Comp-Core System Architecture

Computational Choreography Framework

v2.0.0 | Production Architecture

Computational Choreography

"What does movement mean, and how can we compute it in real time?"

TrajectoryOS

Long-Horizon Operating System

Hours → Years

RAG	**RAG++** 5D Trajectory Memory Fabric	HNSW · I-RCP
ORB	**Orbit** Project & Session Orchestration	Axum
CTW	**CognitiveTwin** Style Learning & Signature	EMA
MCP	**MCP Server** AI Assistant Integration	Tools

Echelon

Real-Time Embodied Engine

Milliseconds

L0	**cc-relay** TLV Parsing, Mocopi Protocol	<50µs · Rust
L1	**cc-collection** Extended Kalman Filter Fusion	<100µs · 13-dim
L2	**cc-anticipation** 7 Scalars: Commitment, Uncertainty	<2ms
L3	**DELL** Dual Equilibrium (60Hz + 2.5Hz)	<500µs
L4	**cc-conductor** Beat Scheduling, Audio DSP	<6ms

End-to-End Data Flow

Mocopi	cc-relay	cc-collection	cc-window	cc-anticipation	DELL	cc-brain	cc-conductor	Audio
27-bone	TLV	EKF	50Hz	7 scalars	Equilibrium	Latent	Beat	DJ
<1ms	<50µs	<100µs	<1ms	<2ms	<500µs	<1ms	<100µs	<6ms

Data Flow Pipeline

End-to-end motion processing from sensor to audio output

LAYER 0: SENSOR INPUT

SENSORS

Sony Mocopi
27-bone IMU array
UDP 12351 · TLV · 30-60Hz

iPhone/Watch
CoreMotion IMU
WebSocket · 60Hz

MediaPipe
Camera landmarks
WebSocket · 30Hz

BoneTransform

```
position: [f32; 3]
quaternion: [f32; 4]
```

LAYER 1-2: FUSION & WINDOWING

PROCESSING

cc-relay
TLV parsing, zero-copy
Rust · <50µs

→

cc-collection
Extended Kalman Filter
13-dim state · <100µs

→

cc-window-aligner
Canonical 50Hz windows
50 frames · <1ms

FusedSkeleton

```
limbs: Vec<LimbState>
coverage: f32
```

LAYER 3-4: SEMANTIC EXTRACTION

ANALYSIS

cc-anticipation
7 scalar extraction
commitment · uncertainty · <2ms

→

cc-gesture
Neighbor voting
HNSW k=10 · <500µs

→

DELL
Dual equilibrium
60Hz + 2.5Hz · <500µs

AnticipationPacket

```
commitment: f32
uncertainty: f32
transition_pressure: f32
```

LAYER 5-6: OUTPUT

CONTROL

cc-brain
Latent state machine
Entry→Peak→Release · <1ms

→

cc-conductor
Beat scheduling
Ableton Link · <100µs

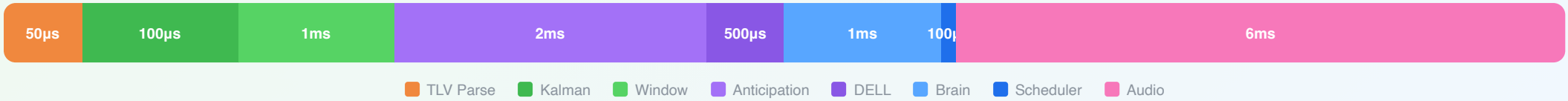
→

Audio Engine
Real-time DSP
48kHz · <6ms

→

DJ Software
Rekordbox / Serato
MIDI · Keyboard

⏱ Latency Budget Distribution (Total: <10ms)



📐 Position → Velocity

```
v = dp/dt
```

First derivative of position

📐 Velocity → Acceleration

```
a = dv/dt = d²p/dt²
```

Second derivative

📐 Quaternion → Angular Velocity

```
ω = 2·q*·dq/dt
```

Rotation rate extraction

📐 Energy → Commitment

```
C = 1 - U·(1 - constraint)
```

Anticipation scalar

Deployment Topology

Hybrid Cloud Infrastructure

Local Network

Real-time processing

Sony Mocopi

27-bone IMU sensors

`UDP broadcast`

iPhone/Watch

CoreMotion IMU

`WebSocket`

DJ Software

Rekordbox/Serato

`MIDI/Keyboard`

Visualization

Three.js UI

`localhost:3000`

DigitalOcean VPS

Edge relay

cc-relay (Rust)

TLV parsing, batching

`159.65.190.11:12351`

Prometheus Metrics

Monitoring endpoint

`:9090/metrics`

Google Cloud

Backend services

cc-mcs-headless

Motion processing daemon

`136.114.76.114:8765`

RAG++ Service

Memory fabric (FastAPI)

`Cloud Run`

Orbit Server

Orchestration (Axum)

`Cloud Run`

Supabase

PostgreSQL + pgvector

`memory_turns`

Network Flow

~110ms	<17ms	3	99.9%
Cloud Path Latency	Local Path Latency	Cloud Regions	Target Uptime

Generated: 2025-01-01 | Version 2.0.0

Anticipation Pipeline

cc-anticipation: MotionWindow → 7 Scalars + 3 Vectors

📥 Input: MotionWindow

SkeletonFrames
50 frames @ 50Hz (1 second)
`27 bones × 7 values per frame`

Coverage Check
Minimum coverage required
`coverage ≥ 0.9`

Timestamps
Window time boundaries
`t_start, t_end (microseconds)`

⚙️ Feature Extraction

Position → Velocity
First derivative: dp/dt

Velocity → Acceleration
Second derivative: d^2p/dt^2

Acceleration → Jerk
Third derivative: d^3p/dt^3

Regime Embedding
Motion character encoding
`64-256 dimensions`

Constraint Vector
Physical limits proximity
`~8 dimensions`

📤 Output: AnticipationPacket

7 Scalars
Anticipation signals

regime_embedding
Motion regime vector
`[64-256 dims]`

constraint_vector
Physical constraints
`[~8 dims]`

derivative_summary
Dynamics summary
`[~8 dims]`

Primary Scalars

Commitment `[0, 1]`

Secondary Scalars

Recovery Margin `[0, 1]`

Gesture Recognition Pipeline

cc-gesture: Anticipation-Based Neighbor Voting Classification

1. INPUT

AnticipationPacket

regime_embedding

commitment, uncertainty

transition_pressure

→

2. HNSW QUERY

Query MotionPhraseLibrary

Find `k=10` neighbors

O(log n) lookup

→

3. VOTE AGGREGATION

Rank weight: `(1-decay)^i`

Distance weight: `1/(1+d)`

Sum votes per label

→

4. CONFIDENCE

vote_conf = best/total

ant_conf = C × (1-U)

final = average

→

5.

C

A

E

🚦 Gating Thresholds

Min Confidence	Commitment Gate	Transition Pressure	Cooldown
≥ 0.6	≥ 0.7	≥ 0.5	500ms
Must pass to proceed	Motion is committed	Futures collapsing	Between same gesture

Vote Aggregation Algorithm

```
for (i, neighbor) in neighbors.iter().enumerate() {
    let rank_weight = (1.0 - decay).powi(i as i32);
    let distance_weight = 1.0 / (1.0 + neighbor.distance);
    let weight = rank_weight * distance_weight;

    for label_id in &neighbor.label_ids {
        vote_map.entry(label_id)
            .and_modify(|v| v.add_vote(weight))
```

DELL: Dual-Equilibrium Latent Learning

Fast (60Hz) + Slow (~2.5Hz) Equilibrium Dynamics

⚡ Fast Equilibrium

Micro-dynamics, frame-level

60 Hz

x* (Equilibrium Point)
Attractor position [4 floats]

ψ (Latent Energy)
Sum of squared velocities

ϕ (Latent Phase)
Oscillatory phase

Residual
Distance from equilibrium

Coherence
exp(-residual / scale)

Limb Energy
Per-limb kinetic energy

🌊 Slow Equilibrium

Macro-dynamics, beat-level

~2.5 Hz

Macro Energy
Envelope of fast energy

Periodicity
Autocorrelation peak [0,1]

Phase
Position within beat [0,1]

Section State
Entry→Build→Peak→Release

DELLCoordinator

Manages both equilibria and couples their outputs

Fast State + Slow State → DELLState

📤 Output: DELLState

x_star	psi	phi	limb_energy

Sensor Fusion: Extended Kalman Filter

cc-collection: Multi-Sensor Fusion with 13-Dimensional State

📐 State Vector (13 dimensions)

p	v	q	ω
Position	**Velocity**	**Quaternion**	**Angular Velocity**
3D world position	Linear velocity	Orientation (w,x,y,z)	Rotation rate
`[3 floats]`	`[3 floats]`	`[4 floats]`	`[3 floats]`

EKF Algorithm

1. Predict

$$\hat{x} = f(x, dt)$$

$$\hat{P} = F \cdot P \cdot F^T + Q$$

Propagate state forward using motion model

2. Update

$$K = \hat{P} \cdot H^T \cdot (H \cdot \hat{P} \cdot H^T + R)^{-1}$$

$$x = \hat{x} + K \cdot (z - h(\hat{x}))$$

$$P = (I - K \cdot H) \cdot \hat{P}$$

Incorporate measurements with Kalman gain

📡 Sony Mocopi

27 IMU sensors

Quaternion + position per bone

Rate: 30-60 Hz

Weight: `0.85`

📷 MediaPipe

Camera landmarks

3D positions (normalized)

Rate: 30 Hz

Weight: `0.15`

📱 iPhone/Watch

CoreMotion IMU

Accelerometer + gyroscope

Rate: 60 Hz

Weight: `variable`

RAG++ Architecture

5D Trajectory Memory Fabric for TrajectoryOS

📐 TrajectoryCoordinate5D

D	S	H	T	C
Depth	**Sibling Order**	**Homogeneity**	**Temporal**	**Complexity**
Session nesting level	Position among peers	Neighbor similarity	Turn index (normalized)	Embedding variance

🦀 Rust Core

HNSWIndex
Hierarchical navigable small world graph for O(log n) ANN search

IRCPPropagator
Inverse Ring Contextual Propagation for attention weights

SalienceScorer
Compute importance based on trajectory position

TrajectoryCoordinate5D
5D coordinate computation and geodesics

🐍 Python ML

CognitiveTwin
ML model learning user reasoning patterns

HybridTrainer
Batch + incremental training pipeline

GlobalStyleSignature
EMA-based evolving style embedding

MemoryRetriever
Trajectory-aware semantic search

🗄️ Storage

Supabase
PostgreSQL + pgvector extension

memory_turns
Unified knowledge fabric table

claude_prompts
Claude interaction logs

Vector Search
cosine_distance for embeddings

Orbit: Project Orchestration

Session Management & RAG++ Proxy for TrajectoryOS

🔌 Clients

Cursor IDE
MCP integration
mcp_server.py

Claude Desktop
MCP tools
rag_search

Next.js Frontend
Visualization UI
Three.js

cc-mcs-headless
Motion processor
Rust

🌐 Orbit Server (Axum)

Project Registry
In-memory project state

Session Manager
Active session tracking

RAG++ Proxy
Route to FastAPI backend

Context API
PROJECT_CONTEXT.md

🔗 Backend Services

RAG++ (FastAPI)
Memory retrieval & training
Cloud Run

Supabase
PostgreSQL + pgvector
Persistence

CognitiveTwin
Style learning
ML Model

📡 API Routes

Project Management

`GET` `/api/projects`

Session Management

`GET` `/api/sessions`

RAG++ Proxy

`POST` `/api/rag/search`

Audio Engine: Beat Scheduling & DSP

cc-conductor: Real-Time Audio Processing Pipeline

Motion
LatentState
<1ms

→

Conductor
Tempo tracking
<100µs

→

Scheduler
Beat quantize
<500µs

→

Timeline
Ableton Link
<1ms

→

DSP
Audio callback
<6ms

🎼 cc-conductor

TempoTracker
Estimates BPM from periodicity
range: 60-180 BPM

PhaseEstimator
Beat phase [0, 1] from motion
precision: ±10ms

EnergyMapper
Latent energy → intensity
smoothing: EMA α=0.1

⏱️ Beat Scheduler

ScheduledAction
Target beat + callback
schedule(beat, action)

QuantizeMode
Off / Beat / Bar / Phrase
quantize_to(mode)

BeatTimeline
Ableton Link sync
link.beat_at_time()

🔊 Audio DSP

AudioEngine
Real-time callback
48kHz / 128 samples

Lock-Free
No allocations in callback
ring buffer messaging

Priority
Real-time thread priority
SCHED_FIFO 99

🎯 Quantization Modes

Off
Immediate execution

Beat
Next beat boundary

Bar
Next bar (4 beats)

Phrase
Next phrase (8/16 bars)

Why cc-collection Before cc-window-aligner?

Understanding the Pipeline Ordering Decision

🤔 The Core Question

Why must sensor fusion (cc-collection) happen BEFORE windowing (cc-window-aligner)?

❌ Wrong: Window → Then Fuse

1. Collect Windows Per Device

Mocopi: frames at t=[0, 33, 66...]ms
MediaPipe: frames at t=[0, 50, 100...]ms

↓

2. Try to Align Windows

Windows don't align! Different rates!
Mocopi window: [0-1000ms]
MediaPipe window: [17-1017ms]

↓

3. Fusion Fails

Can't fuse misaligned windows
Lost temporal coherence

✅ Right: Fuse → Then Window

1. Stream All Sensors

All frames → single EKF stream
Handle arrivals as they come

↓

2. EKF Produces Unified State

Output: 50Hz unified skeleton
All sources fused in real-time

↓

3. Window the Fused Stream

Clean 50-frame windows
Perfect temporal alignment

🎯 Why This Matters