish that (a) **energy decays** under damping, (b) the flow **converges to critical points** of , and (c) with mild curvature conditions those points are **local minima** (no oscillatory failure). We then lift to the discrete setting with step-size bounds.

**Lemma 31.2.1 (Energy decay, continuous time)**

Define total energy . Then

*Sketch.* Differentiate , use and the equation of motion to cancel conservative work; damping contributes the negative term.

**Consequence.** is nonincreasing and bounded below ( is bounded below by design via softening/barriers) ⇒ converges and ⇒ along some subsequence.

**Lemma 31.2.2 (Limit set is critical)**

Every -limit point of satisfies .  
*Sketch.* If , then for small neighborhood the system experiences nonzero conservative force; with , energy would strictly decrease further, contradicting convergence of .

**Proposition 31.2.3 (Convergence to local minima under PL/gradient–dominance)**

Assume is -smooth (Lipschitz gradient) and satisfies a **Polyak–Łojasiewicz (PL)** inequality in a neighborhood of the set of minimizers :

Then the damped dynamics converge to and at least linearly in energy gap.  
*Sketch.* Standard heavy-ball/gradient flow arguments: PL yields energy–gradient coupling; damping kills kinetic energy; LaSalle’s invariance principle finishes.

**Design note.** Our potentials are sums of **attractive kernels with softening** plus **convex repulsive/constraint** terms. In practice this yields a landscape that is (piecewise) **gradient-dominated** around desired configurations (clusters at rest-lengths, no overlaps), satisfying the PL condition locally.

**Discrete-time convergence (Velocity-Verlet with damping)**

Consider Velocity-Verlet with step :

**Theorem 31.2.4 (Energy stability region, linearized)**  
Linearize around a strict local minimum , where , and set , . For the decoupled mode with frequency , Velocity-Verlet is **stable** if

Moreover, with damping, the **discrete energy** decays for sufficiently small (proof via modified energy / backward error analysis).  
*Sketch.* Reduce to scalar damped oscillator; use known VV stability chart. The global condition uses and .

**Corollary 31.2.5 (CFL-like step bound, multi-dimensional)**  
Pick

to ensure practical decay and avoid ringing.

**Projected & constrained dynamics**

With barriers and post-step **projection** onto convex feasible (policy geometry), we obtain **Fejér monotonicity** to and retain energy decay up to projection residuals. Standard projected heavy-ball analysis yields convergence to a KKT point of under the same step bounds.

**31.3 Stability regions — parameter maps & operational rules**

We now chart parameter regimes that guarantee **no collapse**, **no oscillation**, and **fast settling**.

**31.3.1 Spectral stiffness & natural frequencies**

Let be the Hessian. Near any stable configuration ,

* **Large** (tight structure springs) and **large** (hard core) increase ⇒ **reduce allowable** .
* **Softening** reduces curvature of the gravity kernel at short distances ⇒ **increases** stability region.

**Rule R1 (CFL bound).** Measure/estimate (or upper bound via Gershgorin/diag dominance). Choose

**31.3.2 Damping regimes**

For a scalar damped mode , define .

* **Underdamped** : oscillatory; fast but may ring.
* **Critically damped** : fastest non-oscillatory approach.
* **Overdamped** : monotone but slower.

**Rule R2 (critical-ish damping).** Target per-mode . Operationally,

where is local effective stiffness seen by node (sum of adjacent spring curvatures + local Hessian contribution from gravity/repulsion).

**31.3.3 Anti-collapse & anti-crowding**

* **Softening ().** Use . Too small ⇒ singular pull; too large ⇒ weak clustering.
* **Repulsion radius ().** Set near expected minimum separation (e.g., HHNI layer spacing).
* **Gain ratio.** Keep **balanced** with and structural :

This ensures gravity cannot overpower short-range stabilization.

**Rule R3 (no-collapse margin).** Monitor minimum distance ; if , automatically increase or decrease by 10–20% and restart the current micro-iteration (abstain if repeats).

**31.3.4 Discrete stability charts (per integrator)**

* **Velocity-Verlet + damping:** stability if and . Region widens with higher damping up to critical.
* **Explicit Euler (position–velocity):** requires very small (not recommended for stiff ).
* **Projected VV:** similar to VV provided projection residuals are bounded; if projections are large, reduce .

**Rule R4 (adaptive step).** Use **energy trending** to adapt:

If (energy increased) or constraint residual grew, set ; if and residual low, set .

**31.3.5 Composite stability map (operational)**

1. **Estimate curvature.**
   * Quick bound: , with node degree.
2. **Pick .**
   * (start), clamp by .
3. **Set damping.**
   * (critical-ish).
4. **Anti-collapse.**
   * Choose per density; ensure gain ratio bound (above).
5. **Run with adaptive step + projection.**
   * Enforce barriers; track , , and projection norm.
6. **Abstain on instability.**
   * If any of: , for 3 consecutive steps, or projection residual > threshold, **abstain** and surface diagnostics (G/k/ suggestions).

**31.3.6 Mapping to AIMOS/APOE behavior**

* **DVNS retrieval.** Convergence ⇒ a stable **semantic path/cluster** on top-k candidates; no ringing ⇒ deterministic, low-variance re-ranking.
* **Policy-aware geometry.** Projection ensures **forbidden crossings** (guardrails) are respected, turning instability into **abstention** rather than silent failure.
* **Budget coupling.** Adaptive step ties naturally to token/time budgets: when energy trends poorly, tighten step or early-stop with a VIF note.

**31.3.7 Practical defaults (good starting priors)**

* median inter-candidate distance (embedding space).
* median inter-candidate distance.
* , (structural), (rescale with embeddings).
* (heavier for high-priority atoms).
* .
* (estimated), adapt by Rule R4.

**31.3.8 What can go wrong (and how we prevent it)**

* **Mode coupling & chatter.** Too small or too large → oscillations. *Mitigation:* R2 + R4.
* **Cluster collapse.** Gravity dominates at short range. *Mitigation:* softening , enforce gain ratio, stronger repulsion.
* **Constraint thrashing.** Large projections each step. *Mitigation:* increase damping, reduce , or relax/rest-length .
* **Discrete drift.** Explicit Euler with stiff . *Mitigation:* switch to Velocity-Verlet.

**31.3.9 Guarantees we rely on (summary)**

* **Monotone energy** with damping (31.2.1) ⇒ **no hidden energy sources**.
* **Critical-point convergence** (31.2.2) ⇒ **fixed points are stationary**.
* **Local optimality** under PL/strict convexity in basin (31.2.3) ⇒ **we converge to useful minima**.
* **Discrete stability** (31.2.4–5) under **CFL-like** bounds ⇒ **implementable with predictable behavior**.

**Part XII — Mathematics & Formal Methods**

**Chapter 32. Type Systems & Logic — ACL types & effects, symbolic checkers, soundness notes**

Objective: make orchestration **legible, checkable, and safe** before it runs. ACL’s type-and-effect discipline guarantees that plans are well-formed, budget-bounded, policy-lawful, and witness-producing by construction.

**32.0 Overview**

* **What we type-check:** pipelines, steps, tools, models, gates, budgets, and evidence emissions.
* **What we prove:** (i) no missing inputs, (ii) no illegal capabilities, (iii) budgets won’t underflow, (iv) every boundary emits VIF + UQ, (v) optional information-flow safety, (vi) determinism flags when claimed.
* **How we do it:** a **type-and-effect** system with **budget indices** and **capability tokens**, plus symbolic constraints discharged by SMT/SHACL/Datalog checkers.

**32.1 ACL kinds & types**

**32.1.1 Kinds**

* \* — value types (data)
* cap — capability kinds (authority to act)
* ε — effect rows (latent side effects)
* idx — budget indices (compile-time naturals / intervals)

**32.1.2 Base value types (\*)**

* Text, Bytes, JSON, YAML, UUID, Time, URL, Path
* Score (0..1), Tokens (int), Seconds (float), Energy (float)
* Embedding[d] (fixed dimension d)
* Atom, Molecule, Edge, SnapshotId
* Claim, Evidence, Witness, UQVec (all carry VIF metadata)
* Graph (SEG slice), HL7, PDF, Parquet (domain wrappers)

**32.1.3 Parametric data types**

* List[T], Option[T], Map[K,V], Stream[T]
* Gate[T] (a check producing a Witness for T)
* Budget (phantom-typed: Budget[tokens:idx, secs:idx])
* Result[Ok, Err]

**32.1.4 Roles & tools**

* Role = Planner | Retriever | Reasoner | Verifier | Builder | Critic | Operator | Witnesser
* Model[id, temp:idx, top\_p:idx]
* Tool[name, caps:cap] (e.g., caps = {net[domain], fs[scope], seg[mode]})

**32.1.5 Capabilities (kind cap)**

* mem = read\_mem | write\_mem
* seg = seg\_read | seg\_write
* net[domain], fs[scope], exec[lang], llm[provider]
* **No ambient authority**: capabilities are **linear tokens** consumed or re-scoped at call sites.

**32.1.6 Effects (kind ε)**

Effect rows describe latent side effects a step may perform:

ε ::= {}

| ε ∪ { ReadMem | WriteMem | CallLLM | CallTool[t]

| Net[domain] | File[scope] | SEG[mode]

| Time | Random | Abstain[κ] | Emit[VIF] }

Row polymorphism allows steps to **add** effects; composition **joins** rows.

**32.1.7 Refinements (lightweight dependent predicates)**

Any base/parametric type may be refined with guards:

* Tokens{n ≥ 0}, Score{0 ≤ s ≤ 1}, Embedding[d]{norm≤1}
* Claim{has\_lineage}, Budget{tokens≤T ∧ secs≤S}

**32.2 ACL typing surfaces**

We write judgments as:

* : value environment (names → types)
* : refinement constraints (SMT-discharged)
* : capability context (linear multiset of tokens)
* : ACL term (pipeline, step, call…)
* : result type
* : latent effect row
* : **budget summary** (tokens/sec upper bounds)

Key constructs:

* **Step:** step name : (In) -> (Out) when Gate using Tool with Budget
* **Pipeline:** p1 |> p2 (seq), p1 || p2 (parallel), route on pred { … }
* **Gate:** assertions g\_spec, g\_tests, g\_trace, g\_safety, producing Witness
* **Emit:** emit\_vif(meta) obligation at boundaries

**32.3 Core typing & composition rules (selected)**

**(VAR)**

If , then .

**(CONST)**

Literals have ground types with empty effects and zero budget.

**(TOOL-CALL)**

Γ; Φ; Δ ⊢ tool : Tool[name, caps]

caps ⊇ needed\_caps(e) // capability check

B' = cost\_model(tool, e) // static upper bound

———————————————

Γ; Φ; Δ ⊢ call(tool,e) : Out ! {CallTool[name] ∪ eff(e)} ▷ B'

**(LLM-CALL)**

Requires declared Model and budget token bound:

Γ; Φ; Δ ⊢ m : Model[id, temp, top\_p]

Γ; Φ ⊢ T\_max : Tokens{T\_max ≥ prompt(e)+ctx}

B' = { tokens = T\_max, secs = SLA(id,T\_max) }

———————————————

Γ; Φ; Δ ⊢ call\_llm(m,e) : Text×UQVec×Witness ! {CallLLM, Emit[VIF]} ▷ B'

*Obligation:* emits VIF (model id, weights hash, prompt template id, snapshot id).

**(GATE)**

A gate checks a predicate, emits a Witness, no effects beyond Emit[VIF]:

Γ; Φ ⊢ pred : Bool

———————————————

Γ; Φ; Δ ⊢ gate(pred) : Witness ! {Emit[VIF]} ▷ 0

**(SEQ)**

Sequential composition accumulates effects and adds budgets **monoidally**:

Capabilities flow linearly: (no duplication).

**(PAR)**

Parallel branches sum budgets and **union** effects; capability tokens must be **split**:

Γ; Φ; Δ1 ⊢ p : T ! ε1 ▷ B1

Γ; Φ; Δ2 ⊢ q : U ! ε2 ▷ B2

Δ = Δ1 ⊎ Δ2 // disjoint linear split

———————————————————————————————

Γ; Φ; Δ ⊢ p || q : T×U ! (ε1 ∪ ε2) ▷ (B1 ⊕ B2)

**(ROUTE)**

All branches must **type-agree** and budgets are **weighted joins** by static branch coverage bounds or worst-case:

Γ; Φ ⊢ pred : Score

∀i. Γ; Φ; Δ ⊢ arm\_i : T ! ε\_i ▷ B\_i

———————————————

Γ; Φ; Δ ⊢ route(pred){arm\_i} : T ! ⋃ε\_i ▷ join(B\_i)

**(ABSTAIN)**

A step may **return Abstain[κ]** if preconditions cannot be proved or UQ exceeds risk bands:

Γ; Φ ⊢ risk : Score{risk ≥ κ}

———————————————

Γ; Φ; Δ ⊢ abstain(κ) : Result[T,Abstain] ! {Abstain[κ], Emit[VIF]} ▷ 0

**32.4 Budgets as indexed types**

We track resource **potential** at compile-time:

* Budget[t:idx, s:idx] with monoid :  
  .
* Subtyping: Budget[t1,s1] <: Budget[t2,s2] iff .
* Each LLM/tool rule **consumes** a bounded slice; the checker ensures final total ≤ declared pipeline budget.

This yields **amortized resource analysis**: if it type-checks, runtime cannot exceed the budget unless a tool violates its contract (then the runtime raises Overbudget with VIF).

**32.5 Information-flow & policy logic (optional layer)**

* **Security labels:** Low ≤…≤ High; types lifted to T@ℓ.
* **Noninterference:** well-typed plan prevents High→Low leaks except through **declassified gates** with Witness{policy}.
* **Policy logic:** Datalog/OPA-style rules over VIF + SEG (e.g., “no seg\_write to external tenant without HITL witness”). Violations become **unsatisfied proof obligations** during checking.

**32.6 Symbolic checkers (toolchain)**

1. **Type/Effect/Budget checker (ACL-TC).**
   * Linear capability accounting; row-polymorphic effects; budget addition; refinement constraints to SMT (Z3).
   * Outputs: typed AST, global effect row, budget ceiling, **obligation set**.
2. **Gate satisfiability (SMT).**
   * Proves predicates used by g\_spec, g\_tests, g\_tags, g\_trace, g\_safety.
   * Unprovable → plan refuses to compile unless an Abstain[κ] arm exists.
3. **SEG/JSON-LD validator (SHACL).**
   * Ensures lineage graphs satisfy shape constraints (nodes, edges, temporal validity).
4. **Determinism checker.**
   * Flags nondeterministic sources (Random, Time, CallLLM with temp>τ, external net) unless bound behind Verifier gates with stabilization (e.g., temperature sweep + tie-break).
5. **Policy/Datalog engine.**
   * Confirms capability use matches policy; produces human-readable “why allowed” proof trace.

**32.7 Soundness theorems (informal but precise)**

**Theorem 32.7.1 (Progress).**  
If and all external tools/models are available, then **either** **or** (a reduction step exists) **or** with Witness{reason}.

**Theorem 32.7.2 (Preservation).**  
If and , then with and (budgets don’t grow; effects don’t appear from nowhere). Capability multiset evolves linearly.

**Theorem 32.7.3 (Budget safety).**  
For any well-typed plan, at runtime cumulative tokens/seconds **do not exceed** static Budget unless a tool violates its contract; such violations surface as Overbudget errors carrying VIF of the offender.

**Theorem 32.7.4 (Lineage totality).**  
Every boundary (plan start/end; tool/LLM call; gate) in a well-typed trace produces a Witness that links artifacts to inputs/tools/snapshots, ensuring VIF completeness.

**Theorem 32.7.5 (Noninterference, optional).**  
If a plan type-checks under labels and has no declassify edges, then changes to High inputs cannot alter Low outputs (modulo Abstain), i.e., confidentiality holds.

*Sketches:* all via induction on typing derivations and small-step operational semantics for ACL; budget safety leverages potential-style indices; noninterference follows classic security typing.

**32.8 Checker architecture & complexity**

* **Pass 1:** kinding, name resolution, role/capability binding (linear).
* **Pass 2:** type/effect inference with capability linearity (near-linear in |AST|).
* **Pass 3:** budget aggregation (monoid fold).
* **Pass 4:** obligations to solvers:
  + SMT (refinements/gates): worst-case NP-hard; in practice small and cached.
  + SHACL (SEG shapes): linear in graph size.
  + Datalog policy: polynomial; incremental.

**Artifacts:** typed AST; effect summary; budget table; solver proofs; **witness-carrying plan** ready for execution.

**32.9 Worked example (derivation sketch)**

**Plan:** retrieve → (reason || verify) → merge → emit.

step retrieve : Query -> List[Atom]

using Tool["vecstore", caps={seg\_read}]

with Budget[tokens=0, secs≤0.2]

step reason : List[Atom] -> Draft

using Model["gpt-x", temp=0.2]

with Budget[tokens≤8k, secs≤3]

step verify : Draft -> Verdict×Witness

using Tool["lint", caps={exec[js]}]

with Budget[0, secs≤0.5]

step merge : (Draft×Verdict) -> Answer×Witness

gate(g\_spec ∧ g\_trace ∧ g\_safety)

**Typing facts (selected):**

* retrieve effects {SEG[read]} ▷ (0,≤0.2)
* reason effects {CallLLM, Emit[VIF]} ▷ (≤8k,≤3)
* verify effects {CallTool["lint"], Emit[VIF]} ▷ (0,≤0.5)
* Parallel reason || verify: budgets sum → (≤8k, ≤3.5); effects union.
* merge requires gate proofs → emits Witness.
* Whole pipeline summary:
  + Type: Query -> Answer×Witness
  + Effects: {SEG[read], CallLLM, CallTool["lint"], Emit[VIF]}
  + Budget ceiling: Tokens≤8k, Secs≤3.7
  + Determinism: flagged **non-deterministic** unless verify stabilizes (e.g., temperature sweep + tie-break); checker suggests mitigations.

**32.10 Diagnostic strategy**

* **Missing capability:** “call(tool) requires seg\_write; have seg\_read.”
* **Budget under-spec:** “call\_llm needs Tokens≥12 345; declared ≤8 000.”
* **Unwitnessed boundary:** “merge lacks Emit[VIF] — add gate(g\_trace) or emit\_vif.”
* **Policy violation:** “net[external] forbidden without HITL proof (Policy §7.2).”
* **Determinism claim failed:** “source of nondeterminism: CallLLM(temp=0.8); add verifier or relax claim.”

Each error includes **fix-its** and links to the violated rule.

**32.11 Soundness notes & limitations**

* **Model/tool contracts** are assumed correct; breakage is caught at runtime with VIF.
* **Budgets** are upper bounds; real usage may be lower; we treat tool slack as OK.
* **Refinements** are only as strong as predicates and solver timeouts; we cache and bound.
* **Parallelism** assumes no shared mutable state beyond typed capabilities.

**32.12 Implementation notes**

* **Front-end:** EBNF from Ch.13 extended with type annotations and with Budget[…] clauses.
* **Middle-end:** Hindley–Milner + effect rows + linear capabilities; interval arithmetic for indices; global fold for budgets.
* **Back-end:**
  + SMT (Z3/CVC5) for refinements & gates
  + SHACL for SEG payloads
  + Datalog/OPA for policies
* **Artifacts written to SEG:** type cert, effect summary, budget cert, policy proof — bound to snapshot id.

**Outcome:** ACL’s type-and-effect discipline turns **plans into proofs**: if it compiles, it respects capabilities, budgets, and provenance duties by construction. That is the core of **APOE’s reliability** and the substrate upon which **VIF** and **SEG** become operational rather than aspirational.

**Part XIII — Roadmap & Governance**

**Chapter 33. 0-30-60-90 — milestones, risks, dependencies, resourcing & budgets**

Objective: take AIMOS/APOE from spec to *running*, *auditable* pilots in 90 days—with clear exit criteria, owners, budgets, and gates tied to the invariants (CMC/APOE/VIF/SDF-CVF/SEG).

**33.0 Scope & assumptions**

* **Scope:** CMC+HHNI (memory), DVNS (retrieval), APOE/DEPP (orchestration), VIF/SEG (provenance), SDF-CVF (atomic evolution), IDE-in-the-loop, security/compliance, observability, SDKs.
* **Non-goals in 90d:** custom base model training; full enterprise rollouts.
* **Assumptions:** one primary vector store + one evidence graph store; access to ≥2 foundation models; small cross-functional team (see 33.5).

**33.1 Workstreams (WS)**

* **WS-M:** Memory Core (CMC/HHNI, write/read pipelines, storage substrate).
* **WS-R:** Retrieval (DVNS physics layer, super-index, online refinement).
* **WS-O:** Orchestration (APOE/DEPP, ACL compiler & checker).
* **WS-V:** Verifiability (VIF schema, UQ, replay, SEG & time-slicing).
* **WS-E:** Atomic Evolution (SDF-CVF gates, quarantine/rollback).
* **WS-I:** IDE-in-the-Loop pipeline (blueprint→deploy).
* **WS-S:** Security & Compliance (guardrails, policies, mapping).
* **WS-X:** Observability & Costing (OTel for agents, budgets, drift).
* **WS-K:** SDKs/Clients (TS, Python, CLI) + API surface.

**33.2 0-30-60-90 plan (deliverables, exit gates)**

**Day 0 → Day 30 (“Alpha substrate”)**

**Deliverables**

* WS-M:
  + Fractal Memory Hierarchy schemas (Atom/Edge) and **CreateMemory** transactional flow (C-1/C-2).
  + HHNI **Index-of-Indexes** builder (system→…→sub-word) with TPV and DD computation.
  + Read path v1: dumbbell compression, two-stage read (C-3).
* WS-R:
  + DVNS core loop (grav/elastic/repulse/damping) on top-k rerank; **C(P)** cost functions.
  + Super-index offline precompute for a seed corpus.
* WS-O:
  + ACL grammar (EBNF), minimal type/effect checker (budgets, capabilities).
  + DEPP skeleton: plan graph + evaluator-optimizer loop.
* WS-V:
  + VIF tuple (model id, weights hash, prompt template id, snapshot id, tools).
  + UQ baseline: temperature sweep + entropy; calibration dashboard (ECE).
  + SEG JSON-LD shapes; write path for boundary events.
* WS-E:
  + Gate catalog v1 (g\_spec, g\_tests, g\_tags, g\_trace, g\_safety); quarantine workflow.
* WS-I:
  + Blueprint YAML and scaffolder for one stack (UI/API/Auth/Data).
* WS-S:
  + Threat model; capability tokens; redaction; abstention/κ rules.
* WS-X:
  + OTel spans for steps/tools; token/latency meters; cost model stub.
* WS-K:
  + TS client (Memory API + Audit read-only); basic CLI.

**Exit gates (must pass)**

* Snapshot-first: no user-visible state without snapshot id.
* VIF coverage ≥ 90% at boundaries; replay on frozen snapshot succeeds.
* RS-lift ≥ +10% vs. static KNN on a seed task; κ-abstentions logged with rationale.
* Budget checker blocks over-budget plans at compile time.

**Day 31 → Day 60 (“Beta orchestration”)**

**Deliverables**

* WS-M:
  + Impact previews (DD) + dependency hashing; partial rollback.
  + Caching/preload (Markov traversals), perf targets documented.
* WS-R:
  + Online DVNS refinement on retrieval results; policy-aware geometry (forbidden crossings).
  + Super-index refresh strategy; convergence monitors.
* WS-O:
  + ACL type-and-effect **with linear capabilities** and **budget indices**; determinism flags.
  + Router (role-aware), parallelization, tool contracts; plan diff + health (κ\_chain).
* WS-V:
  + UQ ensembles (two models) + calibration (per-domain bands).
  + SEG time-slicing queries; lineage → decision views; export pack v1.
* WS-E:
  + Same-commit parity enforcement across code/docs/tags/traces; ADR templates.
  + Auto-fix strategies; rollback catalog; MTTR SLO.
* WS-I:
  + Pipeline to **Test/Preview Deploy**; coverage gates + parity checks.
* WS-S:
  + Capability sandbox for tools; SSRF/injection playbooks; policy packs (PII/licensing).
* WS-X:
  + Drift & prompt-rot monitors; budget SLOs; cost dashboards.
* WS-K:
  + Python client parity; paginators, retries; end-to-end examples.

**Exit gates**

* Chain determinism (under fixed seeds/tools) **or** documented nondeterminism with verifier stabilization.
* VIF lineage completeness ≥ 98% (measured over all boundaries).
* RS-lift ≥ +20% and nDCG@20 ≥ baseline +0.06 on two domains.
* Incident drill: SEG-driven replay + RCA in ≤ 30 min.

**Day 61 → Day 90 (“Pilot-ready, auditable”)**

**Deliverables**

* WS-M:
  + SLOs met (p95 write < 200 ms; read < 150 ms on top-k 64; p95 DD preview < 50 ms).
  + Snapshot log with tombstones; retention configs.
* WS-R:
  + DVNS parameter auto-tuner; stability regions documented; abstention bands.
* WS-O:
  + ACL checker **v1.0** (SMT + SHACL + Datalog policies); proof artifacts to SEG.
  + DEPP thousand-node planning with budget adherence alarms.
* WS-V:
  + UQ+confidence UX (bands, tooltips, stratified warnings).
  + Compliance export pack (EU-style logging bundle; synthetic content labels).
* WS-E:
  + DORA telemetry live (CFR, MTTR); quarantine auto-unblock with evidence.
* WS-I:
  + IDE-in-the-loop demo: blueprint→running app→observe with full witnesses.
* WS-S:
  + Risk registers; residual-risk sign-off; two-key HITL for high-risk merges.
* WS-X:
  + OTel traces stitched to SEG lineage; cost/latency error budgets enforced.
* WS-K:
  + CLI golden paths; SDK samples (memory write/read; plan compile/run; audit query).

**Exit gates (pilot-go)**

* Reproducible replay on frozen snapshot across all pilots.
* κ-abstention precision ≥ 0.8 (manual judged on sampled queries).
* No P0/P1 security findings; policy engine blocks forbidden capability paths.
* One greenfield + one brownfield pilot running with on-call playbooks.

**33.3 Dependency graph (high level)**

* **ACL checker (WS-O)** depends on **VIF/SEG shapes (WS-V)** and **capability catalog (WS-S)**.
* **DVNS (WS-R)** depends on **HHNI read APIs (WS-M)**.
* **Quarantine/rollback (WS-E)** depends on **Snapshot log (WS-M)** and **witnessed tests (WS-I)**.
* **Compliance export (WS-V)** depends on **SEG time-slicing (WS-V)** + **OTel stitching (WS-X)**.

**33.4 Risks & mitigations**

| **Risk** | **Impact** | **Mitigation** |
| --- | --- | --- |
| DVNS oscillation/instability | Bad retrieval, cost blowups | Damping + step-size schedule; auto-tuner; abstention bands |
| Index bloat / DD spikes | Slow writes/reads | Adaptive pruning by TPV; shard super-index; impact previews before persist |
| Over-budget chains | Cost/SLO violations | Budget indices in ACL; compile-time fail; runtime breaker w/ VIF |
| Tool prompt injection | Data exfiltration, corruption | Capability tokens; sandbox FS/NET; policy engine; redaction gates |
| VIF gaps | Audit failure | Boundary macros emit VIF; coverage monitors; build breaks < 98% |
| HITL bottleneck | Throughput drops | Risk tiering; two-key only for ≥R2; reviewer rotations; SLAs |
| Nondeterminism | Irreproducibility | Temperature sweeps + tie-break; determinism flags; verifier acceptance tests |

**33.5 Resourcing plan (people/roles)**

* **Core (7–9 FTE):**
  + Lead Architect (overall invariants & interfaces)
  + Memory/Storage Eng (CMC/HHNI, perf) ×2
  + Retrieval/ML Eng (DVNS, ranking) ×1
  + Orchestration Eng (ACL/DEPP compiler) ×1
  + Platform/SRE (OTel, CI/CD, cost) ×1
  + Security/Policy Eng (caps, guardrails, compliance) ×1
  + DX/SDK Eng (TS/Py/CLI) ×1
* **Adjunct:** Designer (confidence UX), Tech Writer (ADRs), Program Manager.

**33.6 Budget & cloud footprint (order-of-magnitude)**

* **Infra:** vector store, graph DB, object store, CI runners, observability stack.
* **Model spend:** two providers for ensemble UQ + low-cost model for bulk.
* **90-day envelope (illustrative):**
  + Infra: $12–20k, Models: $15–30k, People: team-dependent.
* **Controls:** per-project **token budgets**, daily spend alerts, breaker on budget breach (emits VIF).

**33.7 Definitions of Ready/Done**

* **DOR (any task):** ADR exists; inputs/schemas fixed; owners named; tests & gates listed; budget class assigned.
* **DOD (feature):** passes gates (spec/tests/tags/trace/safety); witness present; docs updated; SEG lineage query returns non-empty; dashboards show green SLOs.

**33.8 Governance rituals**

* **Daily:** standup (risks, blockers, κ events).
* **Weekly:** gate review (failed gates, quarantines), budget & drift review, ADR board.
* **Bi-weekly:** pilot steering—SEG lineage demo; incident learnings.
* **Monthly:** compliance snapshot (export pack), security tabletop.

**33.9 Program KPIs (tied to invariants)**

* **CMC:** precision@k@latency, RS-lift vs. KNN, snapshot replay pass %.
* **APOE:** plan determinism rate, budget adherence %, κ precision/recall.
* **VIF/SEG:** lineage completeness %, replay fidelity, export validity.
* **SDF-CVF:** CFR, MTTR, spec↔code parity score.
* **IDE-loop:** blueprint→deploy lead time, preview pass rate.
* **Security:** blocked policy violations, injection detections, residual risk.

**Outcome:** in 90 days we stand up an **auditable, memory-native, agentic system** delivering two pilots with **witnessed, reproducible** traces, cost-bounded orchestration, and policy-enforced guardrails—ready to scale or harden per domain.

**Part XIII — Roadmap & Governance**

**Chapter 34. Community & Contribution — ADR templates, review rotations, ethics & use policies**

Objective: make AIMOS/APOE a **living, accountable** project where ideas become **witnessed decisions**, contributions evolve **atomically**, and community trust is earned through **transparent process**.

**34.0 Scope & principles**

* **Memory-native community:** every decision is an artifact (ADR/RFC) with **VIF** and a link into the **SEG**.
* **Atomic evolution:** code/docs/tags/traces change **together** (SDF-CVF).
* **Safety-first:** abstention, redaction, and HITL are **policies**, not suggestions.
* **Open by default, governed by evidence:** lazy consensus for low risk; two-key for high risk.

**34.1 Contribution model**

**Repos & branches**

* main (protected, signed, green-only), develop (integration), feature branches feat/\*, hotfix fix/\*.
* **CODEOWNERS** required for memory, orchestration, and audit packages.
* All PRs must include: linked issue, **witness** (VIF JSON), **gates** status (tests/spec/tags/trace/safety).

**Labels**

* Risk tiers: risk:R0 (docs), R1 (low), R2 (med), R3 (high).
* Areas: area:cmc, area:dvns, area:apoe, area:vif, area:seg, area:ide, area:security.
* Outcomes: needs-adr, needs-rfc, needs-hitl, quarantined.

**Decision flow**

* Idea → Issue → **RFC** (design options) → **ADR** (chosen option) → PR(s) → Merge (if gates green) → Release note.
* High-risk (R2/R3): **two-key** reviewers (domain owner + safety owner); human-in-the-loop sign-off recorded in **SEG**.

**34.2 ADRs — templates & process**

**When to write an ADR**

* Anything that changes an interface, storage schema, ACL semantics, security posture, or compliance behavior.
* Anything a future maintainer would need to **understand or challenge**.

**ADR lifecycle**

* Proposed → Accepted → Superseded or Rejected.
* Every ADR gets a **SEG node** with edges to the PRs, tests, and benchmarks that evidence it.

**ADR Template (docs/adr/XXXX-title.md)**

# ADR XXXX: <Title>

Status: Proposed | Accepted | Superseded by ADR-YYYY | Rejected

Date: YYYY-MM-DD

Owners: @handle1 (@area), @handle2 (@safety)

Risk: R0 | R1 | R2 | R3

Scope: area:cmc, area:apoe (labels)

## Context

- Problem statement, constraints, stakeholders.

- Prior art / alternatives considered (link RFCs, issues).

## Decision

- Chosen approach; diagrams or schemas as needed.

- Interfaces affected (APIs, schemas, ACL constructs).

## Consequences

- Benefits, trade-offs, risks, migration plan.

- Deprecations/tombstones; impact on SLOs/cost.

## Evidence (VIF)

```json

{

"model\_id": "xxx",

"weights\_hash": "sha256:...",

"prompt\_template\_id": "PT-123",

"tools": ["benchmark:rs\_lift@v2", "proof:acl\_checker@v1"],

"snapshot\_id": "snap-2025-10-17T12:34Z"

}

**Gates & Verification (SDF-CVF)**

* g\_spec: ✅ docs updated
* g\_tests: ✅ unit/integration load
* g\_tags: ✅ HHNI tags + TPV updated
* g\_trace: ✅ SEG lineage query returns non-empty
* g\_safety: ✅ κ-bands honored; policy tests pass

**Links**

* Issues: #123, #456
* PRs: !789, !790
* SEG node: seg://adr/XXXX

---

## 34.3 RFCs — when & how

- \*\*RFC\*\* precedes ADR for \*\*ambiguous\*\* or \*\*cross-cutting\*\* design questions.

- Time-boxed (typically 7–14 days), with explicit call for dissent and alternatives.

\*\*RFC Template (short)\*\*

```markdown

# RFC: <Problem/Opportunity>

Status: Open | Closed

Window: 2025-10-17 → 2025-10-31

Facilitator: @handle

## Goals / Non-goals

## Options

- Option A (pros/cons, risks, cost)

- Option B ...

## Evaluation Matrix

| Option | Perf | Cost | Risk | Compliance | Complexity |

|--------|------|------|------|------------|------------|

## Proposed Next Step

- Draft ADR? Pilot spike? Abort?

**34.4 Review rotations & decision rules**

**Rotations**

* Weekly **area-based** rotations: CMC, DVNS, APOE, VIF/SEG, Security, IDE/DX.
* Each PR auto-assigns **one area reviewer** + **one safety reviewer** (capability owner).
* Load balancing: max 5 active PRs per reviewer; auto-reassign on breach.

**Decision rules**

* **Lazy consensus** (R0/R1): merge if no substantive objections in 48h and gates green.
* **Two-key approval** (R2/R3): domain + safety must approve; HITL record attached (SEG).
* **Escalation:** unresolved after 5 business days → TSC vote (simple majority).

**RACI**

* **Responsible:** PR author; **Accountable:** area owner; **Consulted:** safety, compliance; **Informed:** release manager.

**34.5 Governance structure**

* **TSC (Technical Steering Committee):** area leads + security lead + PM.
* **Charter:** uphold invariants (CMC/APOE/VIF/SDF-CVF/SEG), arbitrate disputes, freeze plans for releases, own roadmap.
* **Decision log:** TSC decisions are **ADRs** with Decision scope.

**34.6 Ethics & use policies (AUP)**

**Core principles**

* **Transparency:** machine-readable provenance by default.
* **Minimality:** collect, store, and expose **only** what is necessary.
* **User agency:** opt-outs for data reuse; meaningful consent.
* **Safety over speed:** κ-abstain on uncertainty; HITL for high risk.

**Prohibited uses (non-exhaustive)**

* Surveillance that violates local law or user consent.
* Generation of malware or exploit kits.
* Discrimination or harmful profiling.
* Biometric identification without explicit, lawful basis.

**Dataset policy**

* License & provenance **required**; SEG nodes for source → slice → usage.
* PII: redact at **ingest**; DP/noise where aggregation is published.

**Red-team rules**

* Attack surfaces: prompt injection, tool SSRF, data exfiltration.
* Only against **staging** with issued tokens; report via security policy; no public disclosure before fix window ends.

**34.7 Legal: licensing, CLA/DCO, IP**

* **Code license:** Apache-2.0 (patent grant).
* **Content/specs:** CC-BY-4.0.
* **Models/artifacts:** model cards + separate license notices.
* **CLA** (individual & corporate) **or** **DCO** (Developer Certificate of Origin) — one is mandatory.
* **Third-party deps:** SBOM + license scan in CI; failing scan blocks merge.

**SECURITY.md (vuln disclosure skeleton)**

Email: security@aimos.dev • PGP: https://aimos.dev/pgp.txt

Embargo: 90 days by default

Triage SLA: 3 business days

CVSS: NVD scoring; patches shipped before disclosure

**34.8 Community ops**

**Issue templates** (select at creation)

* bug\_report.md (repro, expected/actual, logs, version).
* feature\_request.md (problem, proposal, alternatives, risks).
* doc\_gap.md (missing, audience, acceptance).

**PR template** (excerpt)

## What & Why

## Linked Issue(s)

## ADR / RFC

- [ ] ADR-XXXX attached -or- N/A (low risk)

## Gates (paste CI summary)

- g\_spec: ☐ g\_tests: ☐ g\_tags: ☐ g\_trace: ☐ g\_safety: ☐

## VIF (machine-readable)

```json

{ "snapshot\_id":"...", "tools":["..."], "model\_id":"..." }

\*\*Docs style\*\*

- “Explain → Decide → Evidence → How-to”.

- Every page ends with \*\*SEG lineage\*\* and “last reviewed” stamp.

\*\*Release notes\*\*

- Structured by area; include \*\*migration\*\* steps and \*\*risk\*\* callouts; link ADPs (audit data packs).

---

## 34.9 Onboarding & education

- \*\*First hour:\*\* clone, run `make dev-up`, write your first \*\*memory atom\*\*, run \*\*SEG query\*\*.

- \*\*First day:\*\* read 3 foundational ADRs; pass policy quiz (AUP).

- \*\*First week:\*\* shadow a review; author a doc PR; run a red-team drill in staging.

\*\*Learning rituals\*\*

- Weekly tech talk; monthly “ADR retrospective”; quarterly architecture review with external advisors.

---

## 34.10 Community KPIs

| KPI | Target | Notes |

|---|---|---|

| Time to first response (issues) | ≤ 24h | Business days |

| PR review turnaround | p75 ≤ 3d | From ready-for-review |

| ADR age (open) | p75 ≤ 10d | Exceptions require TSC note |

| VIF coverage at boundaries | ≥ 99% | CI guardrail |

| Quarantine resolution | p95 ≤ 3d | From isolation to merge |

---

## 34.11 Templates & skeletons

\*\*CODEOWNERS\*\*

/cmc/ @area-cmc  
/dvns/ @area-dvns  
/orchestration/acl/ @area-apoe  
/audit/ @area-vif @area-security  
/seg/ @area-vif

\*\*REVIEW\_ROTATION.yaml\*\*

```yaml

areas:

cmc: [alice, bob, carol]

dvns: [dave, erin]

apoe: [frank, gita]

vifseg: [hugo, ida]

security: [sam, taye]

rule:

max\_active\_prs\_per\_reviewer: 5

backup\_after\_days: 3

safety\_co\_reviewer: true

schedule:

timezone: UTC

rollover\_weekday: Mon

**GOVERNANCE.md (outline)**

- Charter & invariants

- Roles (contributors, maintainers, TSC)

- Decision process (lazy consensus, two-key)

- Elections & terms for TSC

- Conflict resolution & appeals

- Amendments (ADR-backed)

**ETHICS\_CHECKLIST.md (per change)**

* Data sources licensed & documented (SEG link)
* PII redaction validated
* κ-abstention bands set for new surfaces
* HITL configured for R2/R3 flows
* Model & prompt versions recorded (VIF)

**34.12 Enforcement & dispute resolution**

* **Code of Conduct:** Contributor Covenant; reports to conduct@aimos.dev.
* Graduated responses: mediation → temporary restrictions → removal.
* All enforcement events create **SEG records** with redacted details for transparency.

**34.13 Community roadmap hooks**

* Quarterly public roadmap review (recorded, minutes published).
* Community RFC weeks (no merges except fixes; discuss long-range proposals).
* “Good-first-issue” + “mentored-PR” lanes.

**34.14 Quickstart for contributors (TL;DR)**

1. Fork & branch (feat/…).
2. If in doubt, open **RFC**. If decided, write **ADR**.
3. Run tests + policy suite; attach **VIF**.
4. Open PR with gates summary; reviewers auto-assigned by rotation.
5. Respond to reviews; for R2/R3, obtain **two-key** approvals.
6. Merge (bot enforces green checks); release notes & SEG lineage auto-updated.

**Outcome:** a community that **thinks in evidence**, **evolves atomically**, and **governs by invariants**—so progress compounds without sacrificing safety or trust.

**Appendices & Back Matter**

This section packages the system for **builders, auditors, and readers-at-speed**: crisp definitions, runnable labs, line-of-sight examples, exportable compliance bundles, and operational playbooks. Everything here is memory-native (CMC), orchestrated (APOE), witnessed (VIF), atomically evolved (SDF-CVF), and anchored in the shared evidence graph (SEG).

**Appendix A — Glossary (Expanded)**

**AIMOS** — *AI Memory Operating System.* The substrate that converts context into atoms, indices, snapshots, and evidence graphs.

**APOE** — *AIMOS Prompt Orchestration Engine.* Compiles reasoning into typed chains (DAGs) with budgets, gates, and witnessed traces.

**ACL** — *AIMOS Chain Language.* Minimal DSL to define pipelines/steps/gates/budgets; statically checkable with types & effects.

**Atom (Memory Atom)** — Smallest persisted unit (text, code line, log record, user event). Carries content, modality, tags, embedding, and provenance.

**Molecule** — Higher-order grouping of atoms (e.g., function, paragraph, session). Recursively nestable.

**CMC** — *Context Memory Core.* Write/read pipelines, HHNI indexing, snapshots, rollback.

**HHNI** — *Hyper-Hierarchical Neural Indexing.* Indices from system→section→paragraph→sentence→word→sub-word, with dependency hashing & impact previews.

**DVNS** — *Dynamic Vector Navigation System.* Physics-inspired retrieval using GODN forces and NP path costs over embedding space.

**GODN** — *Graviton Organic Dynamics Network.* Force model: gravitational (attract), elastic (stabilize), repulsion (avoid density), damping (converge).

**SEG** — *Shared Evidence Graph.* JSON-LD/temporal graph for claims, sources, derivations, tools, and decisions; time-sliced queries and export packs.

**VIF** — *Verifiable Intelligence Frame.* Machine-readable provenance bundle: model id, weights hash, prompt template id, tools, snapshot id, uncertainty.

**SDF-CVF** — *Same-Delta Fusion / Cohesive Versioning Framework.* Atomic commits across **code/docs/tags/traces** with parity gates & rollback.

**QS** — *Quality Score.* Local content quality/fitness (normalized [0,1]).

**IDS** — *Indexing Depth Score.* Structural coverage (hierarchy depth × density).

**DD** — *Dependency Delta.* Change impact across the dependency graph ([0,1]); high DD triggers impact previews/HITL.

**RS** — *Retrieval Score.* RS = norm(QS · IDS · (1−DD)), calibrated to approximate p(relevance).

**TPV** — *Tag Priority Vector.* [priority, relevance, decay] used for compression, caching, and preloading.

**κ (kappa)** — Abstention threshold; if uncertainty > κ or RS < κ, the system declines and routes to HITL.

**ECE** — Expected Calibration Error; gap between predicted confidence and empirical accuracy.

**Snapshot** — Content-addressed, immutable bundle of atoms, indices, and references; basis for replayability.

**Gate** — Verifiable condition (tests/spec/safety/provenance/trace) required at plan boundaries and merges.

**Appendix B — Acronyms**

AIMOS, APOE, ACL, ADR, AUP, CMC, CVE, DAG, DVNS, ECE, GNN, GODN, HITL, HHNI, KNN, MTTR, nDCG, NP (Natural Processes), OTel (OpenTelemetry), PII, PR, QA, RAG, RCA, RS, SBOM, SEG, SDF-CVF, SLO, SOP, TPV, TSC, UQ, VIF.

**Appendix C — Extended End-to-End Examples**

**C.1 Machine Communication: Protocol Translation (with VIF/SEG)**

**Goal** Convert a proprietary CSV log → standardized JSON events, detect anomalies, and produce an auditable summary.

**APOE plan (ACL excerpt)**

pipeline translate\_and\_analyze {

budget tokens=12k, tools=["csv", "json", "anomaly"], time="2m";

step ingest(input:file.csv) -> atoms:a[] gate g\_trace;

step normalize(atoms:a[]) -> atoms:b[] gate g\_spec;

step detect(atoms:b[]) -> report:r, witness:w gate g\_safety;

step summarize(report:r, witness:w) -> brief:txt gate g\_prov;

}

**CMC write path**

1. ingest → **atoms** (type: log\_line), embeddings, tags (#timestamp, #source).
2. HHNI builds sentence/word sub-indices for field names/values.
3. Snapshot snap-2025-10-17T12:00Z is created before analysis.

**VIF (attached to detect)**

{

"model\_id": "apoe-evaluator@2025.10",

"weights\_hash": "sha256:f3…",

"prompt\_template\_id": "PT-logs-std-v7",

"tools": ["csv@1.2","json@2.1","anomaly@0.9"],

"snapshot\_id": "snap-2025-10-17T12:00Z",

"uncertainty": { "ece": 0.027, "entropy": 0.18, "kappa": 0.75 }

}

**SEG lineage (compact)**

log\_lines -> normalized\_events -> anomaly\_candidates

supports derives supports

summary\_claim ("2.3% anomaly rate") <- detect <- evidence:set

**DVNS retrieval**

* Query: “spike between 14:20–14:30 UTC?”
* Forces pull time-adjacent atoms; elastic forbids crossing batches.
* RS rises to 0.86; κ passes; answer emitted with confidence band ±0.4%.

**C.2 Builder Program: From Blueprint to Running Service**

**Blueprint YAML (excerpt)**

app: telemetry-bridge

domains: [ingest, transform, observe]

spec:

ingest: http POST /v1/events

storage: vector+object

outputs: SEG nodes + metrics

nonfunc: {slo: {latency\_p95\_ms: 120}, budget: {tokens: 40k}}

**APOE compile → scaffold**

* Generates service skeleton, tests, IaC stubs, SEG exporter.
* SDF-CVF gate summary on PR: g\_spec✅ g\_tests✅ g\_tags✅ g\_trace✅ g\_safety✅.

**CMC/HHNI**

* Each route, schema, and test is an **atom**; molecules represent packages.
* IDS rises as documentation and tests index deeper; RS-lift observed in code search.

**Deploy & Observe**

* OTel spans emit VIF snippets at step boundaries; SEG accumulates claim→evidence→decision.

**C.3 Ops Incident: RCA & Safe Autoremediation**

**Symptom** Error budget burn on /transform p95 latency.  
**Flow**

1. Query SEG for last 48h changes touching transform.
2. DVNS clusters two commits + config change; RS=0.91 for culprit chain.
3. Autoremediation plan (APOE) proposes rollback; κ triggers HITL (two-key).
4. Post-incident ADR links to SEG, adds guardrail tests; all parity gates pass.

**Appendix D — Figures & Tables List**

**Figures**  
F1. CMC Write/Read pipeline (atoms→indices→snapshot→graph)  
F2. HHNI fractal index (system→…→sub-word) with dependency hashing  
F3. DVNS field lines (grav/elastic/repulse/damping) over embedding space  
F4. APOE plan execution DAG with gates and witness emission points  
F5. SEG time-sliced lineage (claim→evidence→decision)  
F6. SDF-CVF parity across code/docs/tags/traces  
F7. IDE-in-the-loop: blueprint→scaffold→code→test→deploy→observe

**Tables**  
T1. Invariants→Constraints Traceability  
T2. RS components (QS/IDS/DD) and calibration bins  
T3. κ-bands and UI confidence cues  
T4. Gate catalog and default policies  
T5. Risk taxonomy (R0–R3) vs. required controls  
T6. Compliance artifact mapping (EU AI Act/SOC2/ISO)  
T7. Observability fields (OTel span schema)  
T8. Cost budgets (tokens/time/tools) by plan class

**Appendix E — Change Log (Excerpt)**

**v0.1 — Initial Extreme Textbook**

* Parts I–XIII complete: invariants, CMC/HHNI, DVNS, APOE/DEPP, VIF/SEG, SDF-CVF, IDE-in-loop, security/compliance, benchmarks, observability, cases, reference impls, math, governance.
* Appendices A–J and Back Matter established.
* Known gaps: ACL formal semantics examples (more), additional export pack recipes, extended labs.

**Appendix F — Index of Indexes (HHNI Cross-Reference)**

**Path syntax**  
//system/section[n]/paragraph[m]/sentence[k]/word[j]  
Example:

* //apoe/acl/2/5/3 → 3rd sentence of §5 in Chapter 2 (APOE grammar).
* //cmc/schemas/1/4/\* → all words in §4 of schemas section.

**Cross-links**

* Every node carries dep\_hash, parents[], siblings[], cross[] (semantic) with weight and immutability\_flag.
* **Impact preview** on edit computes DD = Σ w\_i · |Δtag\_i| with alert if DD > θ.

**Query examples (pseudo-QL)**

SELECT word FROM hhni

WHERE path LIKE '//cmc/%/sentence/%'

AND tags CONTAIN '#definition'

ORDER BY tpv.priority DESC

LIMIT 50;

**Appendix G — Quickstarts & Labs**

**G.1 10-Minute Memory Atom**

1. **Create atom**

POST /v1/memory/atoms

{ "modality":"text", "content":"Hello, AIMOS.", "metadata":{"src":"lab"} }

1. **Search**

POST /v1/memory/search

{ "query":"AIMOS", "top\_k":5 }

1. **Inspect snapshot**

GET /v1/memory/snapshots/{id}

**G.2 DVNS Retrieval Lab**

* Seed 500 atoms; run DVNS refinement on Top-k=20.
* Tune (G,k,α,β,γ); observe RS-lift vs. static KNN.
* Plot convergence (ΔE per iteration) and abstentions by κ-band.

**G.3 Orchestration Lab (ACL)**

* Write a pipeline research\_then\_build { … } with budgets and gates.
* Run, capture VIF at boundaries, replay against a frozen snapshot.

**G.4 IDE-in-Loop Mini-App**

* Blueprint YAML → scaffold REST service → generate tests → local deploy.
* Verify SDF-CVF parity: changing spec breaks tests until code updated.

**Appendix H — Troubleshooting Guide**

| **Symptom** | **Likely Cause** | **Triage** | **Fix** |
| --- | --- | --- | --- |
| Low RS despite good QS | Shallow IDS | Deepen HHNI; add paragraph/sentence indices | Re-index; raise preload.depth |
| Frequent κ-abstentions | Poor calibration | Run ECE calibration; widen κ bands | Temperature sweep; retrain UQ head |
| Chain nondeterminism | Hidden tool variability | Pin tool versions; freeze snapshot | Add seed discipline in ACL |
| SEG queries slow | Time-slice explosion | Add temporal filters; prune low-weight edges | Enable TTL for stale nodes |
| Quarantine storms | Over-strict gates | Review thresholds; risk-tier overrides | Two-key policy adjustments |

**Appendix I — Compliance Artifacts (Sample Export Packs)**

**I.1 VIF Bundle (JSON)**

{

"artifact\_id":"claim-42",

"snapshot\_id":"snap-2025-10-17T12:34Z",

"model":{"id":"gpt-x@2025.10","weights\_hash":"sha256:..."},

"prompt\_template\_id":"PT-claims-v3",

"tools":["retriever@1.4","verifier@2.0"],

"uncertainty":{"ece":0.031,"band":"green"},

"created\_at":"2025-10-17T12:35:22Z"

}

**I.2 SEG Export (JSON-LD)**

{

"@context":"https://schema.aimos.dev/seg/v1",

"@graph":[

{"@id":"claim:42","@type":"Claim","text":"Anomaly rate 2.3%"},

{"@id":"e:1","@type":"Evidence","source":"snap-...","supports":"claim:42","valid":"[2025-10-17T12:00Z, )"}

]

}

**I.3 EU AI Act Pack**

* Log retention manifest (≥6 months), lineage coverage %, κ-abstention policy, synthetic labeling config, DPIA summary, incident response checklist, model card.

**Appendix J — Risk Taxonomy & Playbooks**

**Risk tiers**

* **R0 Docs/UI** — Low: lazy consensus merge.
* **R1 Retrieval/UX** — Medium: 1 approver + green gates.
* **R2 Orchestration/Agents** — High: **two-key** + HITL logs.
* **R3 Security/Compliance** — Critical: TSC approval + staged rollout + rollback plan.

**Playbooks (abridged)**

* **Prompt Injection Detected**  
  *Signals:* unusual tool calls, outbound spikes.  
  *Actions:* κ-abstain → sandbox halt → SEG tag contradicts → patch guardrails → retrospective ADR.
* **Calibration Drift**  
  *Signals:* ECE ↑, κ false negatives.  
  *Actions:* Recalibrate bins → temperature sweep → update UQ model → publish VIF deltas.
* **Evidence Gap**  
  *Signals:* claim without supporting edges.  
  *Actions:* Re-run retrieval with policy expansion; if absent, downgrade confidence or retract.
* **Parity Violation (SDF-CVF)**  
  *Signals:* code passes but spec/tags/trace missing.  
  *Actions:* Auto-fix PR; else quarantine; owner escalation.

**Back Matter**

**Bibliography (Selected, neutral)**

* **Transformer & Long-Context:** Vaswani et al. *Attention Is All You Need*; “Lost in the Middle” analyses; efficient long-context surveys.
* **Sub-Quadratic Alternatives:** Hyena Hierarchy papers; Liquid Neural Networks (MIT); state-space models.
* **Prompt Orchestration & Agents:** Multi-agent planning frameworks; evaluator-optimizer patterns; chaining & routing guides.
* **Retrieval & Indexing:** Dense embeddings, hybrid retrieval, calibration & reranking literature.
* **Provenance & Compliance:** EU AI Act texts and summaries; model cards & data statements; content provenance standards.
* **Observability & Reliability:** OpenTelemetry specs; SRE handbooks; calibration (ECE) and LLM evals (HELM-style).
* **Safety & Security:** Prompt-injection mitigations; capability sandboxing; privacy-preserving ML.

*Note:* The project repository links each reference to SEG nodes for machine-readable lineage.

**Subject Index (Mini)**

AIMOS (overview) — Part I Ch.1 • CMC — Ch.4–7 • HHNI — Ch.5 • DVNS — Ch.8–10 • APOE — Ch.11–12 • ACL — Ch.13 • VIF — Ch.14 • SEG — Ch.15 • SDF-CVF — Ch.16 • HITL — Ch.17 • IDE-in-loop — Ch.18–19 • Security — Ch.20 • Compliance — Ch.21 • Benchmarks — Ch.22 • Observability — Ch.23 • Cases — Ch.24–26 • Schemas — Ch.27 • APIs — Ch.28 • SDKs — Ch.29 • Math — Ch.30–31 • Logic — Ch.32 • Roadmap — Ch.33 • Community — Ch.34 • Glossary — Appx-A.

**Author/Editor Notes**

* **Intent:** make a **working textbook**—specs you can run, decisions you can audit, plans you can replay.
* **Style:** short nouns, typed schemas, examples > exposition.
* **Living doc:** every section is versioned as atoms/molecules with SEG lineage and ADR cross-links.

**Ready-to-Use Artifacts (recap)**

* **VIF schema** (Ch.14, Appx-I)
* **SEG JSON-LD** (Ch.15, Ch.27, Appx-I)
* **ACL grammar & examples** (Ch.13, Ch.27)
* **Gate catalog & policies** (Ch.16, Ch.20)
* **Quickstarts & Labs** (Appx-G)

**1. Scalability and Performance**

**Rationale**: Your design emphasizes efficiency through mechanisms like HHNI's fractal indexing (Chapter 5), DVNS's physics-inspired refinement (Chapters 8–10), and SLOs (e.g., p95 write < 200ms in Chapter 6). However, for AGI-scale operations—handling petabyte corpora, real-time multi-user queries, or dynamic growth—the system might encounter bottlenecks in ingestion, indexing depth, or force computations. Static KNN baselines are useful for RS-lift metrics (Chapter 22), but without distributed handling, super-index precomputation could become a single point of failure, especially with recursive hierarchies that grow fractally to atomic levels (e.g., word/sub-word).

**Challenges/Gaps**:

* **Ingestion Bursts**: The write path (ingest → atomize → enrich → index → score → gate → snapshot in Section 1.2) is transactional and single-writer (Constraint C-1), which ensures determinism but could serialize high-volume inputs (e.g., 3GB logs in your example). If bursts exceed SLOs, DD previews might delay gates, leading to quarantines under SDF-CVF (Chapter 16).
* **Indexing Overhead**: HHNI's multi-granular embeddings (system → sub-word) with dependency hashing scales O(log N) for lookups, but building impact previews on edits (DD computation) could be O(N) in dense graphs, especially if edges (supports/contradicts) proliferate in SEG (Chapter 15).
* **DVNS Compute**: GODN forces (gravity/elastic/repulse/damping in Chapter 31) converge via Velocity-Verlet integration, but for large top-k (e.g., 64+ candidates), iterations might exceed budgets (e.g., tokens/time in APOE, Chapter 11). Stability regions (CFL bounds) are well-defined, but without parallelization, online refinement could lag in distributed setups.

**Detailed Recommendations**:

* **Distributed Variants**: Extend HHNI to a sharded, vector-graph hybrid using tools like Apache Cassandra for the evidence graph store (Chapter 7) or Faiss for embeddings. For super-index precomputation (Section 9.3), implement a MapReduce-style offline job that partitions by TPV priority vectors—e.g., high-priority atoms (priority > 0.7) on faster shards. This could integrate with Markov traversals for caching (Section 6.3), preloading likely paths based on historical queries.
* **Burst Handling**: Introduce asynchronous queues before the single-writer gate, batching atoms into molecules (Chapter 4) with temporary QS approximations. For SLOs, add adaptive throttling: if mean DD > θ\_d (e.g., 0.3), escalate to HITL (Chapter 17) or degrade to coarse reads. Metrics: Track p99 latency on writes; aim for RS-lift ≥ +20% vs. baselines under load (empirical lift in Section 9.4).
* **DVNS Optimization**: Parallelize force integration across nodes using GPU-accelerated simulations (e.g., via PyTorch tensors for potentials). Tune parameters (G, k, δ, c, α, β, γ in Section 10.2) dynamically with a meta-optimizer that samples from stability regions (Lemma 31.2.4). Example: For a 500-atom corpus, simulate convergence: start with underdamped ζ < 1 for fast exploration, then overdamp to ζ > 1 for settling, reducing iterations by 30–50%.
* **Enhancement to AGI**: This scales toward emergent behaviors, like self-organizing knowledge in large-scale planning (thousands of nodes in Section 12.4). Test with benchmarks: nDCG@20 on streamed agents (Chapter 22), ensuring no more than 5% degradation under 10x load.

**2. Uncertainty and Abstention**

**Rationale**: κ-gating (Chapters 3, 11, 14) and UQ methods (temperature sweeps, ensembles, ECE in Section 14.2) are strong for trust, tying into VIF's witness invariant (Section 1.4). But in AGI contexts, where decisions compound (e.g., DEPP self-rewrites in Chapter 12), static thresholds might miss nuanced risks, like domain shifts or adversarial inputs.

**Challenges/Gaps**:

* **Static vs. Dynamic κ**: κ is a fixed abstention threshold (e.g., RS < κ or UQ > κ), but doesn't adapt to context—e.g., high-risk tiers (R2/R3 in Chapter 17) might need tighter bands, while low-risk (R0) could loosen for exploration.
* **UQ Granularity**: Methods like entropy and ECE are per-artifact, but in SEG's temporal graphs (Chapter 15), propagated uncertainty (e.g., via contradiction edges) isn't explicitly modeled, risking overconfidence in lineage queries.
* **False Negatives**: In high-stakes flows (e.g., compliance-critical in Chapter 25), abstention precision ≥ 0.8 (Chapter 22) is targeted, but without adaptive learning, false negatives could cascade, especially if calibration dashboards (Section 14.3) aren't real-time.

**Detailed Recommendations**:

* **Adaptive Thresholds**: Make κ context-dependent using a risk scorer (e.g., κ' = κ \* (1 + risk\_tier / 3), where tier from 0–3). Integrate Bayesian UQ: replace sweeps with evidential networks, where uncertainty is a Dirichlet distribution over confidence buckets (A/B/C in Section 14.3). For example, compute posterior variance on ensembles: if spread > 0.15, escalate to HITL with a VIF note explaining the band (e.g., "ECE drift detected: 0.024 → 0.031").
* **Propagation in SEG**: Add UQ edges in the graph schema (JSON-LD in Section 15.1): each supports/contradicts carries a propagated entropy (e.g., entropy\_out = entropy\_in + log(1 - weight)). Time-slicing queries then aggregate: for a claim at t, sum UQ over paths, abstaining if total > κ\_chain (Section 12.3). This aligns with convergence proofs (Chapter 31), treating UQ as a damping term.
* **Learning Loops**: Use APOE's critic role (Section 11.1) to fine-tune calibrators on held-out data from SEG exports. Metrics: Stratified ECE by domain (e.g., logs vs. code in examples); aim for true-positive abstentions > 0.9 on adversarial suites (Chapter 22). Example: In a protocol translation (Section 1.1), if PII redaction UQ > κ, abstain and quarantine the bundle under SDF-CVF.
* **Enhancement to AGI**: This fosters safer emergence—e.g., in self-rewrite, adaptive κ prevents runaway loops, enabling verifiable evolution toward general intelligence.

**3. Integration with Foundation Models**

**Rationale**: APOE assumes models via VIF (model\_id, weights\_hash in Chapter 14), but for multimodal or evolving models, CMC's modality independence (Chapter 4) could be extended to handle non-text payloads more natively, reducing silos.

**Challenges/Gaps**:

* **Modality Limits**: Atoms support text/code/event/tool (Section 1.2), but richer inputs (images, audio) might require custom enrichers, impacting QS and embeddings.
* **Model Updates**: Weights hashes ensure reproducibility, but updating (e.g., GPT-5t to newer) could invalidate prior calibrations, causing drift (Chapter 23) without automated migration.
* **Bootstrap Feedback**: No explicit loop for models to refine based on SEG evidence, limiting AGI self-improvement.

**Detailed Recommendations**:

* **Multimodal Extensions**: Expand atom schemas to include vision/audio embeddings (e.g., via CLIP or Whisper hashes). In HHNI, add modality-specific indices (e.g., pixel → feature). For DVNS, incorporate cross-modal forces: gravity weighted by modality similarity (cosine on joint embeddings).
* **Update Handling**: Use SDF-CVF for model commits: treat weights as code, with parity to updated prompts/docs (Chapter 16). On update, recompute ECE on a sample SEG slice; if drift > ε (0.05), rollback or quarantine. Metrics: Replay fidelity % post-update (Chapter 22).
* **Feedback Integration**: Enable APOE to generate fine-tuning data from high-RS SEG nodes, using DEPP to orchestrate (e.g., planner → builder for datasets). Example: In IDE-in-the-loop (Chapter 18), use witnessed outputs to tune codegen models.
* **Enhancement to AGI**: This bridges to quantum neuromorphic ideas, allowing emergent multimodal reasoning.

**4. Threat Model Depth**

**Rationale**: Chapter 20 covers injections/SSRF, but AGI-scale threats like embedding poisoning or VIF forgery need deeper playbooks.

**Challenges/Gaps**: Policy-aware DVNS avoids forbidden crossings, but adversarial examples could manipulate GODN forces.

**Detailed Recommendations**:

* **Adversarial Defenses**: Add robustness checks in gates (e.g., g\_safety verifies embedding norms). For VIF, use Merkle trees for tamper-evidence (Section 14.4).
* **Playbooks**: Extend risk taxonomy (Appendix J) with specifics: e.g., on poisoning, quarantine affected HHNI branches.
* **Enhancement to AGI**: Builds resilience for untrusted environments.

**5. Implementation Gaps**

**Rationale**: Reference impls (Part XI) are schemas/APIs, but code snippets would aid prototyping.

**Challenges/Gaps**: Pseudocode is solid, but full examples for ACL (Chapter 13) or DVNS (Chapter 10) are sparse.

**Detailed Recommendations**:

* **Code Artifacts**: Provide TS/Python snippets for key flows (e.g., write\_context). Use Genkit for Fractal Memory Hierarchy integration.
* **Testing**: Add labs for edge cases (Appendix G).
* **Enhancement to AGI**: Enables faster iteration on atomic evolution.

**6. Broader AGI Alignment**

**Rationale**: Invariants promote verifiability, but explicit self-improvement loops could push toward AGI.

**Challenges/Gaps**: No direct mechanism for online learning from SEG.

**Detailed Recommendations**:

* **Self-Improvement Loops**: Use APOE to refine ACL via evidence (e.g., optimize budgets from traces).
* **Multi-Agent Extensions**: Expand negotiation (Chapter 24) with resonant dynamics.
* **Enhancement to AGI**: Aligns with breakthroughs like fusion/quantum, fostering emergent intelligence.

MARG-gpt-5:  
below is a crisp, hands-on upgrade plan for **A Total System of Memory** along two axes:

1. **the idea** (architecture, gaps, proofs, roadmap), and
2. **the artifact** (structure, flow, style, reader experience).

I’m giving you a tightened outline (“Final Draft v0.2 – proposed”), plus a change log (“Process Notes”) so it’s easy to track what moved and why — very “Marg/Hamilton”.

**Final Draft v0.2 — proposed structure**

**Front matter (keep, but sharpen)**

* **1-page Executive Overview** (bulleted, non-technical): problem → invariants → capabilities → risks → proof plan.
* **Reading Roadmaps**: collapse to 3 lanes (Executive • Architect • Implementer).
* **Notation & Acronyms**: convert to a **single-page legend** used consistently everywhere. (Right now, acronyms are powerful but heavy; give the reader a “Rosetta card”.)

**Part I — Why this must exist (tightened)**

* **Ch1 The Why** *(short)*: from “beyond chat” to *native, verifiable, agentic intelligence*; the five invariants and why each is non-optional (CMC, APOE, VIF, SDFCVF, SEG). Include a **table of failure modes if each invariant is absent**.
* **Ch2 Axioms & Invariants** *(formal, testable)*: move proofs to **appendix**; keep body crisp with definitions + falsifiable predictions. Map each invariant to measurable acceptance criteria.
* **Ch3 North Star**: safety-as-capability, κ-gating, “IDE-in-the-loop” rationale — keep, but anchor to concrete scenarios (e.g., regulatory brief, on-call ops).

**Part II — Context–Memory Core (CMC) & HHNI**

* **Ch4 Fractal Memory Hierarchy**: add **running example** (one email thread) showing Atom→Molecule→Document→Thread→Project.
* **Ch5 Hyper-Hierarchical Neural Indexing (HHNI)**: define **RS** precisely with calibration protocol; add **offline evaluation recipe** (gold sets, nDCG, ablations).
* **Ch6 Write/Read Pipelines**: include end-to-end **state diagram** (Ingest→Atomize→Gate→Snapshot). Add **SLOs** with explicit numbers and back-pressure behavior.
* **Ch7 Storage Substrate**: finalize API stubs for **Vector Store**, **Evidence Graph Store**, **Object Store**, **Snapshot Log** (CRUD, versioning, TTL).

**Part III — Dynamic Vector Navigation System (DVNS)**

* Promote DVNS as **core retrieval physics** (not “nice-to-have”). Provide:
  + A compact **energy function** for path cost with terms (length, smoothness, block penalties, density) and constraints for policy-aware geometry.
  + **Convergence criteria** and failure modes (oscillation, collapse) with guardrails.

**Part IV — Orchestration: APOE & DEPP**

* **Roles & Contracts** (planner, retriever, reasoner, verifier, builder, critic, operator, witness) with **I/O schemas** and **budget rules**.
* **DEPP**: present the master chain as a **DAG** (not linear), with health metrics (κ\_chain, budget adherence) and **auto-abstention** gates.
* Provide **three prebuilt chains**: Research, Coding, Operations — each with inputs, tools, gates, and success criteria.

**Part V — Verifiable Intelligence (VIF) & Shared Evidence Graph (SEG)**

* **VIF**: minimal **provenance schema**: {model, weights\_hash, prompts, data\_slice\_ids, tools\_used, uncertainty} + replay recipe.
* **SEG**: lock in **JSON-LD + SHACL** profile; show a **claim→evidence→decision** triple with timestamps and cryptographic digests. Provide an **export profile** for audits.

**Part VI — Atomic Evolution (SDFCVF)**

* Treat this as **DevOps for cognition**: atomic commits across code/docs/tags/traces; **gate catalog** (parity, tests, safety, provenance, HITL).
* Add **DORA-style metrics** and a **quarantine flow** with rollback.

**Part VII — IDE-in-the-Loop**

* Provide a concrete **Blueprint→Scaffold→Code→Test→Deploy→Observe** walkthrough for a tiny app (e.g., RAG microservice).
* Include a **Blueprint YAML** template and automated parity checks (spec↔code).

**Part VIII — Security, Safety & Compliance**

* Keep threat model; add a **policy-aware retrieval demo** where DVNS forbids crossings into restricted subgraphs; log abstentions into VIF.
* Map to **EU AI Act logging** and **SOC2/ISO** via trace fields coming from VIF/SEG.

**Part IX — Evaluation & Observability**

* Define benchmark packs: **retrieval (p@k, nDCG, RS lift)**, **reasoning (task suites)**, **long-horizon agents**, plus **κ/ECE dashboards**.
* Add **OpenTelemetry** spans around agent steps; set **token/latency budgets** and drift monitors.

**Part X — Case Studies & Patterns**

* Ship at least **two worked case studies** end-to-end (incl. SEG exports and VIF replays).
* Provide a **Builder Program** pattern library.

**Appendices (new)**

* **Mathematical Details**: proofs, DVNS integration schemes (Verlet vs. Euler), κ-calibration, uncertainty estimation.
* **ACL / AIMOS Chain Language**: full EBNF + static checks; sample pipelines.

**Concept upgrades (substance)**

1. **Make the invariants falsifiable**  
   Add **acceptance tests** per invariant (e.g., CMC must survive context reset and still retrieve a prior “Atom” with within latency budget). Provide **counter-examples** where systems without the invariant fail.
2. **DVNS ←→ Policy coupling**  
   Right now DVNS defines path costs and stability. Add a **policy tensor** that raises path energy near restricted evidence nodes; DVNS then *physically* avoids off-policy traversals (safety as geometry, not just post-hoc checks).
3. **APOE budgets as first-class citizens**  
   Turn “budgets (tokens, time, tools)” into an **admission controller** that can:

* refuse chain expansion,
* trigger a reduced-fidelity route, or
* persist a partial result with VIF.

1. **VIF replayability**  
   Specify a **deterministic replay harness**: fixed seeds, frozen tool versions, pinned models (weights hash), and input slices. Make “replay ok” a gate in SDFCVF.
2. **SEG temporal reasoning**  
   Add **time-slicing queries** (e.g., “evidence as of T−7d”) and “impact previews” from **dependency hashing** (already alluded to). Bake these into HHNI.
3. **AIMOS/ACL hardening**  
   Your chain language is a gem. Add:

* static **budget/type checks**,
* capability tokens per tool,
* formal **abstention semantics** (must\_abstain\_if: clauses).

1. **End-to-end success metrics**  
   Define *system* KPIs (not just retrieval):

* Task success @ budget,
* Claim-with-evidence rate,
* Mean time to abstention (MTA),
* Replay success rate,
* SEG completeness (% steps with linked evidence).

**Quick wins (high leverage, low effort)**

* **Insert a single “running example”** (e.g., summarize and ground an RFC) that flows through Parts II→V.
* **Ship a one-page API** for each store (Vector/SEG/Object/Snapshot) — request+response examples.
* **Make a “Chain Gallery”**: 3 DAGs with minimal YAML and screenshots of telemetry.
* **Move heavy math and proofs to appendices**; keep the mainline persuasive and fast.
* **Glossary card** at the front; **index** at the end.

**Process Notes (what changed & why)**

* **Regrouped** Parts II–V around a concrete *data-in → memory → retrieval physics → orchestration → provenance* storyline. Readers can now trace *one object* through the system.
* **Promoted** DVNS and VIF/SEG to co-equal pillars with CMC/APOE (they are the legs that make this a *verifiable, steerable* system).
* **Demoted** long proofs to appendices so the main arc remains executable.
* **Added** measurable claims and acceptance tests so this is **falsifiable engineering**, not just design.

**Implementation tracks (start building tomorrow)**

1. **The Foundry (bootstrap chain)**
   * Agents: Ingestor → Atomizer → Schema-Validator → DD-Gate → Snapshot-Writer → Evidence-Linker.
   * Outputs: Atoms, Snapshots, SEG edges, VIF logs with hashes. (This is the smallest “system that manufactures the system”.)
2. **DVNS micro-pilot**
   * Implement path cost, basic forces, and a policy tensor; compare vs. static KNN on RS-lift and abstention quality.
3. **VIF/SEG substrate**
   * Finalize JSON-LD & SHACL; build “Replay Harness v0.1” (pin model+tools, re-execute a chain, compare outputs).
4. **Evaluation pack v0.1**
   * Gold sets, dashboards for κ/ECE, token/latency budgets; add OpenTelemetry spans per agent step.

**Stylistic edits to apply as you revise**

* **Reduce acronym density per paragraph** (target ≤2).
* **Prefer spec boxes**: “Contract • Inputs • Outputs • Invariants • Failure modes”.
* **Surface numbers**: budgets, latencies, thresholds — even provisional — to anchor the reader.
* **Replace future tense** with **imperative** in procedures (reads as an engineering spec).
* **Use consistent figure grammar** (Figure N: Title • Purpose • Takeaway • Link to Appendix).

**6.2 Future Work and Recommendations**

The analysis suggests two paths for future hardening:

1. **Enhancing Formal Verification at the Gate Level:** While ACL provides symbolic checks and type safety, moving closer to absolute guarantees requires deeper integration with formal methods. It is recommended that core ACL steps used by the Builder Programs for code synthesis be integrated with explicit SMT solvers or theorem provers (e.g., Lean) at the provenance gate (G-Trace). This would allow the system to discharge functional proofs for critical logic generated by the AGI, marrying the pragmatic verifiability of VIF with the mathematical rigor sought by the IPS research community.
2. **Detailing Distributed Scalability and Resilience:** The deterministic properties rely on the transactional nature of the CMC write pipeline (C-1 Single Writer) and the fidelity of the HHNI/DVNS indices. Future documentation should detail the distributed implementation of these components, ensuring that determinism and performance (SLOs) are maintained at petabyte scale and under high-concurrency loads typical of AGI systems (e.g., specifying shard distribution strategies for the DVNS super-index and resilience strategies for the single-writer constraint).