Project Chimera
Specification and Context Engineering Report

Name
Nurye Nigus

Email: nurye.nigus.me@gmail.com

February 5, 2026

Table of content

Table of Figures

1. Purpose of This Report

This report documents the **Specification and Context Engineering phase** of Project Chimera. It explains the rationale, structure, and governance mechanisms that were established before any implementation work begins.

Project Chimera is an **autonomous influencer network** designed to operate a fleet of persistent, goal-driven agents. These agents plan, create, review, and publish content while adhering to explicit safety, auditability, and governance constraints.

The purpose of this phase is not to describe how the system will be coded, but to define **what the system is allowed to do**, **how decisions are made**, and **how correctness is verified**.

This submission fulfills the **Day 2 challenge requirements**.
No production code, tests, containers, or CI pipelines were created.
All decisions remain reversible and are encoded as specifications.

The primary outcome of this phase is to ensure that future agent-driven implementation can proceed **without ambiguity**, **without hidden assumptions**, and **without uncontrolled agent behavior**.

2. Specification Strategy

2.1 Why Spec-Driven Development is Mandatory

Agent-based systems differ from traditional software in one critical way: they reason.
When reasoning systems encounter ambiguity, they invent solutions. This leads to hallucinated logic, silent scope expansion, and governance failure.

To avoid this, Project Chimera adopts **Spec-Driven Development (SDD)** as a non-negotiable rule.

In this model:

- Specifications define intent before execution
- Agents are not trusted to infer missing requirements
- Behavior is constrained by contracts, not prompts
- Correctness is defined before implementation exists

This ensures that the system remains predictable even as autonomy increases.

2.2 Structure and Role of the Specification Set

The specification layer is composed of four tightly coordinated documents. Each document eliminates a specific class of ambiguity.

specs/_meta.md
This document functions as the **constitutional layer** of the system. It defines scope, non-negotiable constraints, success criteria, and known risks. It explicitly states what is out of scope to prevent premature complexity.

This file ensures:
- Safety and traceability are mandatory
- MCP is enforced as a hard boundary
- Multi-tenancy readiness is considered early
- Risk awareness is embedded before implementation

specs/functional.md
This document defines **what the system does**, expressed through actor-based workflows. Each workflow includes acceptance criteria written to be verifiable.

This document ensures:
- Business intent is translated into system behavior
- Review and escalation paths are explicit
- Each requirement can later become a failing test
- Nothing “implicit” is allowed

specs/technical.md
This document defines **how the system is structured**, without writing code. It formalizes architecture boundaries, service roles, message contracts, data models, and non-functional requirements.

This document ensures:
- Planner, Worker, Judge roles are explicit
- Data contracts are stable before implementation
- High-velocity ingestion is addressed deliberately
- Observability and audit hooks are mandatory

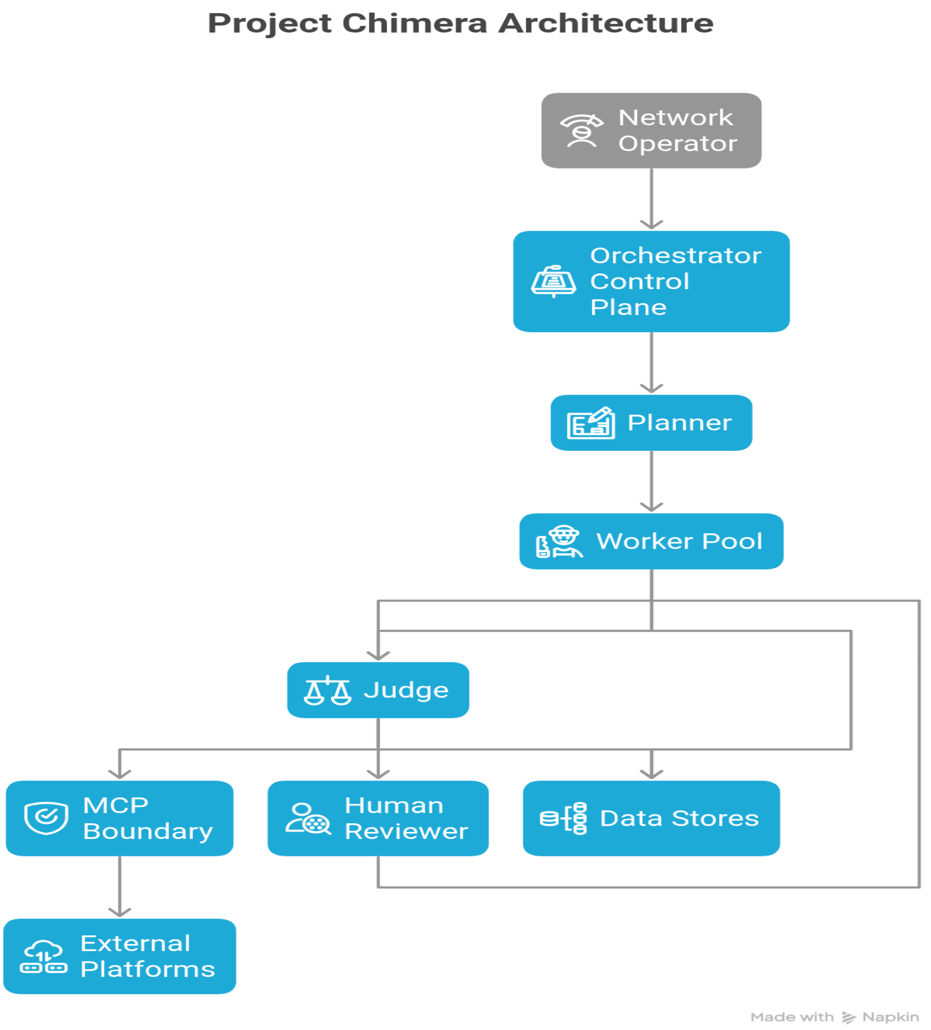
specs/context_engine.md
This document defines the **Context Engine**, which solves a critical agent-system failure mode: inconsistent context.

Without a context layer, agents assemble their own understanding, leading to divergent decisions and weak traceability. The Context Engine centralizes, versions, and audits task-scoped context.

This document ensures:
- Workers and Judges operate on the same snapshot
- Context freshness and TTL are controlled
- Sensitivity tagging drives HITL routing
- MCP-only inputs reduce injection risk

2.3 High-Level System Architecture

Figure 1 illustrates the overall architecture of Project Chimera, emphasizing separation of concerns, governance points, and integration boundaries.



This architecture demonstrates that:
- Planning, execution, and review are separated
- Human review is structurally enforced
- External systems are never accessed directly
- Governance is embedded, not added later

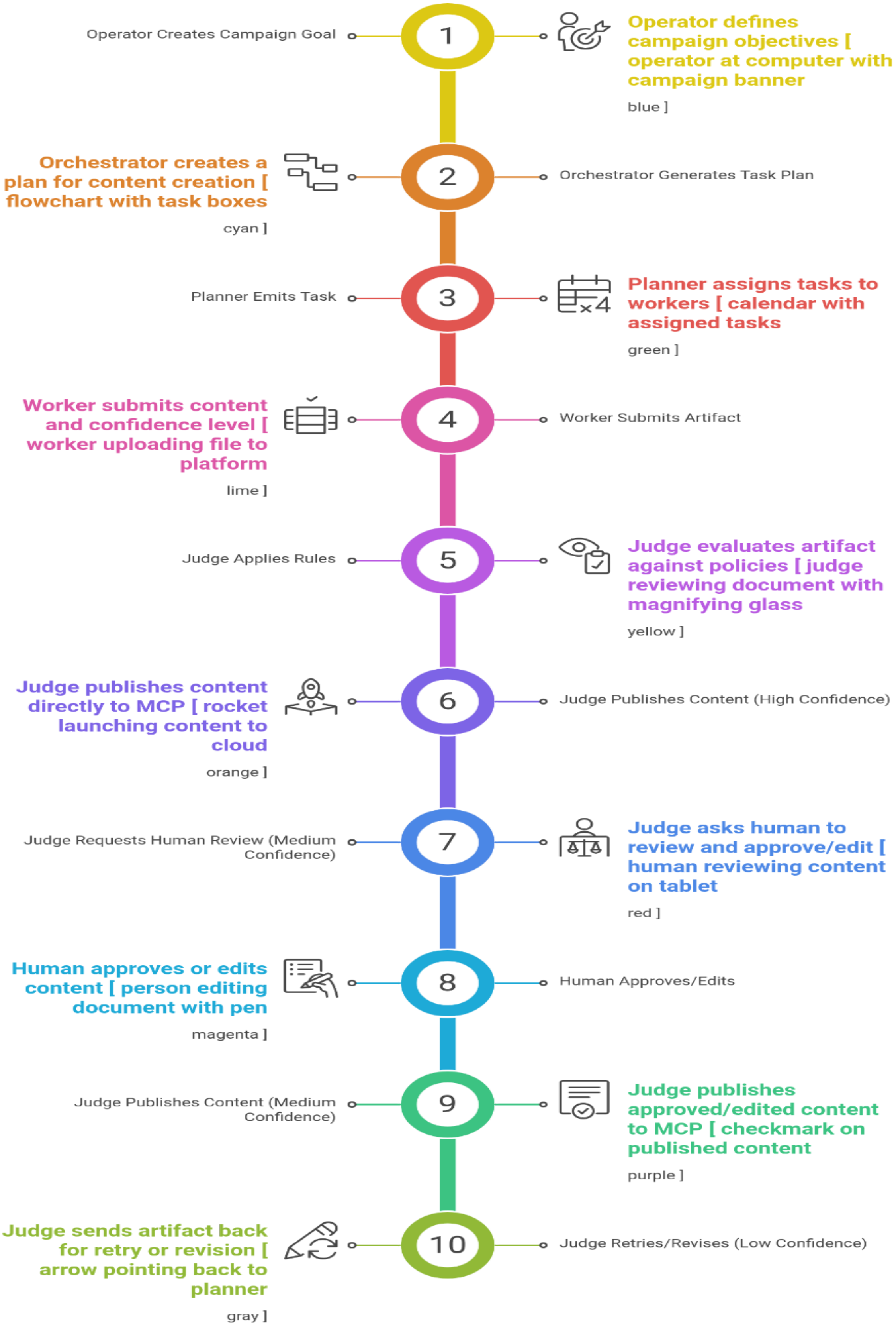
3. Campaign Execution and Decision Flow

Campaign execution follows a structured decision pipeline. No content reaches publication without passing through explicit review gates.

Figure 2 illustrates this flow.

This flow ensures:
- Confidence scores drive routing
- Sensitive content cannot bypass humans
- Retries are intentional, not accidental
- Publish actions are always auditable

Campaign Content Creation and Approval Workflow



1 Operator Creates Campaign Goal

Operator defines campaign objectives [operator at computer with campaign banner

blue]

2 Orchestrator Generates Task Plan

Orchestrator creates a plan for content creation [flowchart with task boxes

cyan]

3 Planner Emits Task

Planner assigns tasks to workers [calendar with assigned tasks

green]

4 Worker Submits Artifact

Worker submits content and confidence level [worker uploading file to platform

lime]

5 Judge Applies Rules

Judge evaluates artifact against policies [judge reviewing document with magnifying glass

yellow]

6 Judge Publishes Content (High Confidence)

Judge publishes content directly to MCP [rocket launching content to cloud

orange]

7 Judge Requests Human Review (Medium Confidence)

Judge asks human to review and approve/edit [human reviewing content on tablet

red]

8 Human Approves/Edits

Human approves or edits content [person editing document with pen

magenta]

9 Judge Publishes Content (Medium Confidence)

Judge publishes approved/edited content to MCP [checkmark on published content

purple]

10 Judge Retries/Revises (Low Confidence)

Judge sends artifact back for retry or revision [arrow pointing back to planner

gray]

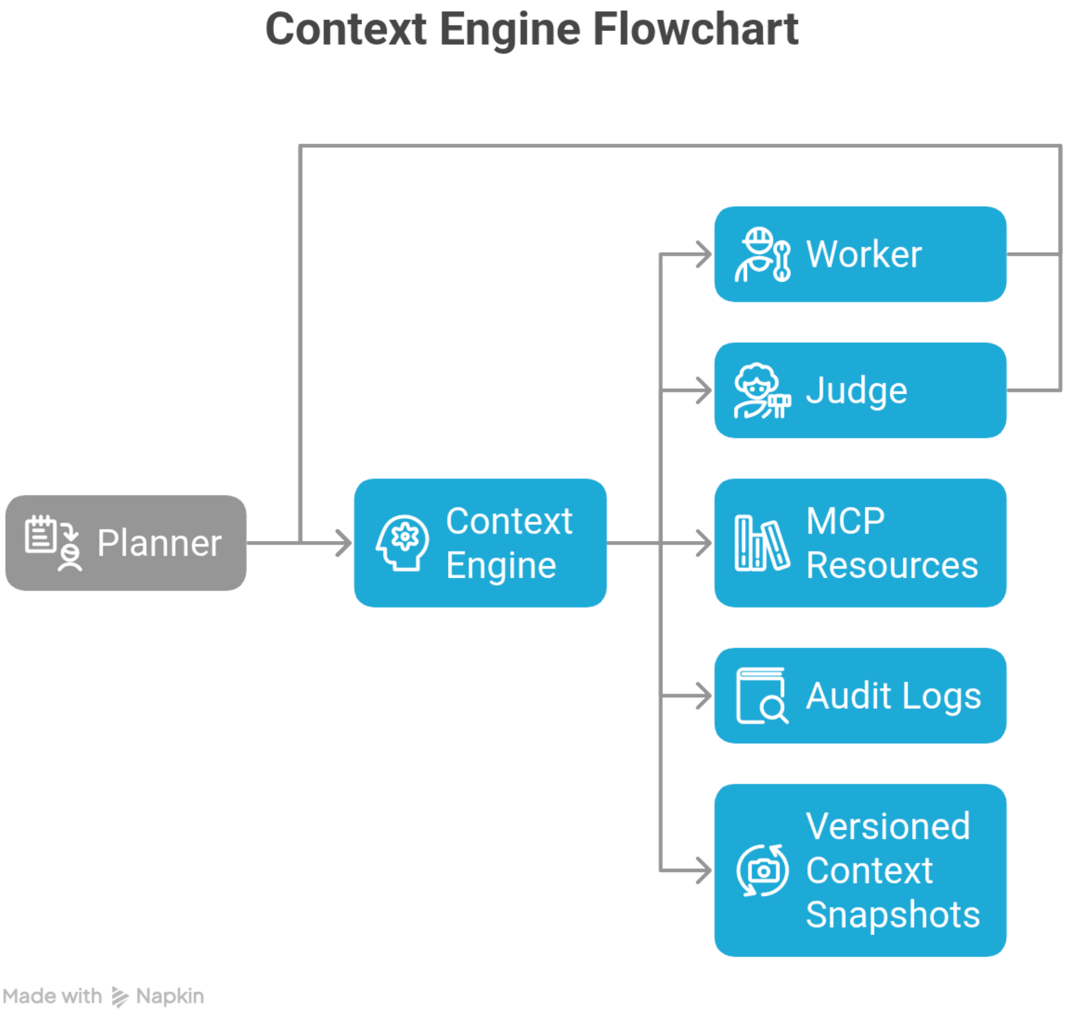
4. Context Engineering and Agent Governance

4.1 Why Context Engineering is Required

In autonomous systems, context is as important as logic. If two agents reason with different context, outputs diverge even if logic is correct.

Project Chimera addresses this with a **Context Engine** that normalizes task-scoped context into immutable, versioned snapshots.

Figure 3 illustrates context flow.



This ensures:
- Consistent reasoning across agents
- Controlled freshness for trend signals
- Explicit audit trails
- Reduced attack surface via MCP

4.2 IDE-Level Context Enforcement

Agent governance is enforced not only at runtime, but also at development time.

The VS Code instruction file enforces:
- Mandatory spec reading
- Plan-first reasoning
- No implementation before ratification
- MCP-only external access
- HITL routing for sensitive domains

This ensures that even IDE assistance behaves predictably and remains aligned with system governance.

5. Tooling and Skills Strategy

5.1 Separation of Developer Tools and Runtime Skills

Project Chimera explicitly separates **developer MCP tooling** from **runtime agent skills**.

Developer tools exist to:
- Inspect files
- Capture logs
- Assist documentation
- Support traceability

They are never part of the runtime system.

Runtime skills are:
- Stable
- Contract-defined
- MCP-mediated
- Auditable

This separation prevents accidental coupling and improves long-term maintainability.

5.2 Runtime Skills

Three core skills are defined structurally:

skill_trend_fetcher
Normalizes trend signals from MCP resources. Produces relevance-scored items with source links.

skill_video_metadata_ingestor
Handles high-frequency metric ingestion using append-only semantics. Returns acknowledgements without blocking agents.

skill_publish_content
Publishes content via MCP tools. Returns platform identifiers, timestamps, and audit linkage.

Each skill is designed to be testable, replaceable, and governed.

6. Data and High-Velocity Metadata Strategy

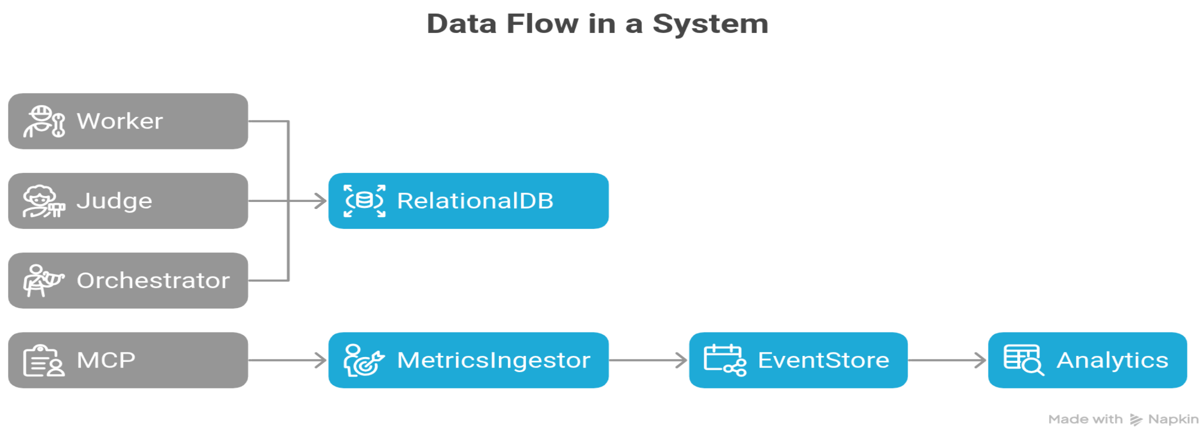
6.1 Data Class Separation

Two distinct data classes are recognized:

Control data
Campaigns, tasks, artifacts, decisions. Requires strong consistency and lineage.

Performance metadata
High-frequency video and social metrics. Requires write throughput and time-based aggregation.

Figure 4 illustrates this separation.

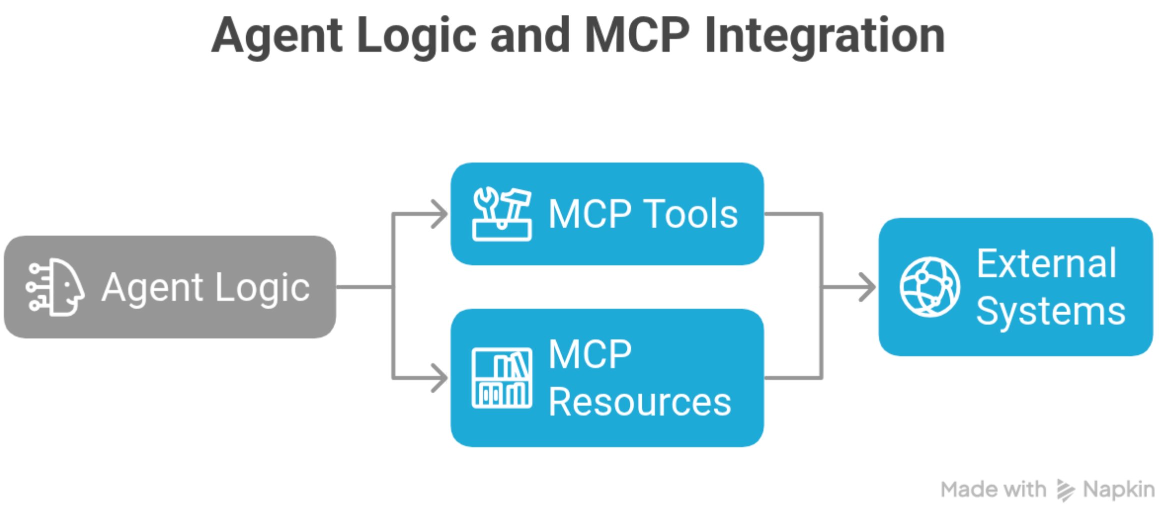


This directly addresses the challenge requirement for **high-velocity video metadata storage** while preserving governance.

7. MCP Boundary and External Integration

MCP is enforced as a hard boundary.

Figure 5 illustrates this constraint.



This ensures:

- Replaceable integrations
- Full observability
- Consistent audit trails
- Reduced platform lock-in

8. Environment and Traceability

The repository is initialized with:

- Professional Python environment using uv
- MCP Sense connected and authenticated
- Structured artifact logging
- Clean spec-first repo layout

These choices support traceability, reproducibility, and grading verification.

9. What Was Intentionally Deferred

This phase intentionally excludes:
- Implementation code
- Tests
- Docker
- CI pipelines

Deferral prevents premature architectural lock-in and preserves flexibility.

10. Readiness for Next Phase

The project is now ready for:
- Test-Driven Development
- Failing test definition
- Containerization
- CI and governance automation

All future work will reference the ratified specifications.

11. Conclusion

This report establishes a **complete, governed, and auditable foundation** for Project Chimera. The work performed in this phase does not merely outline an idea; it defines a system that can be implemented by autonomous agents **without ambiguity, drift, or unsafe behavior**.

Through Spec-Driven Development, the project translates business intent into precise, verifiable specifications. System behavior, boundaries, and non-functional requirements are explicitly defined before implementation exists. This eliminates reliance on assumptions and prevents agents from inventing missing logic.

The architectural strategy demonstrates deliberate separation of concerns. Planning, execution, review, and publishing are structurally isolated, while Human-in-the-Loop review is embedded as a first-class control mechanism rather than an afterthought. The Context Engine further strengthens governance by ensuring that all agents reason over consistent, versioned, and auditable context snapshots.

The tooling and skills strategy reinforces long-term maintainability. Developer tooling is clearly separated from runtime capabilities, preventing accidental coupling. Runtime skills are defined through stable contracts, enabling replacement, testing, and controlled evolution without system-wide refactoring.

The data strategy directly addresses the challenge of high-velocity video and social metadata ingestion. By separating governance-critical data from append-heavy performance metrics, the system supports scale without sacrificing traceability, correctness, or operational control.

Finally, the MCP boundary enforces disciplined external integration. By requiring all platform interactions to pass through standardized tools and resources, the system remains observable, auditable, and resilient to platform change.

As a result, Project Chimera is now prepared for **controlled, test-driven implementation**. The specifications define not only what should be built, but how correctness will be measured. Future work can proceed with confidence, knowing that autonomy is constrained by governance, safety is enforced by design, and system behavior is fully traceable.