都市のAI化 — 分散ローカルLLMによる空間知能アーキテクチャ

SOMS (Symbiotic Office Management System)

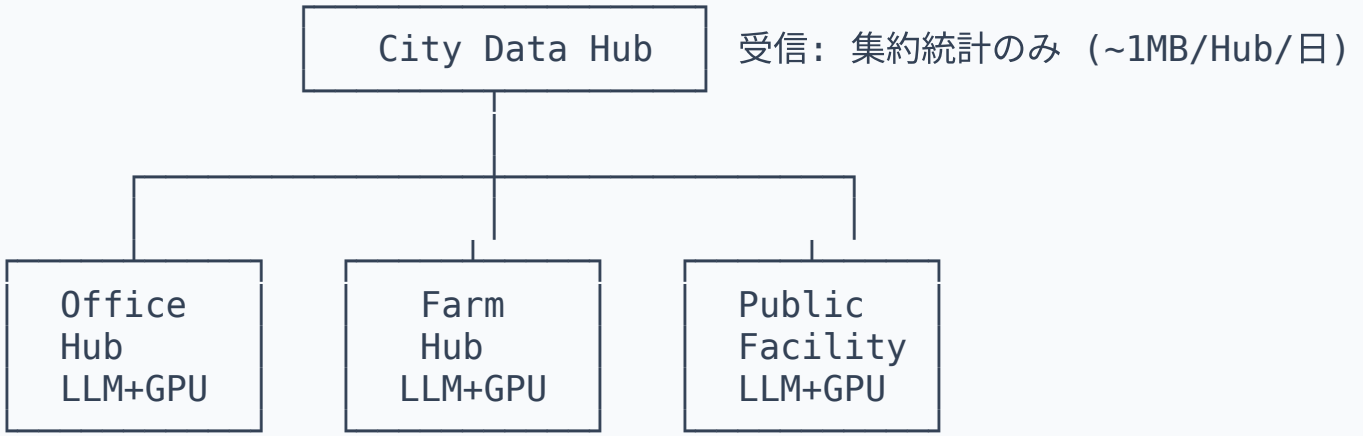
Core Hub Phase 0 技術実証

現行スマートシティの構造的課題

標榜	実態
データ駆動の都市経営	センサーデータはクラウドベンダーに集約され、自治体はAPI経由で自身のデータを購入する構造
AIによるリアルタイム最適化	推論サーバーはクラウド上。ネットワーク障害時に全機能が停止する単一障害点
市民のための技術	カメラ映像の保管・アクセス権限が外部企業に帰属。データ主権が自治体にない
リアルタイム分析	クラウド往復による数百ms〜秒の遅延。エッジ処理との差は桁違い

共通する構造的問題: データの生成場所（建物）と処理場所（クラウド）の物理的分離。ローカルGPUによるLLM推論が実用水準に達した現在、この前提を見直す時期にある。

Core Hub アーキテクチャ: 建物単位の自律AI拠点



各建物にGPU1台のサーバーを配置し、センサーとカメラの全データをローカルで処理する。Hub間で生データの交換は行わない。システムプロンプト（行動原則）とセンサー構成の差し替えだけで、オフィス・農場・店舗・公共施設に展開できる。ネットワーク切断時も72時間の自律動作を継続する設計。

三層データ処理モデル: 50,000:1 圧縮

層	データ量 (1拠点/日)	処理内容
Layer 0: 物理信号	~50 GB	カメラ映像(VGA 3fps) + 全センサー生値。RAM上で処理、ディスク保存なし
Layer 1: Core Hub	~500 MB	ローカルLLM + YOLOによるリアルタイム解釈。構造化JSON・イベントログ・判断記録として保持。元データの99%を破棄
Layer 2: City Data Hub	~1 MB	1時間集約の統計値のみ受信。気温平均、CO2ピーク、在室率、タスク完了数

GDPRコンプライアンスの観点で注目すべき点: カメラ映像はYOLOの推論完了と同時に破棄される。Hub本体を物理的に撤去すれば、その拠点の全生データが消失する。「データを送信しない」が最も確実なプライバシー保護となる。

SOMS: Core Hub Phase 0 の全体構成

SOMS はオフィス1部屋を対象にした最初のCore Hub実装。GPU1台、Docker 11サービスで全機能が稼働する。

生体アナロジー	コンポーネント	技術	実装状況
脳	ReAct認知ループ	Qwen2.5 14B (Ollama, ROCm)	5ツール, 最大5反復, 3層安全機構
神経系	メッセージバス	MQTT (Mosquitto) + MCP (JSON-RPC 2.0)	QoS 1, ESP32実機通信済み
視覚	コンピュータビジョン	YOLOv11 (検出+姿勢推定)	3モニター, 4層活動分析, カメラ自動発見
触覚	環境センサー	BME680, MH-Z19C 他6種ドライバ	SensorSwarm Hub+Leaf (4種トランスポート)
声	音声合成	VOICEVOX (Speaker 47)	拒否ストック100件事前生成, 4トーン
経済系	タスク報酬	複式簿記 (PostgreSQL)	デマレッジ2%/日, 5%焼却, PWAウォレット

設計方針: Node-RED・LangChain・Kubernetesを排し、Python + MQTT による純粋なイベント駆動。LLMがシステムのソースコードを直接理解できる透明性を優先。

4層アーキテクチャと6つの設計原則

原則	内容
自律性 > 自動化	IF-THENルールではなく、LLM推論による目的関数指向の判断
憲法的AI + 安全弁	言語原則(システムプロンプト) + ハードコード制約(Sanitizer)の二重構造
イベント駆動	Python + MQTT のみ。重量級ミドルウェアを排除
コードファースト	ビジュアルプログラミング不使用。全ロジックをコードで記述
善意モデル	認証なし・性善説。報酬は緊急度と感謝のシグナル
ローカルファースト	全処理オンプレミス。クラウド送信ゼロ

ReAct 認知ループ

30秒の定期サイクル、またはMQTTイベント到着時にバッチ遅延3秒で起動。

```
[トリガー] ← MQTT イベント（3秒バッチ） or 30秒定期
    ▼
[THINK]  WorldModelからゾーン状態取得 → LLMに送信
         （温度, CO2, 湿度, 照度, 在室人数, 活動レベル, 姿勢, 直近イベント50件）
    ▼
[ACT]    LLMがツール呼び出し（0〜複数） → Sanitizer安全検証 → 実行
    ▼
[OBSERVE] 実行結果をLLMにフィードバック → 追加行動を判断
    ▼
[完了]   追加行動不要と判断 → サイクル終了（最大5反復で打ち切り）
```

パラメータ	値
最大反復回数	5回/サイクル
LLMタイムアウト	120秒
MCPデバイス応答	10秒
最小サイクル間隔	25秒
イベントバッチ遅延	3秒

正常時は「何もしない」で1反復で終了する。

WorldModel: センサーフュージョンとイベント検知

```
WorldModel
├── zones: Dict[zone_id] → ZoneState
│   ├── environment: { temperature, humidity, co2, illuminance }
│   ├── occupancy: { person_count, activity_class, posture_status }
│   ├── devices: Dict[device_id] → { power_state, specific_state }
│   └── events: List[Event]  # 直近50件
└── sensor_fusion: 指数減衰加重平均
```

センサーフュージョン: 複数センサーの読み取り値を指数減衰加重平均で統合。

センサー	半減期	設計意図
温度	120秒	空調効果の時定数に合わせた平滑化
CO2	60秒	窓開閉・人の出入りへの追従性確保
在室人数	30秒	YOLO検出結果の即時反映

イベント検知: 状態変化を検出し、クールダウン付きで発火。

イベント	条件	クールダウン
CO2閾値超過	> 1000ppm	10分
温度急変	3度以上/短時間	なし
長時間座位	同姿勢30分以上	1時間

LLMツール: 5種の機能と使い分け

ツール	機能	副作用
`create_task`	ダッシュボードにタスク掲示 + 音声告知	人間への依頼発生
`send_device_command`	MCP over MQTT経由のエッジデバイス制御	物理デバイス操作
`speak`	音声のみのアナウンス (70文字以内)	発話のみ、ダッシュボードに記録なし
`get_zone_status`	ゾーンの詳細状態をJSON取得	なし (読み取り専用)
`get_active_tasks`	既存タスク一覧取得	なし (読み取り専用)

ツール選択の指針 (システムプロンプトに定義):

状況	選択するツール	理由
CO2 1000ppm超 + 在室者あり	`create_task`	物理的な換気作業が必要
30分間同一姿勢	`speak` (caringトーン)	助言であり作業依頼ではない
センサー値の急変	`speak` (humorousトーン)	改竄の可能性を軽く指摘
全指標正常	何もしない	speakも禁止。無意味な報告は行わない

タスクスケジューリング: 文脈対応ディスパッチ

タスクは作成と配信が分離されており、状況に応じて配信タイミングを制御する。

条件	判定
緊急度4 (CRITICAL)	即時配信
緊急度3 + ゾーンに在室者あり	即時配信
ゾーンに在室者なし	キュー待機 (在室者検知まで保留)
22時以降 + 低緊急度	翌朝まで保留
キュー滞留24時間超過	強制配信

重複検知 (2段階):

- Stage 1: title + location の完全一致
- Stage 2: zone + task_type の一致

多層安全機構

```
LLM出力 → [憲法的AI] → [Sanitizer] → [行動履歴] → 物理デバイス
```

第1層: 憲法的AI (システムプロンプトの行動原則)
安全最優先 / 報酬は難易度比例 (500〜5000) / タスク作成前に既存確認必須 / 正常時は何もしない / 個人特定情報を扱わない

第2層: Sanitizer (ハードコードされた物理限界)

パラメータ	許容範囲
温度設定	18〜28℃
ポンプ動作	最大60秒
報酬上限	5000
緊急度	0〜4
タスク作成レート	10件/時間

第3層: 行動履歴 + レート制限

- 重複ツール呼び出しフィルタ + speakレート制限 (1回/サイクル)
- サイクル間隔の強制 (最小25秒)

MCP over MQTT: IoT向けプロトコル設計

MCP (Model Context Protocol) は通常HTTP/stdio上で動作するが、IoT環境にはMQTTが適合する。

比較項目	HTTP	MQTT
LLM推論(秒) vs デバイス応答(ms〜分)の時間差	タイムアウトリスク	ブローカーが吸収
ESP32での実装負荷	HTTPスタックが重い	MQTTクライアントは軽量
再送制御・デバイス生死監視	自前実装が必要	QoS + LWT が標準装備

トピック設計:

```
# テレメトリ (per-channel)
office/{zone}/sensor/{device_id}/{channel} → {"value": X}

# MCP制御 (JSON-RPC 2.0)
mcp/{device_id}/request/call_tool  → {"jsonrpc":"2.0","method":"call_tool","params":{...},"id":"uuid"}
mcp/{device_id}/response/{req_id}  → {"jsonrpc":"2.0","result":{...},"id":"uuid"}

# ハートビート (60秒間隔)
{topic_prefix}/heartbeat → {"status":"online","uptime":N}
```

SensorSwarmのLeafはドット表記 (`swarm_hub_01.leaf_env_01`) でWorldModelへの統合にコード変更不要。Phase 3までプロトコル変更なし。

SensorSwarm: Hub+Leaf 2層エッジネットワーク

WiFi接続のセンサーノードのみではAP帯域・チャネル干渉がボトルネックとなる。SensorSwarmはHub+Leafの2層構造により、WiFiスタックを持たないLeafノードでの低消費電力運用を実現する。

```
[Brain] ←MQTT→ [SwarmHub (ESP32, WiFi+MQTT)]
                    ├── ESP-NOW → [Leaf: 温湿度] 無線200m, バッテリー駆動
                    ├── UART    → [Leaf: PIR+照度] 有線, Pi Pico
                    ├── I2C     → [Leaf: ドアセンサー] 有線, ATtiny
                    └── BLE     → [Leaf: 未実装, nRF54ターゲット]
```

項目	Hub	Leaf
WiFi	あり (MQTT接続)	不要 (Hub経由)
通信プロトコル	MCP (JSON-RPC 2.0)	バイナリ (5-245B, MAGIC `0x53`, XOR checksum)
デバイスID	`swarm_hub_01`	`swarm_hub_01.leaf_env_01` (ドット表記)
トランスポート	WiFi	ESP-NOW / UART / I2C / BLE (4種)

HubがLeafのバイナリメッセージをデコードし、per-channel MQTTとして再発行。Brain側からはLeafが独立したセンサーとして見える。

エッジデバイス: ファームウェアとドライバ

MicroPython ファームウェア (量産向け)

種別	特徴
unified-node	`config.json` でセンサーを宣言的に定義。SensorRegistryによる遅延初期化・アドレス自動検出・エラー分離
sensor-02	BME680 + MH-Z19C 専用 (レガシー)

C++ / PlatformIO ファームウェア (カメラ向け)

camera-node: ESP32 WROVER + OV2640

共通ライブラリ `soms_mcp.py`: WiFi+MQTT接続管理 (自動再接続) / config.json読み込み / per-channelテレメトリ送信 / MCPツール登録 (`register_tool(name, callback)`) / ハートビート (60秒)

センサードライバ 6種:

ドライバ	バス	計測チャネル
BME680	I2C	temperature, humidity, pressure, gas_resistance
MH-Z19C	UART	co2
DHT22/DHT11	GPIO	temperature, humidity
PIR	GPIO	motion
BH1750	I2C	illuminance

Perception: プラガブルモニター

モニターはYAML設定 (`config/monitors.yaml`) で宣言的に定義。

モニター	頻度	解像度	処理内容
OccupancyMonitor	5秒	QVGA	yolo11s.pt による人物検出。在室カウント
ActivityMonitor	3秒	VGA	2段階推論: 人物検出 → 人がいれば姿勢推定 (yolo11s-pose.pt)
WhiteboardMonitor	60秒	VGA	Cannyエッジ検出によるホワイトボード汚れ度合い判定

カメラ自動検出 (3段階パイプライン):

1. ネットワーク上のカメラポート (80, 81, 554, 8554) を非同期TCP接続でスキャン
2. 候補URLパターンでOpenCV接続テスト
3. YOLOで「映像が実際に映っている」ことを確認

活動分析: 4層時間バッファ

生フレームから長時間の姿勢傾向まで、段階的に時間解像度を落としながら保持する。

層	時間解像度	保持期間	検出内容
Tier 0 (raw)	毎フレーム	60秒	瞬間姿勢
Tier 1	10秒ごと	10分	短期の動作パターン (歩行/着座/起立)
Tier 2	1分ごと	1時間	中期傾向 (持続的な着座など)
Tier 3	5分ごと	4時間	長時間姿勢追跡

姿勢正規化: 位置・スケール不変の骨格特徴量で比較。アンカー: 腰中点、スケール: 肩幅。カメラからの距離や映り方に依存しない。

Tier 2 で30分以上 `mostly_static` が継続 → 座位イベント発火。Brain は `speak` ツール (caringトーン) で通知。クールダウン1時間。

Dashboard: ゲーミフィケーションUIとタスクライフサイクル

React 19 + TypeScript + Vite 7 + Tailwind CSS 4 + Framer Motion。認証不要のキオスクモード。

タスクカード構成: タイトル / 場所バッジ / 報酬バッジ (金色、「最適化承認スコア」表記) / 緊急度インジケータ (色分け) / 受諾・完了・無視ボタン

ポーリング: タスク一覧 5秒 / 音声イベント 3秒 / 完了タスク 5分後フェードアウト / 表示上限 10件

タスクライフサイクル:

```
Brain: create_task → Backend: POST /tasks/ (2段階重複検知) → タスク作成 + 音声合成
→ [Pending] 告知音声自動再生
→ [Accepted] PUT /tasks/{id}/accept → 受諾音声再生
→ [Completed] PUT /tasks/{id}/complete → 完了音声 + Wallet報酬支払い
```

Backend: FastAPI + SQLAlchemy async ◦PostgreSQL 16 (asyncpg) / SQLite (aiosqlite) フォールバック。タスクモデル19カラム (bounty, urgency, voice関連, assignment, completion)。

Voice サービス: VOICEVOX連携と拒否ストック

音声生成フロー: テキスト → VOICEVOX `/audio_query` (韻律) → `/synthesis` (WAV 24kHz) → pydub (MP3 64kbps) → `/audio/{filename}` 配信。Speaker ID 47: ナースロボ_タイプT。

場面	処理方式	レイテンシ
タスク告知	`announce_with_completion`: LLMテキスト生成 + VOICEVOX。告知と完了の2音声を同時生成	4-6秒
タスク受諾	`synthesize`: 定型フレーズの即時合成	1-2秒
タスク完了	告知時に同時生成済みの completion 音声を再生	即時
タスク無視	rejectionストックから取得	<100ms
健康助言・警告	Brain `speak` → `synthesize`	1-2秒

拒否ストック: アイドル時にバックグラウンドでLLMテキスト生成 + VOICEVOX合成。最大100件のMP3を備蓄、残80件以下で補充開始。6バリエーション (傷ついた / 皮肉な等)。50文字以内。

トーン: caring (健康助言) / alert (環境警告) / humorous (軽い指摘) / neutral (一般) 。speak 70文字制限。

Wallet: 複式簿記ベースの信用台帳

PostgreSQL上の複式簿記。1取引 = DEBIT + CREDIT の2行、トランザクションIDによる冪等性保証。

モデル	役割
Wallet	ユーザー残高。user_id=0 はシステムウォレット (通貨発行元、負残高許容)
LedgerEntry	複式仕訳。種別: INFRASTRUCTURE_REWARD / TASK_REWARD / P2P_TRANSFER
Device	デバイスXPトラッキング。topic_prefixによるゾーンマッチ
RewardRate	デバイス種別ごとの報酬レート (llm_node: 5000/h, sensor_node: 500/h, hub: 1000/h)
SupplyStats	通貨発行総量・焼却量・流通量

報酬フロー: タスク完了 → POST /transactions/task-reward → System Wallet → User Wallet 振替 → SupplyStats更新

XP乗数: `multiplier = 1.0 + (device_xp / 1000) × 0.5` 上限3.0x

デフレ機構: 手数料5%焼却 + デマレッジ2%/日。実用通貨ではなくコミュニティ内ゲーム用ポイントとして設計。

ウォレットPWA: 残高表示 / QRスキャン報酬受取 / P2P送金 / 取引履歴 (React Router v6)

E2Eデータフロー: CO2検知からタスク完了まで

```
[T+0s]   ESP32 sensor-02: CO2 = 1050ppm
         → MQTT: office/kitchen/sensor/co2_01/co2 → {"value": 1050}

[T+0s]   WorldModel: CO2値更新、co2_threshold_exceeded イベント発火

[T+3s]   ReActサイクル開始（イベントバッチ遅延完了）
         → LLMに送信: "kitchen: CO2 1050ppm, 3人在室, activity: moderate"

[T+4s]   LLM Think: "CO2が高い。在室者がいるので換気が必要"
         LLM Act: get_active_tasks() → 既存の換気タスクなし
                  create_task(title="キッチンの換気", bounty=1500, urgency=3)

[T+5s]   Sanitizer: bounty ≤ 5000 ✓, urgency ∈ [0,4] ✓
         → POST /tasks/ (2段階重複検知通過) → タスク作成
         → VoiceService: LLMテキスト生成 → VOICEVOX → MP3

[T+10s]  Frontend: ポーリングで検出 → カード表示 + 音声自動再生

[T+??]   完了報告 → completion音声 → Wallet: 1500ポイント振替
```

データ主権の技術的保証

データ分類	例	処理場所	保存	外部送信
映像	カメラRGBフレーム	Core Hub RAM	なし	不可
音声	マイク波形	Core Hub RAM	なし	不可
生テレメトリ	温度 23.5℃	Core Hub	Event Store (90日)	不可
LLM判断ログ	「CO2高→タスク作成」	Core Hub	Event Store (90日)	不可
集約統計	1時間平均気温	Core Hub	Data Mart	City Hub送信
タスク記録	完了タスク一覧	Core Hub	Data Store	匿名化送信可

暗号化: Sensor→Hub: WPA3 + MQTT over TLS / Hub内部: Docker network isolation / Hub→City: TLS 1.3 + mTLS

ネットワーク障害時はローカルキューに蓄積し、復旧後に一括送信。自律動作は通信状態に依存しない。

City Data Hub: 都市規模のパターン抽出

City Data Hub は生データを処理しない。各Core Hubの Data Mart (集約済み) のみを受信する。

```
// Data Mart送信例 (1時間ごと)
{
  "hub_id": "hub_office_01", "period": "2026-02-13T14:00:00/PT1H",
  "zones": { "kitchen": { "avg_temperature": 24.3, "max_co2": 1050,
    "avg_occupancy": 2.8, "comfort_index": 0.72 } },
  "tasks_created": 1, "tasks_completed": 1, "llm_cycles": 120,
  "device_health": { "online": 5, "offline": 0 }
}
```

City Data Hub が抽出するパターンの例:

- 複数地区のCO2同時上昇 → 広域大気汚染の検知
- 月曜午前の全オフィス在室率急上昇 → 通勤パターンの変化
- Hub単体のセンサー精度劣化 → メンテナンス必要性の検知
- 季節・地区ごとのエネルギー消費パターン → 資源配分の最適化

都市の呼吸パターン

各Hubが1時間平均のCO2値を送信するだけで、都市規模の人流パターンが可視化される。

```
Hub-A (オフィス街):  CO2ピーク 9:00, 13:00
Hub-B (商業施設):    CO2ピーク 11:00, 15:00, 19:00
Hub-C (住宅街):      CO2ピーク 7:00, 20:00
```

時空間分析により推定される人流:
住宅街 (朝7時) → オフィス街 (9時) → 商業施設 (昼・夕方) → 住宅街 (夜20時)

特定の風向き条件下で特定地区のCO2が異常上昇するパターンを検出できれば、換気設備の配置計画や緑地計画への入力データとなる。送信データ量は数バイト/Hub/時間。

領域特化Hub: 同一アーキテクチャの展開先

Hub種別	センサー構成	LLMの行動原則	タスク例
オフィス (SOMS)	温湿度, CO2, カメラ	快適性・健康・生産性の最適化	換気, 清掃, 備品補充
農業	pH, EC, 水温, 照度	水耕栽培の生育環境維持	養液調整, 収穫判断
水槽	水温, pH, TDS	水生生物の環境管理	給餌, 水換え
店舗	人流カメラ, 温湿度	顧客体験と在庫の最適化	品出し, 陳列変更
公共施設	騒音, 振動, 気象	安全管理と設備保全	点検, 修繕
屋外環境	気象, 大気質, UV	環境監視と市民への情報提供	警報発令, データ記録

全Hubが共通のMCP over MQTTとData Martスキーマを使用。City Data Hub は異種Hubのデータを統一的に集約・分析可能。

SOMS → Core Hub: 進化パス

Phase 0 の設計で変更不要な部分:
MCP over MQTT / ReActループ / WorldModel + センサーフュージョン / 憲法的AI / per-channel テレメトリ / config.json + 共通ライブラリ / タスク経済

追加が必要なコンポーネント:

コンポーネント	Phase 1	Phase 2
Event Store	TimescaleDB 導入	レプリケーション追加
Data Lake	Event Store = Data Lake	保持期間ポリシー追加
Data Mart	1時間集約バッチジョブ	標準スキーマ + Hub間共有
Hub間通信	N/A	MQTT Bridge or HTTPS (mTLS)
Hub管理	N/A	登録・認証・ヘルスチェック
OTA更新	ESP32 OTA基盤	全ノード一括更新
LoRa対応	実験的導入	屋外ノード標準

ロードマップと成功指標

Phase	内容	規模	成功基準
0 (現在)	単一オフィスでE2Eフロー実証	1 Hub	24時間連続稼働 / E2Eレイテンシ <10秒 / 適切判断率 >80%
1	多ゾーン化 + Data Lake	1 Hub, 10+ノード	同時接続10+ / Data Lake 30日蓄積 / 月5件以上の洞察
2	複数Core Hub + City Data Hub	2-3 Hub	Hub間通信 99.9% / 孤立72時間 / クロスHub相関 3件以上
3	都市展開 + ゼロタッチ配備	10+ Hub	外部クラウド送信 0 bytes / 都市計画への入力 1件以上

Core Hub ハードウェア要件 (Phase 2以降):

	最小構成	推奨構成
GPU	16GB VRAM (量子化LLM)	32GB VRAM (フル推論+Vision)
CPU / RAM	8コア / 32GB	16コア / 64GB
Storage	500GB SSD	2TB NVMe
消費電力	150W (アイドル)	350W (推論時)
設置面積	A4用紙程度 (ミニPC構成)	ラックマウント 1U

Phase 0 達成状況

コンポーネント	状態	詳細
ReAct認知ループ	稼働	5ツール, 最大5反復, 3層安全機構
WorldModel	稼働	指数減衰加重平均, 4種イベント検知
MCP over MQTT	稼働	JSON-RPC 2.0, ESP32実機通信済み
Perception	稼働	YOLOv11 検出+姿勢, 4層活動分析, カメラ自動検出
SensorSwarm	稼働	Hub+Leaf, ESP-NOW/UART/I2C, バイナリプロトコル
Dashboard	稼働	React 19 キオスク, 音声統合, 2段階重複検知
Voice	稼働	VOICEVOX, 拒否ストック (最大100)
Wallet	稼働	複式簿記, デマレッジ, 5%焼却, デバイスXP
Wallet PWA	稼働	残高/QR/送金/履歴
仮想テスト環境	稼働	Mock LLM + Virtual Edge + Virtual Camera
E2E統合テスト	稼働	7シナリオ
Data Lake / Data Mart	未実装	Phase 1 で対応

パフォーマンス実測値

AMD RX 9700 (RDNA4) + Qwen2.5 14B (Q4_K_M) — GPU サーバー1台構成

指標	値
LLM推論速度	~51 tok/s
正常時応答	3.3秒
ツール呼び出し応答	6.6秒
エラー率	0% (12リクエスト)
Dockerサービス数	11
月額クラウド費用	$0

30秒の認知サイクルに対して3〜7秒で応答完了。処理能力に十分な余裕がある。

技術スタック

層	技術	補足
LLM	Qwen2.5 14B (Q4_K_M), Ollama (ROCm)	Mock LLM: キーワードマッチでtool call生成
Vision	YOLOv11 (yolo11s.pt + yolo11s-pose.pt)	物体検出 + 骨格推定 (17キーポイント)
Backend	Python 3.11, FastAPI, SQLAlchemy async	Pydantic 2.x, loguru, aiohttp
Frontend	React 19, TypeScript, Vite 7, Tailwind CSS 4	Framer Motion, Lucide icons
Voice	VOICEVOX (Speaker 47), pydub	LLMテキスト生成 + 拒否ストック
Messaging	MQTT (Mosquitto), paho-mqtt 2.x	MCP (JSON-RPC 2.0) over MQTT
Edge (Python)	MicroPython, ESP32-C6/S3	unified-node + SensorSwarm (Hub+Leaf)
Edge (C++)	Arduino, ESP32 WROVER	OV2640カメラノード
Database	PostgreSQL 16 (asyncpg)	SQLite (aiosqlite) フォールバック
Container	Docker Compose (11 services)	ROCm対応
GPU	AMD RX 9700 (RDNA4)	HSA_OVERRIDE_GFX_VERSION=12.0.1

デプロイメント: 2つのモード

シミュレーション (GPU・ハードウェア不要)

```
./infra/scripts/start_virtual_edge.sh
```

Mock LLM + Virtual Edge (ランダムウォークセンサー) + Virtual Camera (RTSPテストパターン)

プロダクション (AMD ROCm GPU + 実ハードウェア)

```
docker compose -f infra/docker-compose.yml up -d --build
```

Ollama + Qwen2.5:14b + YOLOv11 + 実ESP32

サービス	ポート	コンテナ名
Frontend (nginx)	80	soms-frontend
Backend API	8000	soms-backend
Mock LLM	8001	soms-mock-llm
Voice	8002	soms-voice
Wallet	8003	soms-wallet
PostgreSQL	5432	soms-postgres

まとめ

SOMS は、ローカルLLM + IoT + 人間経済で建物を自律管理する Core Hub アーキテクチャの Phase 0 実装。

GPU1台のサーバーに11のDockerサービスを載せ、センサーデータの収集からLLM推論、タスク生成、音声通知、報酬支払いまでの全フローがローカルで完結する。外部へのデータ送信はゼロ。

同一アーキテクチャを接続センサーとシステムプロンプトの差し替えだけで多領域に展開し、City Data Hub による集約統計の分析で、単一拠点では見えない都市規模のパターンを抽出する。

Phase 0 で設計の妥当性を確認した要素 — MCP over MQTT, ReActループ, WorldModel, 憲法的AI, per-channelテレメトリ, タスク経済 — はPhase 3の都市展開まで変更不要。

GitHub: `Office_as_AI_ToyBox`