Loop Symphony

Product Requirements Document

Version 2.0

January 2026

Author: Matt / Pipchi Productions

A modular agentic server framework that orchestrates autonomous cognitive loops across distributed environments.

Executive Summary

Loop Symphony is a distributed agentic framework that orchestrates autonomous cognitive loops across multiple environments: iOS devices, cloud servers, local edge computing, and future robotics platforms. It provides personality-agnostic problem-solving capabilities that can be accessed by any client application.

The framework is built on three core insights:

- Intelligence separation: Keep focused instruments simple with clear termination criteria. Let the orchestrating Manager handle complexity.
- Distributed rooms: Different environments (iOS, Server, Local, Robotics) have different capabilities. Route work to the right place.
- Process visibility layers: Like the human body, some processes run invisibly (heart), some run automatically but can be overridden (breathing), and some require active engagement (walking).

This document defines the complete architecture, interfaces, and phased build plan for Loop Symphony across iOS, Server, and Local environments.

Table of Contents

1. Vision & Mental Model

1.1 The Core Insight

Intelligence isn't just raw model capability—it's knowing which cognitive structure to apply to which problem. Loop Symphony implements this through a Manager that analyzes incoming tasks and orchestrates the appropriate combination of autonomous loops to solve them.

MIT research on agentic AI shows that overly complex frameworks fail at rates as high as 95%. The keys to success are simplicity, clear termination, and iterative oversight. Loop Symphony implements these principles through the scientific method structure and quantifiable done signals.

1.2 The Symphony Vocabulary

Concept	Symphony Term	Description
Single prompt/response	Note	The atomic unit of cognition—one call, one response
Autonomous loop	Phrase	A goal-directed sequence following the scientific method
Loop types	Instruments	Different phrase implementations (research, vision, synthesis)
Task analyzer	Manager/Conductor	Reads tasks, selects instruments, orchestrates execution
Execution environments	Rooms	iOS, Server, Local, Robotics—each with different capabilities
Visibility level	Process Type	Autonomic, Semi-Autonomic, or Conscious

1.3 The Scientific Method Loop

Each autonomous loop (Phrase) follows the scientific method rather than the vaguer "think → act → observe" pattern. This provides clearer structure and explicit done signals:

Phase	What Happens	Done Signal
Problem/Observation	Define scope, understand what we're solving	Clear problem statement exists
Hypothesis	Generate possible approaches (can be parallel)	N hypotheses generated
Test/Experiment	Try things, gather data (can nest sub-loops)	Tests executed, data collected

Results/Analysis	What did we learn? Merge parallel results	Analysis complete
Reflection & Termination	Are we done? Should we iterate?	Explicit done criteria evaluated

2. Core Architecture

2.1 High-Level Overview

Loop Symphony is a distributed system spanning multiple environments, unified by a common protocol and orchestrated by an intelligent Manager.

2.2 Architecture Principles

1. Personality at the edges: Loop Symphony is personality-agnostic. Personality lives in the iOS app (via soul.md), not in the core framework.
2. Intelligence separation: Instruments are focused specialists with clear termination. The Manager is the generalist that understands composition.
3. Clean interfaces: Rooms communicate through well-defined contracts (TaskRequest/TaskResponse). Neither room needs to understand the other's internals.
4. Bounded creativity: The Manager can propose novel loops, but within constraints (scientific method structure, explicit termination, registered tools only).
5. Process visibility layers: Like the human body, different processes have different visibility—autonomic (invisible), semi-autonomic (automatic but overridable), and conscious (full engagement).

3. The Rooms Model

3.1 What is a Room?

A Room is an execution environment with its own capabilities, constraints, and intelligence. Each room does its job well without needing to know how other rooms work—like organs in a body.

3.2 Room Definitions

iOS Room (Reception)

- Location: User's iOS device
- Intelligence: Local LLM or personality partition
- Capabilities: Camera, microphone, sensors, notifications, user presence, personality (soul.md)
- Latency: Realtime (<100ms for local responses)
- Role: Human-like interaction, intent extraction, personality wrapping, quick local responses

Server Room (Engine)

- Location: Cloud server (Railway, Fly.io, etc.)
- Intelligence: Manager/Conductor with full orchestration capabilities
- Capabilities: Claude API, web search, web fetch, database, complex loops, parallel execution
- Latency: Async (2-300+ seconds depending on task)
- Role: Problem solving, research, synthesis, orchestration—personality-agnostic

Local Room (Edge)

- Location: User's Mac or local Linux machine
- Intelligence: Local LLM (Ollama, LM Studio, etc.)
- Capabilities: Offline operation, private data handling, file access, fast inference
- Latency: Fast (<2 seconds)
- Role: Privacy-sensitive tasks, offline fallback, quick local analysis

Robotics Room (Future)

- Location: Physical robot
- Capabilities: Physical actuators, sensors, embodiment
- Role: Physical task execution, real-world interaction

3.3 Room Coordination

The Manager routes tasks to the appropriate room(s) based on task requirements, room capabilities, and current room state (online, load, battery).

Cross-room patterns include: Handoff (iOS → Server → iOS), Parallel (iOS camera + Server search), Escalation (Local → Server if needed), and Sensor+Compute split (Robotics sensors + Server analysis).

4. Process Visibility Layers

4.1 The Three Intelligences Model

Inspired by how the human body operates, Loop Symphony implements three levels of process visibility:

Autonomic (Heart)

- Runs constantly, invisibly
- User CANNOT see or control
- Only surfaces on CRITICAL ERROR ("pain response")
- Examples: heartbeat checks, token refresh, connection keepalive, memory compaction
- Error communication: "Something's not working right. I'm looking into it." (not technical details)

Semi-Autonomic (Breathing)

- Runs automatically by default
- User CAN override/take control if they choose
- Surfaces status on request, or on notable events
- Examples: background research, scheduled tasks, inbox monitoring
- Error communication: "I couldn't find much on X. Want me to try a different approach?"

Conscious (Walking)

- Requires active engagement
- User sees progress, can steer
- Full visibility into what's happening
- Examples: active queries, interactive research, approvals
- Error communication: "I found conflicting information. Source A says X, Source B says Y. Which seems more relevant?"

5. The Core Interface: iOS ↔ Server

5.1 Design Principles

The iOS ↔ Server interface is the core interaction that must be rock solid and seamless. Each side knows its job and doesn't need to understand the other's internals.

iOS Room Knows:

- Who the user is (soul.md, preferences, history)
- How to talk to them (voice, tone, personality)
- What they're asking for (intent extraction)
- How to package a clean request
- When to handle things locally vs. escalate to server

iOS Room Does NOT Know:

- HOW the server solves problems
- What instruments exist
- How loops work internally
- Termination criteria details

Server Room Knows:

- How to analyze tasks and select instruments
- How to run loops (scientific method)
- When things are done (termination)
- How to propose novel approaches

Server Room Does NOT Know:

- Who the user is (just gets task context)
- What personality to use
- How to talk to humans
- UI/UX concerns

5.2 Interface Contract

TaskRequest (iOS → Server)

The iOS room packages user intent into a clean request:

- id: Unique request identifier
- query: What the user wants (cleaned intent)
- context: User ID, conversation summary, attachments, location, time
- preferences: Thoroughness (quick/balanced/thorough), trust level, notification settings

TaskResponse (Server → iOS)

The server returns results with rich metadata:

- outcome: Complete, Saturated, Bounded, or Inconclusive
- findings: Structured list of what was learned
- summary: Pre-synthesized text for iOS to wrap in personality
- confidence: 0-1 score of result quality

- metadata: Loops used, iterations, timing, sources
- discrepancy: If inconclusive, what conflicted
- suggested_followups: If inconclusive, better hypotheses to try

6. Termination & Outcomes

6.1 The Four Outcomes

Most agent loops fail because "done" is vibes-based. Loop Symphony makes termination quantifiable with four explicit outcome states:

Outcome	Description
COMPLETE	Confidence converged, task solved. The answer is reliable.
SATURATED	Nothing new to learn. Best answer available given available information.
BOUNDED	Hit resource limits (iterations, time, API calls). May need more resources to continue.
INCONCLUSIVE	Conflicting signals detected. Needs human review or better hypothesis.

6.2 The Three Done Criteria

Any of these triggers loop completion:

6. Confidence — Has our certainty converged? (<5% change across cycles means we've stabilized)
7. Saturation — Are we still learning anything new? (No new data/insights = diminishing returns)
8. Bounds — Have we hit resource limits? (Max iterations, time budget, API call limits)

6.3 Inconclusive Handling

Inconclusive is not a failure—it's an honest acknowledgment that the problem needs refinement. An Inconclusive outcome includes:

- findings: What we learned despite the conflict
- discrepancy: What specifically conflicted and how
- suggested_refinements: Better hypotheses for a retry

This allows the loop to be rerun with better parameters, making the system learnable.

7. The Manager (Conductor)

7.1 Role and Responsibilities

The Manager is the intelligent orchestrator. While instruments are focused specialists, the Manager is a generalist that understands when to use which specialist and how to interpret their outputs.

7.2 Manager Capabilities

Level 1: Selection

Pick from existing instruments based on task analysis.

Level 2: Composition

Arrange instruments in sequences, parallels, and nested structures.

Level 3: Parameterization

Tune existing instruments for specific problems (tighter bounds, higher confidence thresholds).

Level 4: Novel Arrangement

Create new compositions not explicitly designed (e.g., Research in parallel with Vision, merge conflicts, then nested Research on contradictions).

Level 5: Loop Proposal

Propose entirely new loop specifications when existing instruments don't fit. These proposals are constrained by: scientific method structure required, explicit termination criteria required, registered tools only, resource bounds inherited.

7.3 Creativity Constraints

Following the MIT research insight, the Manager's creativity is bounded:

- MUST use scientific method structure
- MUST have explicit termination criteria
- MUST use registered tools only
- MUST stay within resource bounds
- SHOULD explain rationale for novel approaches

7.4 Meta-Learning

When the Manager proposes a novel arrangement that works well, it can suggest saving it as a named instrument for future reuse. This allows the system to learn new capabilities without code changes.

8. Instruments

8.1 Core Instruments

Note (Atomic)

- Purpose: Answer simple, direct questions
- Iterations: 1 (always terminates after one cycle)
- Termination: Bounds = 1

Research Phrase (Iterative)

- Purpose: Find and synthesize information on a topic
- Iterations: Up to 5 (default)
- Termination: Confidence >80%, or saturation, or bounds

Vision Phrase (Iterative)

- Purpose: Extract information from images
- Iterations: Up to 3
- Termination: All requested information found, or saturation, or bounds

Synthesis Phrase (Single-pass)

- Purpose: Combine inputs from multiple instruments into coherent output
- Iterations: 1 (may iterate if confidence low)
- Termination: Output generated

8.2 Composition Patterns

Sequential

Loop A completes → output feeds into Loop B → output feeds into Loop C

Parallel (Fan-out/Fan-in)

Multiple instruments test competing hypotheses simultaneously. Results merge in Analysis phase with confidence weighting.

Nested (Recursive Sub-loops)

A loop can spawn sub-loops when it encounters complex sub-problems. Nested loops use their own termination criteria. Depth is bounded to prevent infinite recursion (default max: 3).

9. Trust Escalation

9.1 The Brilliant New Employee Model

Loop Symphony implements a trust escalation system, treating the system like a brilliant but new employee who needs oversight at first.

9.2 Trust Levels

Level	Behavior
0: Supervised	Human approves plan before execution. Human reviews results before delivery. All errors surface immediately. Full metadata visible.
1: Semi-Auto	Human approves plan, auto-executes. Results delivered with summary for review. Errors surface but continues if non-fatal. Metadata available on request.
2: Autonomous	Auto-plans, auto-executes. Results delivered directly. Only critical errors surface. Metadata logged for audit.

Trust level can be set per-user (in soul.md), per-task, or can evolve over time as patterns prove reliable.

10. Error Handling & Learning

10.1 Error Communication by Process Type

Process Type	What User Sees	Why
Autonomic	"Something's wrong. Give me a moment."	User can't fix it. System will try to recover.
Semi-Autonomic	"I couldn't find much on X. Try different approach?"	User might want to redirect or let it go.
Conscious	"I found conflicting info. A says X, B says Y. Which is more relevant?"	User is engaged and can help resolve.

10.2 Error Reports for Learning

In early iterations (Trust Level 0), errors are surfaced with learning context:

- error_type: Classification (api_failure, timeout, contradiction, etc.)
- context: What was happening when error occurred
- recovery_action: What the system did
- learning: What this suggests about the approach

Over time, patterns emerge that become institutional knowledge.

10.3 Compaction Strategies

Different strategies for different situations:

- Summarization: Chunk → summarize → merge (for long context)
- Pruning: Score findings by relevance, drop bottom N%
- Archival: Move to external storage, keep reference
- Hierarchical: Collapse inner loop details, keep outer structure
- Selective: Mark "must keep" vs "can compress"

11. Build Phases

11.1 Overview

Loop Symphony is built across three (eventually four) disciplines: iOS, Server, and Local environments. The build is phased to establish core functionality first, then expand.

11.2 Phase 1: Foundation (Weeks 1-4)

Goal: Establish iOS ↔ Server core interaction with basic loop execution.

Server Room - Week 1-2

- Set up Python project structure with FastAPI
- Implement TaskRequest/TaskResponse contract
- Configure Supabase database with core schema
- Implement Termination Evaluator (confidence, saturation, bounds)
- Implement Note instrument (atomic, single-cycle)
- Implement Research instrument (iterative, web search)
- Basic Manager that routes to Note or Research
- Deploy to Railway

iOS Room - Week 3-4

- Set up Swift package structure
- Implement LoopSymphonyClient for server communication
- Basic intent extraction (local/server decision)
- soul.md loader and personality configuration
- Response wrapping with personality
- End-to-end test: iOS → Server → iOS

11.3 Phase 2: Intelligence (Weeks 5-8)

Goal: Add sophisticated Manager capabilities and process visibility.

Server Room - Week 5-6

- Implement four-state termination (Complete, Saturated, Bounded, Inconclusive)
- Implement discrepancy detection and reporting
- Implement Vision instrument
- Implement Synthesis instrument
- Sequential composition pattern
- Parallel composition pattern with conflict resolution

Server Room - Week 7-8

- Manager parameterization (tuning instruments per task)
- Nested sub-loop support with bounded depth
- Process type assignment (autonomic/semi/conscious)
- Checkpoint emission for passive monitoring
- Streaming support for conscious processes

iOS Room - Week 7-8

- Handle all four outcome states in UI
- Streaming display for conscious processes
- Trust level configuration per soul.md
- Error display appropriate to process type

11.4 Phase 3: Autonomy (Weeks 9-12)

Goal: Add Manager creativity, trust escalation, and background processing.

Server Room - Week 9-10

- Novel arrangement generation
- Loop proposal system (constrained creativity)
- Meta-learning: save successful arrangements as named instruments
- Autonomic process layer (heartbeat, compaction, token refresh)

Server Room - Week 11-12

- Semi-autonomic process layer (background research, scheduled tasks)
- Multiple compaction strategies
- Error learning and pattern detection
- Notification layer (Telegram, push)

iOS Room - Week 11-12

- Background task awareness
- "What are you working on?" query support
- Trust level evolution UI
- Novel loop approval flow (Trust Level 0)

11.5 Phase 4: Local Room (Weeks 13-16)

Goal: Add edge computing, offline capability, and privacy-sensitive processing.

Local Room - Week 13-14

- Local LLM integration (Ollama/LM Studio)
- Room registration with Server
- Basic Note instrument (local)
- File access tools

Local Room - Week 15-16

- Offline fallback when server unreachable
- Privacy-sensitive task routing
- Escalation to Server for complex tasks
- Heartbeat and state synchronization

Cross-Room Integration - Week 15-16

- Manager routing across iOS, Server, Local
- Parallel room execution (iOS camera + Server search)
- Graceful degradation when rooms offline

11.6 Phase 5: Knowledge Layer (Weeks 17-18)

Goal: Implement the learnable knowledge layer for user guidance and capability education.

- Implement capabilities.md, boundaries.md, patterns.md, changelog.md
- iOS room reads and applies knowledge files
- Server pushes knowledge updates to iOS rooms
- User pattern detection and intervention system
- Proactive suggestion engine
- Request scoping and pushback logic

11.7 Phase 6: Future (TBD)

- Robotics Room integration
- Advanced meta-learning
- Multi-user/multi-personality support
- FixionMail integration (writing room as personality)

12. Success Criteria

12.1 Phase 1 Complete When:

- iOS app can submit task via POST /task and receive task_id
- Manager correctly routes simple queries to Note, research queries to Research
- Research instrument performs multi-step web research following scientific method
- Termination works: loops stop on confidence convergence, saturation, or bounds
- iOS receives results and wraps them in personality
- End-to-end latency is acceptable for real use

12.2 Phase 2 Complete When:

- All four outcome states are properly detected and communicated
- Inconclusive outcomes include discrepancy and suggestions
- Sequential and parallel composition work reliably
- Streaming works for conscious processes
- Trust levels affect behavior as specified

12.3 Phase 3 Complete When:

- Manager can propose and execute novel arrangements
- Background tasks run invisibly until complete
- Autonomic processes only surface on critical errors
- Error patterns are logged and learning suggestions made

12.4 Phase 4 Complete When:

- Local room handles simple queries offline
- Privacy-sensitive tasks stay local
- Manager routes across all three rooms appropriately
- System degrades gracefully when rooms go offline

12.5 Phase 5 Complete When:

- Knowledge files are synced between Server and iOS
- iOS room proactively suggests solutions to detected pain points
- iOS room pushes back on unrealistic requests with alternatives
- iOS room helps scope overwhelming requests
- Capability education happens naturally in conversation

13. The Knowledge Layer

13.1 Overview

Most users don't know how to effectively use agentic systems. They either underutilize (treating it like Google) or overreach (expecting magic). The Knowledge Layer is a set of learnable .md files that help the iOS room guide users toward effective use.

13.2 The Knowledge Files

The iOS room maintains a set of markdown files that define what the system can do, what it can't, and how to help users:

File	Purpose
soul.md	Personality configuration (voice, tone, preferences)
capabilities.md	What the system CAN do, including new capabilities
boundaries.md	What the system CAN'T do, with redirects and alternatives
patterns.md	Common user patterns and appropriate interventions
changelog.md	What's new (for educating users about new capabilities)
user/{id}.md	Learned patterns for this specific user

13.3 The Four Interventions

1. Proactive Suggestions

When the iOS room notices recurring pain points (topics mentioned 3+ times without resolution), it suggests solutions:

"You've asked about meal planning three times this week. Want me to put together a system that actually works for you?"

2. Pushback on Unrealistic Requests

When users ask for impossible things, the iOS room redirects to achievable alternatives:

"I can't predict the stock market—nobody can reliably. But I could help you research investment strategies and understand your options. Interested?"

3. Scoping Overwhelming Requests

When requests are too broad, the iOS room helps break them down:

"Building an app is a big project. Let's start smaller: What's the ONE thing this app needs to do first?"

4. Capability Education

When users underutilize the system, the iOS room gently educates:

"By the way, I can do more than quick lookups. If you ever have something that needs real research or has multiple steps, I'm here for that too."

13.4 Celebration of Simplification

When complexity becomes simplicity, the iOS room celebrates:

"Notice you haven't asked about email overwhelm lately? I think we cracked it!"

This reinforces value and teaches users what's possible.

13.5 The Update Flow

Knowledge files are updated through a sync process:

- Server pushes updates when capabilities change (new instruments, new model, etc.)
- iOS syncs user-specific learnings back to Server
- Server aggregates patterns across users to improve global patterns.md
- Improvements flow back to all iOS rooms

13.6 When New Technology Arrives

When a new model drops or capability is added:

9. Server updates capabilities.md and changelog.md
10. Server pushes updates to all connected iOS rooms
11. iOS room reads changelog and can naturally mention new capabilities
12. Users learn about improvements through conversation, not release notes

13.7 Why Markdown?

- Human-readable: Developers can edit directly
- Version-controlled: Track changes over time
- No code changes: Update knowledge without app updates
- LLM-friendly: iOS room's local LLM can parse and apply naturally
- Modular: Update one file without touching others

14. Open Questions

13. Authentication: How should iOS apps authenticate with the server? API key per app? JWT? OAuth?
14. Rate limiting: Should there be limits on task submission? Per-user? Per-app?
15. Task cancellation: Can in-progress tasks be cancelled? How does this propagate to nested loops?
16. Result expiration: How long should results stay in the inbox? Should this be configurable?
17. Error recovery: If a loop fails mid-execution, can it resume from checkpoint?
18. Confidence calibration: How do we tune the 0.05 confidence threshold? Per-instrument? Learned over time?
19. Saturation algorithms: What's the best way to detect "nothing new"? Semantic similarity? Fact counting?
20. Parallel conflict resolution: When loops disagree, how much weight for confidence vs. evidence counting?
21. Room discovery: How do rooms find and register with each other? mDNS? Central registry?
22. Multi-user: Can Loop Symphony serve multiple users with different souls? How is isolation handled?
23. Knowledge sync frequency: How often should knowledge files sync? On every update? Batched?
24. Pattern privacy: How much user pattern data should sync to the server? Local-only option?

Loop Symphony: Orchestrating intelligence through the scientific method, across distributed rooms, one bounded loop at a time.