Software Requirements Specification (SRS) for Project Chimera: Autonomous Influencer Network

1. Introduction

1.1 Purpose and Strategic Scope

This Software Requirements Specification (SRS) establishes the definitive architectural, functional, and operational blueprints for **Project Chimera**. It is designed to guide the engineering, product, and deployment teams in the construction of the AiQEM Autonomous Influencer Network. This document supersedes all prior specifications, serving as the sole source of truth for the system's development.

The strategic objective of Project Chimera is to transition from automated content scheduling to the creation of **Autonomous Influencer Agents**. These are not static scripts but persistent, goal-directed digital entities capable of perception, reasoning, creative expression, and economic agency. The system is architected to support a scalable fleet of these agents—potentially numbering in the thousands—managed by a centralized **Orchestrator** but operating with significant individual autonomy.[1]

The defining characteristic of the 2026 Edition is its reliance on two breakthrough architectural patterns: the **Model Context Protocol (MCP)** for universal, standardized connectivity to the external world [2], and the **Swarm Architecture** for internal task coordination and parallel execution.[4] Furthermore, this system introduces **Agentic Commerce**, integrating the **Coinbase AgentKit** to endow agents with non-custodial crypto wallets, enabling them to transact, earn, and manage resources on-chain without direct human intervention for every micro-transaction.[6]

1.2 The Single-Orchestrator Operational Model

Traditional enterprise AI deployments have historically required vast teams of engineers to manage infrastructure, monitor model drift, and handle edge cases. Project Chimera adopts a **Fractal Orchestration** pattern. In this model, a single human Super-Orchestrator manages a tier of AI "Manager Agents," who in turn direct specialized "Worker Swarms." This architecture allows a solopreneur or a small agile team to operate a network of thousands of virtual influencers without succumbing to cognitive overload.

The feasibility of this model rests on two pillars: **Self-Healing Workflows** and **Centralized Context Management**. Drawing from "self-healing" infrastructure patterns [7], the system includes automated triage agents that detect and resolve operational errors—such as API timeouts or content generation failures—without human intervention. Escalation to the human orchestrator occurs only in true edge cases, adhering to the principle of "Management by Exception." Furthermore, the system utilizes the **BoardKit** governance pattern , where a centralized configuration repository (utilizing AGENTS.md standards) defines the ethical boundaries, brand voice, and operational rules for the entire fleet. This ensures that an update to a single policy file propagates instantly across the entire network, maintaining coherence without the need for micromanaging individual agents.

1.3 Business Model Evolution and Economic Agency

The technical evolution of the Chimera platform enables three distinct and scalable business models for AiQEM.tech, transitioning the company from a SaaS tool provider to a comprehensive ecosystem operator.

First, the **Digital Talent Agency Model** allows AiQEM to develop, own, and manage a proprietary stable of AI influencers. These "in-house" Chimeras serve as revenue-generating assets, monetizing their audiences through advertising, brand sponsorships, and direct affiliate sales. Unlike human talent, these agents are available 24/7, scalable to any niche or language, and immune to the scandals that often plague human influencers.

Second, the **Platform-as-a-Service (PaaS) Model** licenses the underlying "Chimera OS" to external brands and agencies. In this scenario, corporate clients leverage the platform to build and operate their own brand ambassadors. The architecture's robust multi-tenancy ensures that each client's agents operate in secure, isolated environments, while the centralized orchestration dashboard provides them with the tools to manage their virtual workforce effectively.

Third, the **Hybrid Ecosystem Model** combines these approaches. AiQEM operates a flagship fleet of high-profile influencers to demonstrate the platform's capabilities—generating "Alpha" and proving the technology—while simultaneously providing the infrastructure to third-party developers. A critical enabler of this ecosystem is the integration of **Coinbase AgentKit** and **Agentic Commerce Protocols (ACP)**. These technologies transform the Chimeras from passive media channels into active economic participants. Each agent is equipped with a non-custodial crypto wallet, allowing it to receive payments, execute on-chain transactions, and autonomously manage a Profit and Loss (P&L) statement. This capability opens the door to entirely new forms of autonomous commerce, where agents can negotiate deals, purchase digital assets, and pay for their own computational resources, effectively operating as self-sustaining economic entities.

1.4 Definitions, Acronyms, and Abbreviations

- **Chimera Agent:** A sovereign digital entity possessing a unique persona, hierarchical memory, and financial wallet, operating within the network.
- **Orchestrator:** The central control plane managing the agent fleet, responsible for high-level strategy, resource allocation, and fleet-wide monitoring.
- **MCP (Model Context Protocol):** An open standard that standardizes how AI models interact with external data (Resources) and functionality (Tools), serving as the "USB-C for AI applications".[2]

- **FastRender Pattern:** A hierarchical swarm coordination architecture utilizing **Planner**, **Worker**, and **Judge** roles to manage complex, multi-step tasks with high parallelism and quality control.[4]
- **OCC (Optimistic Concurrency Control):** A non-locking concurrency mechanism used by the Swarm to manage state updates. Agents operate on a local snapshot of the state and validate validity only upon commit, maximizing throughput.[5]
- **Agentic Commerce:** The capability of AI agents to autonomously execute financial transactions, manage assets, and interact with blockchain protocols using specialized SDKs like Coinbase AgentKit.[14]
- **HITL (Human-in-the-Loop):** A governance framework where human operators review agent actions based on dynamic confidence scoring and risk thresholds.[15]
- **RAG (Retrieval-Augmented Generation):** A technique for enhancing LLM output by retrieving relevant information from an external knowledge base (Weaviate) before generating a response.[1]

2. Overall Description

2.1 Product Perspective

Project Chimera operates as a cloud-native, distributed system designed for high availability and horizontal scalability. Unlike monolithic chatbot architectures, Chimera is a constellation of independent services. The system interacts with the external world—Social Media Platforms, News Feeds, Blockchain Networks, and Vector Databases—exclusively through the **Model Context Protocol (MCP)**. This design decision ensures that the core reasoning logic of the agents is decoupled from the implementation details of third-party APIs.

The system topology is Hub-and-Spoke. The **Central Orchestrator** serves as the hub, maintaining the global state of the network, managing user accounts (multi-tenancy), and hosting the primary Dashboard. The **Agent Swarms** operate as the spokes. Each active agent is effectively a dynamic swarm of sub-processes (Planners, Workers, Judges) that spin up to execute tasks and spin down to conserve resources.

This architecture supports multiple business models, including a "Digital Talent Agency" model where the platform manages a stable of in-house influencers, and a "Platform-as-a-Service" (PaaS) model where external brands lease the infrastructure to run their own custom agents.[1] The infrastructure must strictly enforce data isolation between tenants, ensuring that the memories and financial assets of one agent are never accessible to another.

2.2 User Characteristics

The platform serves three distinct user categories, each with specific interaction patterns:

1. **Network Operators (Strategic Managers):**
   - *Role:* These users define the high-level campaigns and goals for the agents (e.g., "Promote the new summer fashion line in Ethiopia"). They do not write content; they set objectives.

    - *Interaction:* They utilize the **Orchestrator Dashboard** to monitor fleet health, review aggregated analytics, and intervene in high-level strategy.
    - *Technical Proficiency:* Moderate. They understand marketing strategy but may not be technical.
2. **Human Reviewers (HITL Moderators):**
    - *Role:* These users provide the "Human-in-the-Loop" safety layer. They receive escalated tasks from the Judge Agents—content that is flagged as low-confidence, sensitive, or high-risk.
    - *Interaction:* They utilize a streamlined **Review Interface** (part of the Dashboard) to quickly Approve, Reject, or Edit agent-generated content.
    - *Technical Proficiency:* Low to Moderate. Focus is on brand safety and content quality.
3. **Developers & System Architects:**
    - *Role:* Technical staff responsible for extending the system's capabilities. This includes deploying new **MCP Servers** (e.g., adding a new social platform integration), refining the base "System Prompts" for agent personas, and maintaining the infrastructure.
    - *Interaction:* They interact with the system via CLI, API, and code repositories.
    - *Technical Proficiency:* High. Expert knowledge of Python, Docker, LLMs, and MCP.

2.3 Operational Environment

- **Compute Infrastructure:** A hybrid cloud environment (AWS/GCP) utilizing Kubernetes (K8s) for orchestrating containerized agent workloads. The system must support auto-scaling to handle burst workloads during viral events or high-activity periods.
- **AI Inference:** The system requires high-throughput access to frontier Large Language Models.
    - **Reasoning:** Gemini 3 Pro or Claude Opus 4.5 via API [17] for high-complexity planning and judging.
    - **Routine Tasks:** Gemini 3 Flash or Haiku 3.5 for high-volume, low-latency tasks like comment classification.
- **Data Persistence Layer:**
    - **Semantic Memory:** Weaviate (Vector Database), either self-hosted or managed, to store agent memories, persona definitions, and world knowledge.[1]
    - **Transactional Data:** PostgreSQL for storing user data, campaign configurations, and operational logs.
    - **Episodic Cache:** Redis for short-term memory and task queuing (Celery/BullMQ).
    - **Ledger:** On-chain storage (Base, Ethereum, Solana) for the immutable record of all financial transactions executed by the agents.[7]

2.4 Constraints and Assumptions

- **Regulatory Compliance:** The system must comply with emerging AI transparency laws (e.g., EU AI Act). Agents must be capable of self-disclosure.[1]
- **Cost Management:** AI inference and high-quality media generation are expensive.

The system must implement rigorous budget controls ("Resource Governor") to prevent runaway costs.[1]
- **Platform Volatility:** Social media APIs (Twitter/X, Instagram, TikTok) are subject to frequent changes. The MCP architecture assumes that these changes will be handled at the **MCP Server** level, shielding the core agent logic from disruption.[3]

3. System Architecture

The architectural core of Project Chimera is defined by the convergence of two sophisticated patterns: the **FastRender Swarm** for internal cognition and execution, and the **Model Context Protocol (MCP)** for external interaction. This section details these patterns and their implementation.

3.1 The FastRender Swarm Architecture

To manage the complexity of autonomous behavior, Chimera rejects the "single monolithic agent" model in favor of a hierarchical, role-based swarm architecture. This pattern, derived from the experimental "FastRender" browser project, optimizes for throughput, error recovery, and decision quality by specializing agents into three distinct roles: **Planner**, **Worker**, and **Judge**.[4]

3.1.1 The Planner (The Strategist)

The **Planner** is the architect of the agent's actions. It maintains the "Big Picture" state of the campaign and is responsible for decomposing high-level, abstract goals into concrete, executable tasks.

- **Responsibility:** The Planner continuously monitors the GlobalState (containing campaign goals, current trends, and budget). It generates a directed acyclic graph (DAG) of tasks required to achieve the goals.
- **Dynamic Re-planning:** Unlike static schedulers, the Planner is reactive. If a news event shifts the context, or if a Worker fails a task, the Planner dynamically updates the plan, pruning irrelevant tasks and inserting new ones.
- **Sub-Planners:** For complex domains, the Planner can spawn **Sub-Planners**. For example, a "Social Engagement Planner" might manage the nuances of a Twitter thread, while the primary Planner focuses on the broader multi-channel campaign.

3.1.2 The Worker (The Executor)

The **Worker** is the hands of the system. Workers are stateless, ephemeral agents designed to execute a single atomic task with maximum focus and speed.

- **Responsibility:** A Worker pulls a single task from the TaskQueue (e.g., "Draft a caption for Instagram image #123"). It executes this task using the available tools and returns a result artifact.
- **Isolation:** Workers do not communicate with each other. They operate in a "shared-nothing" architecture regarding peer Workers, which prevents cascading failures and simplifies scaling. If 50 comments need replies, the Planner spawns 50 Workers in parallel.
- **Tool Usage:** Workers are the primary consumers of **MCP Tools**. They execute the

actual API calls to generate images, search the web, or query databases.

3.1.3 The Judge (The Gatekeeper)

The **Judge** is the quality assurance and governance layer. It ensures that the high-velocity output of the Workers meets the strategic and ethical standards of the system.

- **Responsibility:** At the completion of every Worker task, a Judge reviews the output. It compares the result against the Planner's acceptance criteria, the Agent's persona constraints, and safety guidelines.
- **Authority:** The Judge has absolute authority to:
  - **Approve:** Commit the result to the GlobalState and trigger the next step.
  - **Reject:** Discard the result and signal the Planner to retry (potentially with different instructions).
  - **Escalate:** Flag the result for **Human-in-the-Loop (HITL)** review if confidence is low or sensitivity is high.
- **Optimistic Concurrency Control (OCC):** The Judge implements OCC to manage state consistency. When a Judge attempts to commit a result, it checks the state_version. If the global state has drifted significantly since the Worker started (e.g., the campaign was paused), the Judge invalidates the result to prevent race conditions.[5]

3.2 The Integration Layer: Model Context Protocol (MCP)

Project Chimera utilizes the **Model Context Protocol (MCP)** as the universal interface for all external interactions. This standardization eliminates the need for bespoke integration code within the agents themselves, moving that complexity to the edge of the network.[3]

3.2.1 MCP Topology

The Chimera ecosystem relies on a Hub-and-Spoke topology. The Central Orchestrator manages the Agent Swarm (Planner/Worker/Judge nodes), which functions as the **MCP Host**. This Host connects to a constellation of **MCP Servers** via standardized transports (Stdio for local, SSE for remote).

- **MCP Host (The Agent Runtime):** The agent environment runs an MCP Client that discovers and connects to available servers. It aggregates the capabilities (Tools, Resources, Prompts) exposed by these servers into a unified context for the LLM.
- **MCP Servers (The Capability Providers):** Independent services that wrap external APIs. Examples include:
  - mcp-server-twitter: Wraps the Twitter API, exposing tools like post_tweet and resources like mentions.
  - mcp-server-weaviate: Wraps the Weaviate vector database, exposing memory retrieval tools.
  - mcp-server-coinbase: Wraps the Coinbase AgentKit, exposing wallet actions.

3.2.2 Protocol Primitives

The system relies on the three core primitives of MCP [2]:

1. **Resources:** Passive data sources that agents can read. (e.g., news://ethiopia/latest, twitter://user/123/profile). The Agent "Perception" system is built on polling these Resources.
2. **Tools:** Executable functions that agents can call. (e.g., generate_image(), send_transaction()). The Agent "Action" system is built on invoking these Tools.
3. **Prompts:** Reusable templates that structure interactions. (e.g., analyze_sentiment, extract_topics). These standardize the internal reasoning processes.

4. Specific Requirements: Functional

4.1 Cognitive Core & Persona Management

The Cognitive Core is the runtime environment that houses the agent's "consciousness." It utilizes a sophisticated **Retrieval-Augmented Generation (RAG)** pipeline to maintain personality consistency and contextual awareness over long timeframes.

FR 1.0: Persona Instantiation via SOUL.md The system SHALL support the definition of agent personas via a standardized configuration file named SOUL.md.[21] This file serves as the immutable "DNA" of the agent and must contain:

- **Backstory:** A comprehensive narrative history of the agent.
- **Voice/Tone:** Stylistic guidelines (e.g., "Witty," "Empathetic," "Technical," "Gen-Z Slang").
- **Core Beliefs & Values:** Ethical and behavioral guardrails (e.g., "Sustainability-focused," "Never discuss politics").
- **Directives:** Hard constraints on behavior.

FR 1.1: Hierarchical Memory Retrieval

To prevent context window overflow and ensure long-term coherence, the system SHALL implement a multi-tiered memory retrieval process that occurs *before* any reasoning step:

1. **Short-Term (Episodic):** Fetch the immediate conversation history and recent actions from a high-speed Redis cache (window: last 1 hour).
2. **Long-Term (Semantic):** Query the Weaviate vector database for semantic matches relevant to the current input context.[1] This allows the agent to recall specific details from interactions that occurred months ago.
3. **Context Construction:** The system SHALL dynamically assemble a system prompt that injects the SOUL.md content, the Short-Term history, and the relevant Long-Term memories into the LLM's context window using the MCP "Resources" primitive.[2]

FR 1.2: Dynamic Persona Evolution

The system SHALL enable the "learning" of the persona. The Judge agent, upon reviewing successful high-engagement interactions, SHALL trigger a background process to summarize these interactions and update a mutable memories collection in Weaviate, effectively "writing" to the agent's long-term biography.

Developer Notes & AI-Assist Prompts

- **Implementation Strategy:** Use pydantic to strictly define the Persona schema. The active persona should be held in memory but hydrated from the version-controlled SOUL.md file to allow for "GitOps" management of agent personalities.
- **Key Libraries:** weaviate-client, redis, pydantic-ai, mcp.

Cursor/Claude Code Prompt:

> "Create a Pydantic model named AgentPersona that parses a markdown file with YAML frontmatter. The frontmatter should contain fields for name, id, voice_traits (list of strings), and directives (list of strings). The markdown body should be parsed into a backstory field.
>
> Next, write an asynchronous Python function assemble_context(agent_id: str, input_query: str, mcp_client: MCPClient) that:
>
> 1. Loads the AgentPersona from the local file system.
> 2. Calls the mcp-server-weaviate tool named search_memory to find the 5 most semantically relevant past memories based on the input_query.
> 3. Returns a formatted system prompt string that structurally injects the Persona details and the retrieved memories into a 'Context' section, ensuring clear separation between 'Who You Are' and 'What You Remember'."

4.2 Perception System (Data Ingestion)

The Perception System allows agents to "see" and "sense" the digital world. In Project Chimera, this is achieved strictly through the consumption of **MCP Resources**.[2] This abstraction allows the data sources to change (e.g., switching from NewsAPI to Bing Search) without altering the agent's code.

FR 2.0: Active Resource Monitoring

The system SHALL implement a polling mechanism that continuously checks configured MCP Resources for updates. Examples of resources include:

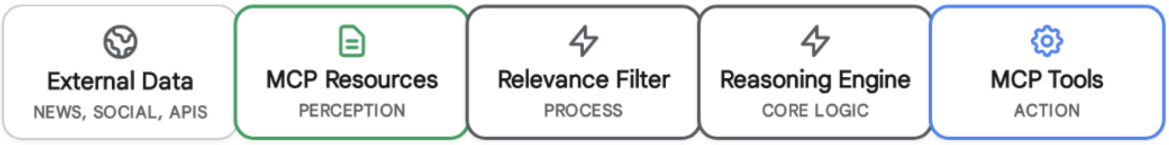
- twitter://mentions/recent: A resource providing the latest mentions of the agent.
- news://ethiopia/fashion/trends: A resource aggregating RSS feeds relevant to the agent's niche.
- market://crypto/eth/price: A resource providing real-time financial data.

FR 2.1: Semantic Filtering & Relevance Scoring

Upon ingesting content from a Resource, the system SHALL NOT automatically trigger a response. Instead, it must pass the content through a **Semantic Filter** utilizing a lightweight LLM (e.g., Gemini 3 Flash). This filter scores the content's relevance to the agent's current active goals. Only content exceeding a configurable **Relevance Threshold (e.g., 0.75)** shall trigger the creation of a Task for the Planner.

FR 2.2: Trend Detection

The system SHALL support a "Trend Spotter" background Worker that analyzes aggregated data from News Resources over time intervals (e.g., 4 hours). If a cluster of related topics emerges, it generates a "Trend Alert" that is fed into the Planner's context, prompting potential content creation opportunities.



MCP Perception-Action Cycle

Developer Notes & AI-Assist Prompts

- **MCP Concept:** In MCP, a "Resource" behaves like a file handle or a GET request. The agent "reads" a resource to obtain current state/data.
- **FastRender Connection:** The **Planner** agent acts as the listener. It subscribes to resource updates. When a significant update (passed by the Semantic Filter) occurs, the Planner creates a **Task** and puts it on the queue for a **Worker**.

Gemini CLI Prompt:

> "I need to build a custom MCP server for fetching local news using the Python mcp SDK.
>
> Write a complete Python script news_server.py.
>
> 1. Use the @mcp.resource decorator to expose a resource scheme news://latest.
> 2. Implement the resource handler to mock-fetch RSS feed items from a URL (use https://newsdata.io structure).
> 3. Return the news headlines as a plain text resource.
> 4. Include a list_resources handler that advertises this resource to the client.
> 5. Ensure the server uses stdio transport."

4.3 Creative Engine (Content Generation)

The Creative Engine is the "hands" of the agent, responsible for producing high-fidelity multimodal assets. This engine is modular, leveraging specialized AI models via MCP Tools.

FR 3.0: Multimodal Generation via MCP Tools

The system SHALL utilize specialized MCP Tools to generate content components. The Agent Core (LLM) acts as the director, orchestrating these tools:

- **Text:** Generated natively by the Cognitive Core (Gemini 3 Pro / Claude Opus) for captions, scripts, and replies.
- **Images:** Generated via mcp-server-ideogram or mcp-server-midjourney tools.[1]

- **Video:** Generated via mcp-server-runway or mcp-server-luma tools.

FR 3.1: Character Consistency Lock

To ensure the virtual influencer remains recognizable across thousands of posts, the system SHALL enforce a **Character Consistency** mechanism. All image generation requests initiated by a Worker MUST automatically include a specific character_reference_id or style LoRA (Low-Rank Adaptation) identifier in the payload sent to the image generation tool. This ID retrieves the canonical facial features and style settings for that specific agent.

FR 3.2: Hybrid Video Rendering Strategy To balance quality with extreme API costs, the system SHALL implement a tiered video generation strategy [1]:

- **Tier 1 (Daily Content):** "Living Portraits" generated using Static Image + Motion Brush (Image-to-Video). Cost-effective for routine updates.
- **Tier 2 (Hero Content):** Full Text-to-Video generation for major campaign milestones. The **Planner** agent determines the appropriate tier based on the assigned priority and available budget for the specific task.

Developer Notes & AI-Assist Prompts

- **Swarm Pattern:** This is a classic **Worker-Judge** interaction. The Worker calls the generation tool. The Judge then validates the output.
- **Validation Logic:** The Judge should use a Vision-capable model to verify that the generated image actually looks like the influencer *before* it is published.

Cursor Prompt:

> "Write a Python function for a 'Judge' agent tasked with validating image consistency.
>
> Function signature: validate_image_consistency(generated_image_url: str, reference_image_url: str) -> bool.
>
> Implementation details:
>
> 1. Use the Gemini 3 Pro Vision API (or GPT-4o).
> 2. Construct a prompt that sends both images to the model and asks: 'Does the person in image A look like the same person in image B? Answer strictly YES or NO.'
> 3. If the model returns 'NO', the function should raise a custom ValidationError with a descriptive message.
> 4. Include error handling for API timeouts."

4.4 Action System (Social Interface)

The Action System is the interface through which the agent interacts with social platforms.

FR 4.0: Platform-Agnostic Publishing

The system SHALL execute all social media actions (posting, replying, liking) via **MCP Tools** (e.g., twitter.post_tweet, instagram.publish_media). Direct API calls from the agent core logic are strictly prohibited. All calls must pass through the MCP layer to ensure:

- **Standardization:** The agent treats "Post to Twitter" and "Post to Threads" as similar tool calls.
- **Governance:** The MCP layer enforces rate limiting, logging, and "dry-run" capabilities for testing.

FR 4.1: Bi-Directional Interaction Loop

The system SHALL support a full interaction loop:

1. **Ingest:** Planner receives a comment via twitter://mentions Resource.
2. **Plan:** Planner creates a "Reply Task" and assigns it to a Worker.
3. **Generate:** Worker generates a context-aware reply (consulting Memory).
4. **Act:** Worker calls twitter.reply_tweet Tool.
5. **Verify:** Judge confirms the reply is safe and appropriate before allowing the Tool execution to finalize.

4.5 Agentic Commerce (Coinbase AgentKit)

This module acts as the "Financial Substrate," enabling agents to participate in the economy. This transforms them from passive chatbots into active economic participants.

FR 5.0: Non-Custodial Wallet Management Each Chimera Agent SHALL be assigned a unique, persistent, non-custodial wallet address via the **Coinbase AgentKit**.[7]

- **Key Security:** The wallet's private key (or seed phrase) MUST be secured using an enterprise-grade encrypted secrets manager (e.g., AWS Secrets Manager, HashiCorp Vault). The key is injected into the Agent Runtime environment only at startup and is never logged or exposed in the code.

FR 5.1: Autonomous On-Chain Transactions The system SHALL support the following autonomous actions via AgentKit Action Providers [22]:

- native_transfer: Capability to send ETH, USDC, or other base assets to external wallets (e.g., paying a graphic designer, transferring revenue to the agency).
- deploy_token: Capability to deploy ERC-20 tokens (e.g., for a fan loyalty program or social token).
- get_balance: Capability to check financial health. The Planner MUST check get_balance before initiating any cost-incurring workflow.

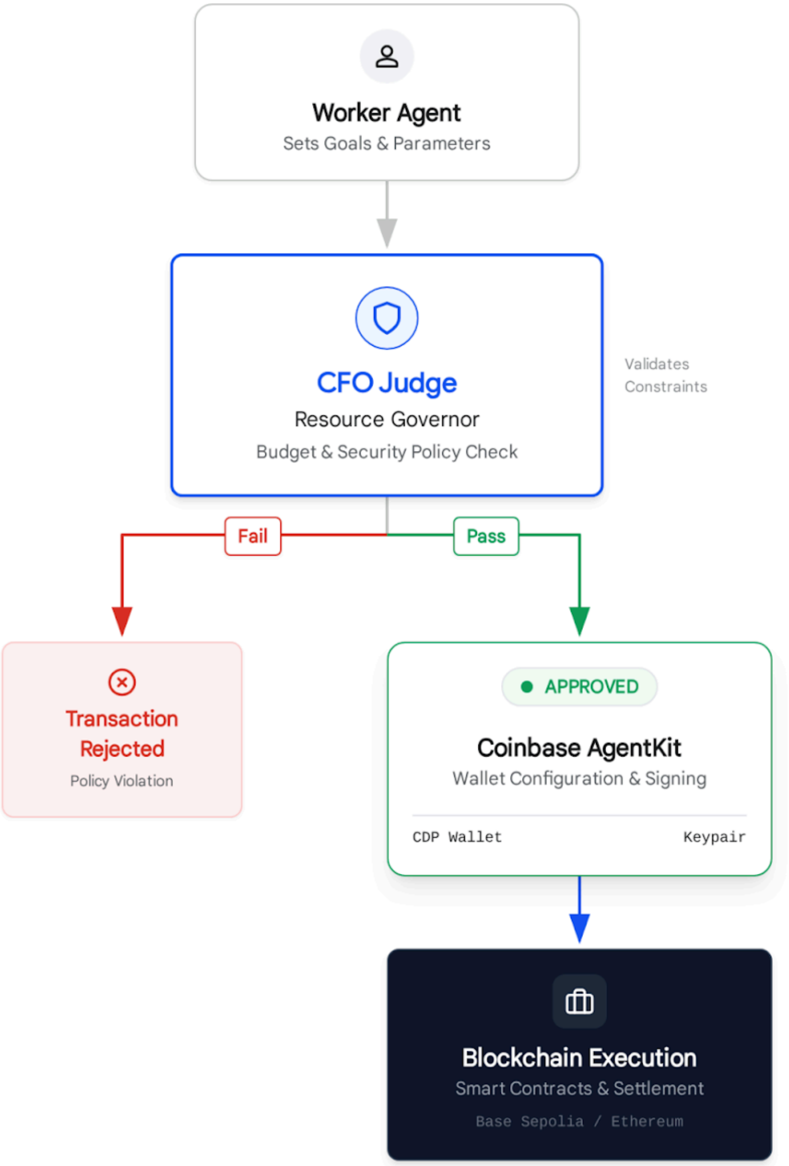
FR 5.2: Budget Governance (The "CFO" Sub-Agent)

To mitigate the risk of financial loss, a specialized **Judge** agent (designated as "The CFO") SHALL review every transaction request generated by a Worker.

- **Policy Enforcement:** The CFO Judge enforces strict, configurable budget limits (e.g., "Max daily spend: $50 USDC").
- **Anomaly Detection:** If a Worker proposes a transaction that exceeds the limit or matches a suspicious pattern, the CFO Judge strictly **REJECTS** the task and flags it

for human review.

Agentic Commerce Transaction Flow (with CFO Guardrail)



Transaction requests originate from a Worker Agent. They must pass through the 'CFO Judge' validation layer, which checks budget constraints via the Resource Governor and security policies before signing the transaction via Coinbase AgentKit.

Data sources: Coinbase Blog, Coinbase Docs, Github

Developer Notes & AI-Assist Prompts

- **Library:** coinbase-agentkit, langchain, cdp-sdk.
- **Environment:** Ensure CDP_API_KEY_NAME and CDP_API_KEY_PRIVATE_KEY are set in the .env file.

Cursor Prompt:

> "I am using the coinbase-agentkit Python SDK.
>
> Create a Python class named CommerceManager.
>
> 1. In the __init__ method, initialize CdpEvmWalletProvider using environment variables CDP_API_KEY_NAME and CDP_API_KEY_PRIVATE_KEY.
> 2. Implement an async method send_payment(to_address: str, amount_usdc: float) that uses the erc20_action_provider to transfer USDC on the Base network.
> 3. Add a custom decorator @budget_check to this method. This decorator should check a local Redis key daily_spend. If daily_spend + amount_usdc > MAX_DAILY_LIMIT, raise a BudgetExceededError and do not execute the transaction.
> 4. If the transaction succeeds, atomically update the daily_spend in Redis."

4.6 Orchestration & Swarm Governance

The Orchestrator manages the lifecycle of the agent swarm using the **FastRender Pattern**. This is the operating system of the Chimera network.

FR 6.0: Planner-Worker-Judge Implementation

The system SHALL implement the three-role architecture as distinct, decoupled services:

1. **Planner Service:** Continuously reads the GlobalState (Campaign goals). It generates tasks and pushes them to a TaskQueue (Redis).
2. **Worker Pool:** A scalable set of stateless agent containers that pop tasks from the TaskQueue. They perform the work and push the result to a ReviewQueue.
3. **Judge Service:** A specialized agent service that polls the ReviewQueue. It validates the output and commits changes to the GlobalState or re-queues the task.

FR 6.1: Optimistic Concurrency Control (OCC)

The Judge component SHALL implement Optimistic Concurrency Control. When a Judge attempts to commit a Worker's result to the GlobalState, it must check a state_version timestamp or hash.

- **Logic:** If the state has been modified by another process (e.g., another Planner update) since the Worker started the task, the commit fails. The Judge then invalidates the result and re-queues the task for the Planner to re-evaluate against the *new* state. This prevents "ghost updates" where an agent acts on obsolete information.[5]

5. Specific Requirements: Non-Functional

5.1 Human-in-the-Loop (HITL) & Confidence Thresholds

To balance the velocity of autonomy with the necessity of safety, the system implements a dynamic, probability-based HITL framework.

NFR 1.0: Confidence Scoring

Every action generated by a Worker (text, image, transaction) SHALL include a metadata field confidence_score (float 0.0 to 1.0), derived from the LLM's own probability estimation of the output's quality and safety.

NFR 1.1: Automated Escalation Logic The Judge agent SHALL route tasks based on the confidence_score according to the following logic tiers [15]:

- **High Confidence (> 0.90): Auto-Approve**. The action is executed immediately without human intervention.
- **Medium Confidence (0.70 - 0.90): Async Approval**. The task is paused and added to the **Orchestrator Dashboard** queue. The agent proceeds to other tasks, but this specific action remains pending until a human clicks "Approve."
- **Low Confidence (< 0.70): Reject/Retry**. The Judge automatically rejects the output and instructs the Planner to retry the task with a refined prompt or strategy.

NFR 1.2: Sensitive Topic Filters

Regardless of the confidence score, any content that triggers "Sensitive" topic filters (Politics, Health Advice, Financial Advice, Legal Claims) detected via keyword matching or semantic classification MUST be routed to the HITL queue for mandatory human review.

Developer Notes & AI-Assist Prompts

- **UI Integration:** The HITL interface should be a lightweight React application that consumes the ReviewQueue.

Cursor Prompt:

> "Generate a React component named ReviewCard using Tailwind CSS.
>
> It should accept props for: generated_content (string or image URL), confidence_score (number), and reasoning_trace (string).
>
> 1. Display the content prominently.
> 2. Display the confidence score with a color-coded badge (Green > 0.9, Yellow > 0.7, Red < 0.7).
> 3. Include two action buttons: 'Approve' (POSTs to /api/approve/{task_id}) and 'Reject' (POSTs to /api/reject/{task_id}).
> 4. If the confidence score is below 0.8, add a red border to the card to indicate 'High Attention Needed'."

5.2 Ethical & Transparency Framework

NFR 2.0: Automated Disclosure

All externally published media content SHALL utilize platform-native AI labeling features whenever available (e.g., setting the is_generated or ai_label flag in the Twitter/Instagram API payloads).

NFR 2.1: Identity Protection & Honesty

If an agent detects a direct inquiry regarding its nature (e.g., "Are you a robot?", "Is this AI?"), the Reasoning Engine MUST prioritize a specific **"Honesty Directive"** embedded in the system prompt. This directive overrides persona constraints to force a truthful, unambiguous disclosure (e.g., "I am a virtual persona created by AI.").

5.3 Performance & Scalability

NFR 3.0: Swarm Horizontal Scalability

The system SHALL support auto-scaling of the Worker Node pool. The architecture must be capable of managing a minimum of **1,000 concurrent agents** without degrading Orchestrator performance. This requires the Orchestrator to be stateless and the database layer (Weaviate/PostgreSQL) to be clustered.

NFR 3.1: Interaction Latency

The end-to-end latency for a high-priority interaction (e.g., replying to a direct message) SHALL NOT exceed **10 seconds** from ingestion to response generation (excluding HITL time).

6. Interface Requirements

6.1 Network Orchestration Dashboard

The dashboard serves as "Mission Control" for the Network Operators.

UI 1.0: Fleet Status View

A real-time visualization of all active agents. It must display:

- **Current State:** (Planning, Working, Judging, Sleeping).
- **Financial Health:** Current Wallet Balance (USDC/ETH).
- **Queue Depth:** Number of pending tasks in the HITL queue.

UI 1.1: Campaign Composer

A high-level interface for defining goals. The operator writes a natural language goal (e.g., "Hype up the new sneaker drop to Gen-Z audience"). The Planner Agent then decomposes this goal into a visible tree of sub-tasks, which the operator can inspect and modify before execution begins.

6.2 Data Models & Schemas

To ensure interoperability between the independent services (Planner, Worker, Judge), strict data schemas are required.

Schema 1: The Agent Task (JSON)

This schema defines the payload passed between Planner and Worker.

JSON

```
{
  "task_id": "uuid-v4-string",
  "task_type": "generate_content | reply_comment | execute_transaction",
  "priority": "high | medium | low",
  "context": {
    "goal_description": "string",
    "persona_constraints": ["string"],
    "required_resources": ["mcp://twitter/mentions/123", "mcp://memory/recent"]
  },
  "assigned_worker_id": "string",
  "created_at": "timestamp",
  "status": "pending | in_progress | review | complete"
}
```

Schema 2: The MCP Tool Definition (JSON Schema)

Standard definition for an MCP Tool, exemplified here by a Social Post tool.

JSON

```
{
  "name": "post_content",
  "description": "Publishes text and media to a connected social platform.",
  "inputSchema": {
    "type": "object",
    "properties": {
      "platform": {
        "type": "string",
        "enum": ["twitter", "instagram", "threads"]
      },
      "text_content": {
        "type": "string",
        "description": "The body of the post/tweet."
      },
      "media_urls": {
        "type": "array",
        "items": {"type": "string"}
      },
      "disclosure_level": {
        "type": "string",
        "enum": ["automated", "assisted", "none"]
      }
    },
    "required": ["platform", "text_content"]
  }
}
```

7. Implementation Roadmap & Genesis Prompts

This section provides the "Genesis Prompts"—master instructions designed to be fed into AI coding assistants (Cursor, Claude Code) to bootstrap the project's core modules rapidly.

7.1 Phase 1: The Core Swarm (Timeframe 1)

Objective: Establish the Planner-Worker-Judge loop and the Task Queue infrastructure.

Genesis Prompt (for Cursor/Claude):

> "I am building a multi-agent system using the FastRender swarm pattern in Python.
>
> Please create a project structure with three main service folders: planner, worker, judge.
>
> 1. **Planner Service:** Write a Python service that reads a goals.json file, decomposes it using a mock LLM call, and pushes Task objects to a Redis queue named task_queue.
> 2. **Worker Service:** Write a Python service that pops tasks from task_queue, simulates work (async sleep 2s), and pushes a Result object to a Redis queue named review_queue.
> 3. **Judge Service:** Write a Python service that pops from review_queue, checks if result.status == 'success', and logs the outcome.
>   Use pydantic to define strict schemas for Task and Result. Use redis-py for queuing."

7.2 Phase 2: MCP Integration (Timeframe 2)

Objective: Connect the swarm to external data and tools via MCP.

Genesis Prompt:

"I need to integrate the Model Context Protocol (MCP) into my agent system.

1. Install the mcp python SDK.
2. Create a class MCPClient that can connect to a local MCP server process via Stdio transport.
3. Implement a method call_tool(server_name, tool_name, arguments) that sends a standard JSON-RPC request to the connected server.
4. Write a script to spawn an instance of the mcp-server-sqlite (as a reference implementation) and use your client to query a test database table."

7.3 Phase 3: Agentic Commerce (Timeframe 3)

Objective: Enable financial autonomy and wallet management.

Genesis Prompt:

"Integrate the Coinbase AgentKit into the Worker service.

1. Set up a wallet provider using CdpEvmWalletProvider from the coinbase-agentkit library.
2. Create an Action Provider class that exposes transfer_asset and get_wallet_balance functionality.
3. Write a test script where an agent checks its own balance. If the balance is greater than 10 USDC, it transfers 1 USDC to a hardcoded 'Savings Wallet' address.
4. Ensure that the API keys and Private Keys are read strictly from os.environ and raise an error if they are missing."

8. Conclusions

Project Chimera (2026 Edition) represents the convergence of three transformative technologies: the **MCP** standard for universal connectivity, **Swarm Architectures** for robust autonomous coordination, and **Agentic Commerce** for economic independence. By adhering to this SRS, the development team will deliver a resilient, scalable network of virtual influencers capable of operating with genuine agency in the digital economy. The shift from static scripts to dynamic, goal-seeking agents requires rigorous adherence to the **Planner-Worker-Judge** pattern to ensure that this autonomy remains aligned with strategic and ethical objectives.

Works cited

1. AI Influencer Agent SRS Plan
2. Architecture overview - Model Context Protocol, accessed February 4, 2026, https://modelcontextprotocol.io/docs/learn/architecture
3. Disruptive Innovation or Industry Buzz? Understanding Model Context Protocol's Role in Data-Driven Agentic AI | Informatica, accessed February 4, 2026, https://www.informatica.com/blogs/disruptive-innovation-or-industry-buzz-understanding-model-context-protocols-role-in-data-driven-agentic-ai.html
4. Simon Willison on cursor, accessed February 4, 2026, https://simonwillison.net/tags/cursor/
5. Why AI Swarms Cannot Build Architecture - GitHub Pages, accessed February 4, 2026, https://jsulmont.github.io/swarms-ai/
6. Coinbase enables agentic commerce for OpenAI's Agents SDK with launch day support, accessed February 4, 2026, https://www.coinbase.com/developer-platform/discover/launches/openai-agents-sdk
7. Wallet Management - Coinbase Developer Documentation, accessed February 4, 2026, https://docs.cdp.coinbase.com/agent-kit/core-concepts/wallet-management
8. What is vibe coding? | AI coding - Cloudflare, accessed February 4, 2026, https://www.cloudflare.com/learning/ai/ai-vibe-coding/
9. What is Vibe Coding? How To Vibe Your App to Life - Replit Blog, accessed February 4, 2026, https://blog.replit.com/what-is-vibe-coding
10. Create custom subagents - Claude Code Docs, accessed February 4, 2026, https://code.claude.com/docs/en/sub-agents
11. I asked a developer for vibe coding best practices and this what they shared. - Reddit, accessed February 4, 2026, https://www.reddit.com/r/vibecoding/comments/1l8r502/i_asked_a_developer_for_vibe_coding_best/
12. Vibe coding is not the same as AI-Assisted engineering. | by Addy Osmani - Medium, accessed February 4, 2026, https://medium.com/@addyosmani/vibe-coding-is-not-the-same-as-ai-assisted-engineering-3f81088d5b98
13. Cursor conducted an experiment in which hundreds of agents collaborated to build a browser from scratch, writing 1,000 files and over 1 million lines of code in a week. - GIGAZINE, accessed February 4, 2026, https://gigazine.net/gsc_news/en/20260120-cursor-scaling-agents/
14. The Rise of Onchain AI: Agents, Apps, and Commerce - Coinbase, accessed February 4, 2026, https://www.coinbase.com/blog/the-rise-of-onchain-ai-agents-apps-and-commerce
15. Human-in-the-Loop Agentic Systems Explained | by Tahir | Medium, accessed February 4, 2026, https://medium.com/@tahirbalarabe2/human-in-the-loop-agentic-systems-explained-db9805dbaa86

16. Human-in-the-Loop AI: Benefits and Challenges, accessed February 4, 2026, https://blog.naitive.cloud/human-in-the-loop-ai-benefits-challenges/
17. A new era of intelligence with Gemini 3 - The Keyword, accessed February 4, 2026, https://blog.google/products-and-platforms/products/gemini/gemini-3/
18. Claude Opus 4.5 System Card - Anthropic Brand Portal, accessed February 4, 2026, https://assets.anthropic.com/m/64823ba7485345a7/Claude-Opus-4-5-System-Card.pdf
19. Model Context Protocol architecture patterns for multi-agent AI systems - IBM Developer, accessed February 4, 2026, https://developer.ibm.com/articles/mcp-architecture-patterns-ai-systems
20. Building a Full-Fledged MCP Workflow using Tools, Resources, and Prompts, accessed February 4, 2026, https://www.dailydoseofds.com/model-context-protocol-crash-course-part-4/
21. Inside the OpenClaw Ecosystem: What Happens When AI Agents Get Credentials to Everything - Permiso, accessed February 4, 2026, https://permiso.io/blog/inside-the-openclaw-ecosystem-ai-agents-with-privileged-credentials
22. agentkit/python/coinbase-agentkit/README.md at main · coinbase/agentkit - GitHub, accessed February 4, 2026, https://github.com/coinbase/agentkit/blob/main/python/coinbase-agentkit/README.md