協調型 AI エージェントによる
様々なタスクの効率化

川越　航太

神奈川工科大学

論文要旨

近年，スマートホームやWebサービスの普及で生活の利便性は向上したが，各サービスが分断され，ユーザーは複数アプリでの詳細な操作を強いられている．生成 AI による自然言語操作への期待は高いが，既存手法は長期的な嗜好や文脈を保持する仕組みが不足しており，曖昧な指示への応答や，Web と物理空間を横断した一貫性のある自動化が困難であった．本研究では，固有の「長期・短期メモリ」を持つオーケストレータを中核とし，Web操作，IoT制御，知識検索，スケジュール管理を専門とする4つのエージェントを統合した「協調型マルチエージェントプラットフォーム」を開発した．特徴として，Model Context Protocol (MCP) により機能を標準化し拡張性を確保した．オーケストレータは，対話履歴や属性情報（住所，健康状態等）に基づき曖昧な発話を補完，具体的タスクへ分解し各エージェントへ配分する．また，IoT制御ではAPI経由の高性能LLMとエッジデバイス（Jetson 等）の軽量 LLM を組み合わせた階層的推論を採用し，プライバシー保護と即応性を両立させた．10種類のシナリオを用いた評価実験の結果，メモリ機能の有効化により，ベースライン比でタスク達成スコアが約1.7倍向上し，確認質問が減少した．特に健康状態を考慮したレシピ提案や居住地に基づく情報検索など，高度なパーソナライズを実現した．さらに，数BパラメータのエッジLLMでも，適切なプロンプトにより抽象的な指示を物理制御コマンドへ正確に変換可能であることを実証した．結論として，本研究は最小限の指示で意図を理解し，デジタルとフィジカルを横断して最適な支援を提供する自律型エージェントシステムの有用性を示した．

目次

1. まえがき

1.1 本研究の背景

スマートホーム機器や Web サービスは多様化しており，個別最適なアプリやアカウントがサービスごとに分断されがちである．例えば企業ごとに EC・モバイルオーダーを利用するには，メールアドレス・パスワード・住所等の登録や画面操作を都度行う必要がある．IoT の世界でも，音声や簡易ルール（IFTTT に代表されるルールベースのトリガー）中心の制御が主で，曖昧な依頼（「明日の朝，晴れそうなら」）や複合条件（「在室かつ高湿度なら除湿，外出なら消灯」）の理解・最適化は限定的である．さらに，個人情報の再入力による手間や入力ミス，機器の種類による操作性の違い，そして各サービスをまたいだ自動実行における確実性の担保が課題として残る．

一方，生成 AI の台頭により，自然言語で意図を解釈し，複数のツールを段階的に組み合わせる「ツール使用型エージェント」が実用段階に入りつつある[1]．これにユーザー固有の嗜好・履歴・制約を保持するパーソナライズドな記憶（メモリ）と，Web・IoT 双方を扱える実行基盤，そして高度な判断支援を加えることで，従来の自動化の限界（曖昧性処理・連携範囲）を超えられる可能性がある．例えば，Home Assistant でも LLM 連携の取り組みが進んでいる[2]．本研究はこの観点から，Web 上の AI エージェントと物理デバイスを一貫した方針で束ねる統合型システムを検討する．

1.2 本研究の目的

本研究の目的は，個人に最適化されたメモリを備えたオーケストレータの下で，Browser エージェントや IoT エージェントなどの4種類のエージェントを統合し，少ない自然言語指示からでも文脈を補完して確実にタスクを自動実行できることを示すことである．具体的には以下を目指す．

(1) **統合アーキテクチャの設計・試作** RAG ベースの Life-Style エージェント，Browser エージェント，IoT エージェント，Scheduler エージェント，メモリ付きオーケストレータからなる構成を定義し，API・権限・データフローを明確化する．

(2) **パーソナライズと曖昧性解消** ユーザーの属性・履歴・好み・制約（例：住所，趣味趣向，職種）をメモリに保持し，情報の乏しい指示でも最適な具体手順へ落とし込む．

(3) **代表ユースケースでの有効性評価** - 保存された個人メモリに基づきながら，天候・予定に基づく照明制御，Web サイトでのフォーム自動入力を行う． - 評価のための架空のペルソナを複数作成して，複数のサンプルタスクを実行する． - 指標：各シナリオのサブゴール達成度，エージェントからの質問回数（人間介入度）に基づくスコア． - 補助的な指標：ブラウザ操作のステップ数．

これらを通じ，分散する Web サービスと IoT を横断統合し，個人に最適な自動化を実現するための実装指針と評価知見を提供する．

1.3 関連研究

近年，LLM の発展により，複数エージェントが協調してタスクを進める研究が活発化

している．本論文のテーマ「協調型 AI エージェントによるタスク効率化」に関連する主要プロジェクトを，目的・構成・本論文提案との共通点/相違点の観点で要約する．

(1) **AutoGen**： Microsoft Research が提案するマルチエージェント会話フレームワークである[3]．LLM・人間・ツール等をエージェントとして扱い，エージェント間の会話パターンを設計して多様なアプリケーションを構築できる．本研究と同様に「対話による協調」を核とするが，AutoGen 自体は汎用フレームワークであり，個人の生活空間（Web と IoT を横断する実行環境）に特化した設計や，個人最適化のための永続メモリ設計を主目的としていない．

(2) **Magentic-One**： 汎用の問題解決を狙うマルチエージェントシステムである[4]．中央のオーケストレータが計画・進捗管理・再計画を担い，Web 操作やファイル操作，コード生成／実行などの専門エージェントを統率する．モジュール構成で拡張しやすい一方，個人の生活空間向けに軽量化し，プライバシーや即応性を含めて設計することは別途必要となる．

(3) **Magentic-UI**： Magentic-One を基盤に，人間中心（Human-in-the-loop）の介入を組み込む UI／アーキテクチャである[5]．人間を一つのエージェントとして扱い，共創計画，タスク引き継ぎ，行動承認，結果検証，長期メモリ等の介入機構を備える．また，Model Context Protocol（MCP）により外部ツールを拡張・接続可能である[5]．本研究も人間介入やツール統合を重視するが，本研究は「特定個人の生活タスク」に焦点を当て，個人メモリと実行基盤（Web と IoT の横断）を一体として最適化する点が異なる．加えて，Magentic-UI の結果検証は，実行履歴の提示やリンク付与により「人間が正否を確認できる」ことを主眼とするのに対し，本研究ではオーケストレータが「計画→実行→確認」を一貫して担い，確認結果（Web の再照会・IoT 状態の再取得／照合など）を用いて完了判定・再計画・再実行・メモリ更新までを閉ループで行い，人間の負担を軽減する点を独自性とする．

(4) **LLMind**： LLM を用いて IoT を含む複合タスクをオーケストレーションする研究である[6]．自然言語指示を外部モジュール／デバイス操作へ変換して実行し，状態管理や経験の蓄積により，複数ステップのタスク遂行を支援する．本研究は IoT 制御に加えて Web 上の情報探索・Web UI 操作・スケジュール管理を統合し，さらに個人の属性・嗜好を保持するメモリに基づき曖昧性を解消して実行計画へ落とし込む点に主眼を置く．

(5) **HuggingGPT**： LLM がオーケストレータとなり，Hugging Face 等の専門モデルを選択・連携してマルチモーダル課題を解く枠組みである[7]．複数モデルの協調という点で示唆は大きいが，本研究が扱う「Web 操作・IoT 制御・スケジュール・知識検索」といった異種ツール群の統合実行とは目的が異なる．

(6) **実装基盤・ツール使用学習**： エージェント運用に必要な共通機能（メモリ，権限管理，ツール接続等）を基盤として提供する実装として AgentOS が提案されている[8]．また LangChain は，チェーン／エージェント／メモリ等の部品化により LLM アプリケーション開発を容易にする．さらに Toolformer は，LLM が外

部ツール（API）利用を自己学習して能力拡張する方向性を示した[1]．本研究はこれらの知見を踏まえつつ，個人の生活空間における「パーソナライズ（長期・短期メモリ）」と「Web＋IoT 横断実行」を同一オーケストレータの下で実装・評価することに焦点を置く．

(7) スマートホーム領域の推論・行動生成： スマートホームでは，LLM による目標指向推論・行動生成を扱う研究も報告されている[9]．一方で，生活空間では Web 上の情報探索／サービス操作と IoT 制御が同一タスク列として連結しやすく，さらに個人文脈（嗜好・予定・環境制約）により曖昧指示の解消が必要になるため，実運用を想定した統合設計と評価が課題となる．

まとめ： 既存研究は，マルチエージェントによる汎用的なタスク遂行（AutoGen，Magentic-One，Magentic-UI）や，IoT を含む複合タスクの自動実行（LLMind），モデル協調によるタスク分解と実行（HuggingGPT），および実装基盤やツール使用学習（AgentOS，LangChain，Toolformer）へと発展している．しかし，個人の生活空間を対象に，Web 空間の情報探索・Web UI 操作と物理空間（IoT）の制御を同一の実行基盤で横断統合し，個人メモリに基づく曖昧性解消・パーソナライズを組み込み，さらにエッジ推論とクラウド推論の役割分担まで含めて E2E に実装・評価した例は限定的である．本論文はこれらの知見を取り込みつつ，以下の点に独自性を持つ．第一に，Web 空間の情報探索と物理空間（IoT）の制御を，MCP を用いて等価な「ツール」として抽象化し，単一のオーケストレータ下で統合した点である．第二に，API 経由での大規模モデルとエッジデバイス上の軽量モデルを連携させ，プライバシーと即応性を両立する階層的な推論アーキテクチャを，個人の生活空間向けに最適化して実装した点である．第三に，個人の属性・嗜好・履歴を保持するメモリに基づき，曖昧な自然言語指示を補完して実行計画へ落とし込む設計を導入し，統合的に評価した点である．

1.4 本論文の構成

本論文は以下の構成で成り立っている．第2章ではシステム全体の説明，第3章では各エージェントの詳細と動作，第4章では評価，第5章では考察，そして第6章ではむすびとして本論文をまとめる．

2. システム全体の説明

本システムは，ユーザーの曖昧な指示を解釈し，Web 空間と物理空間（IoT）を横断してタスクを実行するための「協調型マルチエージェントプラットフォーム」である．システムの中核には，ユーザーの文脈や長期的な好みを記憶する「オーケストレータ」が存在し，その下に特定の機能に特化した4つの専門エージェント（Browserエージェント，IoTエージェント，Life-Style エージェント，Scheduler エージェント）が配下に配置されるハブ＆スポーク型のアーキテクチャを採用している．

2.1 システムの全体構成

システム全体は，独立した機能を持つマイクロサービス群として設計されており，Docker コンテナ上で動作する．各エージェントは HTTP ベースの REST API を公開しており，オーケストレータはこれを経由してタスクの依頼と結果の受け取りを行う． ユーザーがチャットインタフェースを通じて指示を出すと，オーケストレータがそれを解析し，必要なサブタスクに分解する．例えば「明日の天気に合わせて」という指示の場合，オーケストレータは「Browser エージェントに天気予報を検索させるタスク」と，その結果に基づいて「IoT エージェントに実行させるタスク」を生成し，順次実行に移す．これにより，単体の LLM では完結しない，外部情報の取得と物理的な操作を伴う複合タスクを実現している．

2.1.1 Model Context Protocol (MCP) の採用

本システムでは，エージェントの機能を標準化し，将来的な相互運用性を高めるために，Anthropic 社が提唱する「Model Context Protocol (MCP)」を導入している[10]．MCP は，AI モデルと外部データとツールをつなぐための，オープンな標準プロトコルである． 各エージェントは，従来の REST API に加え，MCP サーバーとしての機能も実装している．これにより，MCP に対応した外部クライアントから，本システムの持つ「ブラウザ操作」「スケジュール管理」「知識検索」「IoT 制御」といった機能を，追加のつなぎ込み開発を行うことなく，統一的なインタフェースで直接呼び出すことが可能となる．これは，特定のオーケストレータに依存しない，より汎用的なエージェント利用への道を開くものであり，エージェントのエコシステム構築において将来的な拡張性と相互運用性の観点で有用である．

2.2 オーケストレータとメモリ機能

オーケストレータは，LangGraph を用いたステートマシンとして実装されており，「計画（Plan）→実行（Execute）→レビュー（Review)」という自律的なループ処理を行う[11].

(1) 計画 (Plan): ユーザーの入力と過去の履歴，そして「メモリ」を参照し，どのエージェントに何をさせるべきかという実行計画（JSON 形式）を生成する．

(2) 実行 (Execute): 計画に基づき，各エージェントの API を呼び出す．

(3) レビュー (Review): エージェントからの実行結果を評価し，意図を満たしているか確認する．不足があれば再実行を指示する．

また，本システムの最大の特徴は，以下の2種類のメモリ機能によるパーソナライズである．

(1) 長期メモリ (Long-term Memory): JSON ファイル（「長期記憶データベース」）として保持され，ユーザーの住所，家族構成，趣味，持病といった永続的な属性情報を格納する．

(2) 短期メモリ (Short-term Memory): JSON ファイル（「短期記憶データベース」）として保持され，直近の興味関心や一時的なコンテキストを格納する．

これにより，ユーザーが毎回詳細な前提条件を説明しなくても，「いつもの」という指示だけで意図を汲み取ることが可能となる．

2.3 連携する専門エージェント

オーケストレータの指揮下には，以下の4つの専門エージェントが配置されている．これらは REST API を提供するほか，MCP サーバーとしても動作し，標準化されたツール定義を公開している．

(1) Life-Style エージェント: RAG (Retrieval-Augmented Generation) 技術を用い，家庭内でのコミュニケーション，健康促進，あるいは一般的な生活の知恵を検索・回答する知識ベース担当のエージェント．FAISS を用いたベクトル検索[12]により，膨大なドキュメントから適切な回答を生成する．

(2) Browser エージェント: browser-use ライブラリ[13]を基盤とし，実際の Chrome ブラウザを自律操作するエージェント．Web 検索による情報の取得や，フォーム入力といった操作を担う．

(3) IoT エージェント: 家電やセンサー等の IoT デバイスを統合管理するエージェント．LLM を用いて自然言語の指示をデバイス固有の制御コマンドに変換し，Job Queue を通じて非同期に実行する．

(4) Scheduler エージェント: ユーザーの日々のルーティンやタスク，メモを管理するエージェント．LLM の Function Calling 機能を活用し，自然言語による会話を通じてスケジュールの確認やタスクの追加，完了報告といった操作をデータベースに対して確実に行う．

さらに，各専門エージェントは，タスクの難易度や評価の目的に応じて，LLM モデルを動的に切り替える機能を備えている．例えば，単純なスケジュール登録には軽量なモデル（Gemini 2.5 Flash Lite など）を使用し，複雑な推論を要するタスクには高性能なモデル（GPT-5.1など）を割り当てることで，コストと性能の最適化を図るとともに，モデル性能がシステム全体の挙動に与える影響についても評価する．

3. 各エージェントの詳細と動作

3.1　オーケストレータエージェント

オーケストレータエージェントは，本システムの中枢として機能し，ユーザーからの曖昧な指示を具体的なタスク計画に分解し，専門エージェントの実行を監督する役割を担う．その実装には，ステートマシンとグラフベースの思考プロセスを管理するためのフレームワークである「LangGraph」が採用されている[14]．これにより，エージェントは自律的に「計画」「実行」「レビュー」を繰り返すことで，複雑な問題解決と自己修正を可能にしている．開いたときの画面は図3.1のようになっている．

図3.1　オーケストレータの画面例

3.1.1　アーキテクチャと自律制御ループ

オーケストレータの動作は，「計画（Plan）→実行（Execute）→レビュー（Review）」という一連の状態遷移グラフとして定義されている．図3.2がその流れである．

(1) 計画 (Plan): ユーザーの入力，会話履歴，そして長期・短期メモリを基に，LLM（Gemini, Claude, GPT, Groq）がタスクの実行計画を立案する．この計画は，どのエージェント（「知識ベース機能」，「ブラウザ操作機能」，「IoT 制御機能」，「スケジュール管理機能」）に何（「実行コマンド」）をさせるかという具体的な指示を JSON 形式で記述したものである．

(2) 実行 (Execute): 計画されたタスクリストに基づき，各専門エージェントの HTTP API を順次呼び出す．

(3) レビュー (Review): 各タスクの実行結果を LLM が評価する．ユーザーの当初の目的が達成されているかを確認し，結果が不十分な場合はタスクの再試行を指示することができる．

このループ構造により，一度の失敗で処理を中断するのではなく，目標達成に向けて試行錯誤を繰り返す，より堅牢なタスク遂行が実現される．図3.2と図3.3のような構成とフローになっている．

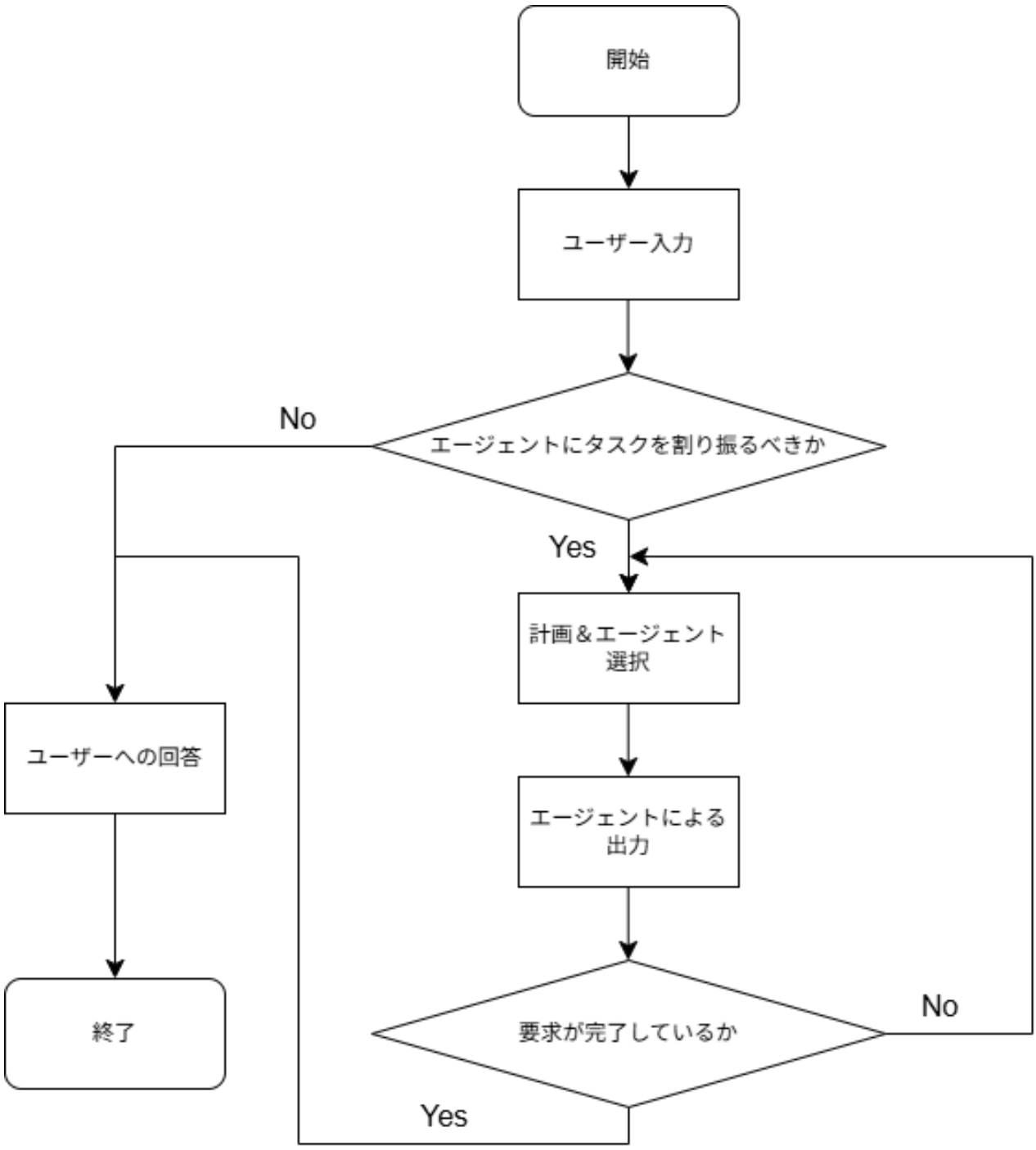

図3.2　オーケストレータのフロー

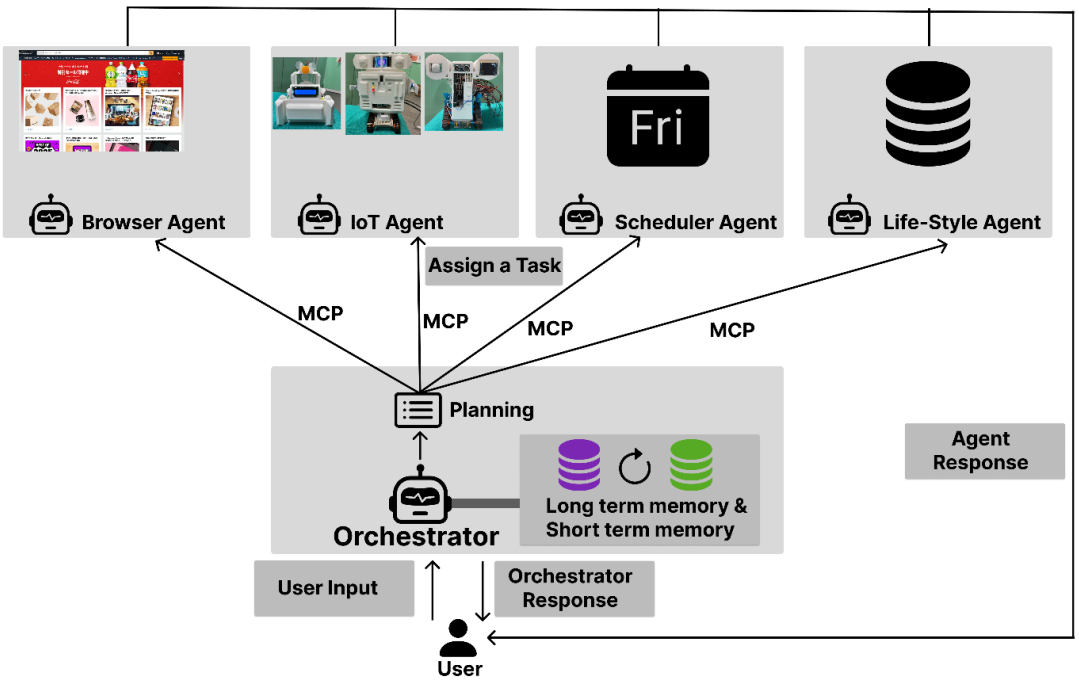

図3.3　オーケストレータを中心としたシステム全体

3.1.2　メモリ更新のアルゴリズム

本システムは MemoryManager がメモリを一元管理し，LLM の推論結果を「差分（Diff）」として既存メモリへ安全に反映する（長期メモリ・ツールを組み合わせる一般設計も踏まえた）[15]．更新は単純な上書きではなく，整合性と永続性を優先して次の手順で行う．図3.4のような忘却曲線を描くように設計されている．

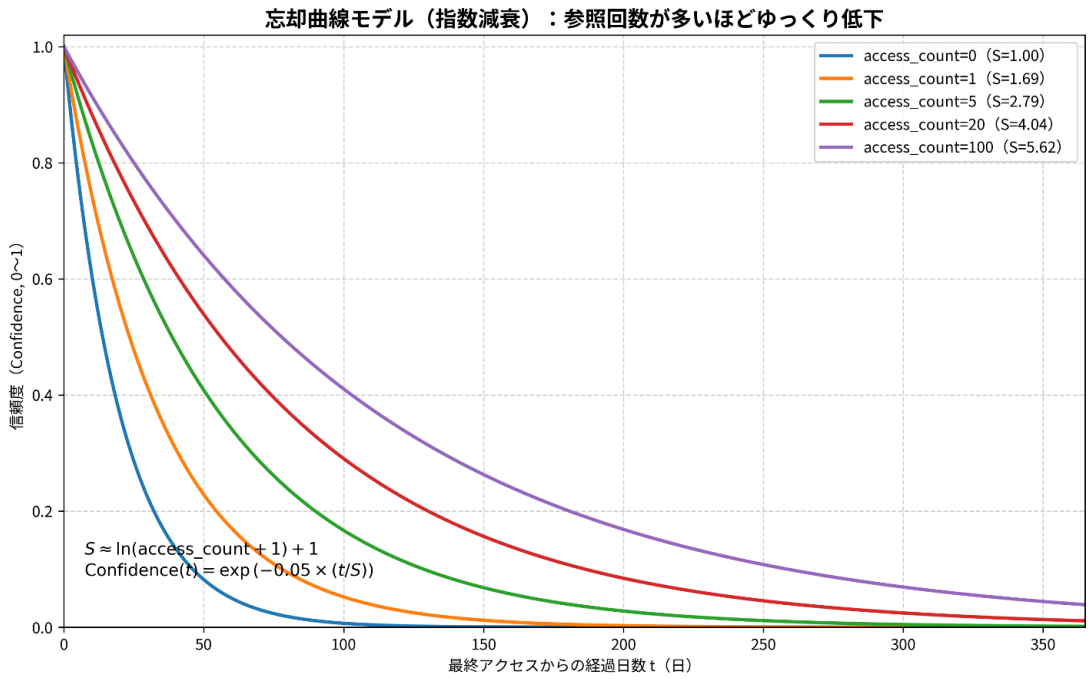

図3.4　メモリの更新アルゴリズムの例

(1) 更新差分

対話が一定量進むと会話ログとメモリスナップショットを LLM へ入力し，更新内容を更新差分（JSON）として生成する．

(a) 短期メモリ：作業バッファとして，上書き・破棄を許容．

(b) 長期メモリ：追記を原則とし，リスト更新は全量置換ではなく { "add": [...], "remove": [...] } の差分操作を強制．

(2) 差分適用

(a) 更新データ：辞書は再帰マージ，リストは add/remove で重複排除しつつ追加・削除（既存情報の消失を防止）．

(b) 操作列：

(i) スロット更新：ID 正規化＋ファジーマッチで表記ゆれを統合し，更新履歴と信頼度・優先度等を更新

(ii) 利用記録：参照回数とスコアを更新し，頻繁に使う情報の重要度を上げる

(iii) エピソード／プロジェクト管理：エピソード記憶・プロジェクト状態を管理

(3) 昇格

短期メモリの TTL（Time To Live）期限時に評価し，参照回数や重要度が閾値を超えるスロット／エピソードを長期メモリへコピーして定着させる．

(4) 減衰（忘却曲線）

参照されない情報の信頼度を指数的に低下させる．この「忘却曲線」に基づく記憶の強化／減衰の考え方は，長期対話における記憶更新機構として提案された MemoryBank の設計に着想を得た[16]．

(a) 安定性 S （参照回数が多いほど忘れにくい）： $S \approx \ln(\text{access\_count} + 1) + 1$

(b) 信頼度 Confidence(t) （最終アクセスからの日数 t に応じて減衰）： $\text{Confidence}(t) = \exp\left(-0.\ 05 \times \frac{t}{S}\right)$

係数0.05は経験的に設定した．本設定では，t/S ≈ 14 日で Confidence(t) が約0.5まで低下する減衰率となり，短期の文脈変化には追従しつつ長期属性を急激に消し過ぎないことを狙っている．係数の最適化は今後の課題とする．

長期未参照の情報はスコアが下がり「低優先度」へ分類するが，完全削除はせず痕跡を保持する．

この設計により，「作業中の文脈（短期）」と「人物像として残す情報（長期）」を分離しつつ，対話の継続で自律的に更新・定着させる．

3.1.3 計画フェーズ：プランナーの役割と安全性確認

計画フェーズでは，「計画生成用プロンプト」を用いて，LLM に対しユーザーの要求を分析し，適切な行動方針を決定する． まず，エージェントによる操作が不要な場合（例：単なる知識の確認や挨拶）は，「直接回答モード」を選択し，タスク生成を行わず

に即座に応答することで，応答速度の向上とリソース節約を図る． 専門エージェントの利用が必要と判断された場合は，ユーザーの要求を実行可能な具体的な単位へと細分化し，あらかじめ設定された上限（最大タスク数）の範囲内で複数のタスクへと分解する．ここで本システムは，一度に全ての計画を固定するのではなく，**増分再計画** を採用している．これは，一つのタスクが完了するたびに，その実行結果（成功・失敗・得られた情報）をコンテキストに含めて再度計画プロセスを実行する仕組みである．これにより，前のタスクの結果（例：ブラウザで検索した天気情報）に応じて，次のタスク（例：IoT で照明の操作）の内容を動的に調整することが可能となる．

また，タスク生成後には**実行可能性判定** と呼ばれる安全機構が設けられている．これは，生成されたタスクコマンドが「具体的に実行可能か」を別の LLM 呼び出しによって検証するプロセスである． また,決済や配送といった取り返しのつかない操作において必須情報が欠落している場合に限り，ユーザーへ「追加質問」を行う．これにより不確実性を減少させている．

3.1.4 実行フェーズ：専門エージェントの呼び出し

実行フェーズでは，計画されたタスクの「エージェント識別子」種別に応じて，以下の通り各専門エージェントの API が呼び出される．

(1) **「知識ベース機能」（Life-Style エージェント):** 「Life-Style エージェント呼び出し関数」を通じて，エージェントの「回答生成 API」エンドポイントに HTTP POST リクエストを送信する．オーケストレータが Life-Style エージェントを操作している様子を図3.5に示す．

(2) **「ブラウザ操作機能」（Browser エージェント):** 「Browser エージェント呼び出し関数」を通じて，デフォルトで「Browser エージェントのサービスアドレス」で待機するエージェントの「チャット API」エンドポイントを呼び出す．進捗をリアルタイムで受け取るためのストリーミング通信もサポートされている．オーケストレータが Browser エージェントを操作している様子を図3.6に示す．

(3) **「IoT 制御機能」（IoT エージェント):** 「IoT コマンド送信関数」を通じて，エージェントに HTTP リクエストを送信し，デバイスの操作を指示する．オーケストレータが IoT エージェントを操作している様子を図3.7に示す．

(4) **「スケジュール管理機能」（Scheduler エージェント):** 「Scheduler エージェント呼び出し関数」を通じて，エージェントにチャットメッセージを送信し，予定の確認やタスクの追加・更新等の操作を行う．オーケストレータが Scheduler エージェントを操作している様子を図3.8に示す．

3.1.5 レビューフェーズ：自己修正メカニズム

各タスクの完了後，レビューフェーズが開始される．ここでは「レビュー用プロンプト」というプロンプトを用いて，LLM にエージェントの実行結果を評価させる．レビューの結果，ステータスが「再試行ステータス」と判断された場合，オーケストレータは同じタスクを最大再試行回数まで再実行する．この自己修正メカニズムにより，一時的なエラー

や不十分な結果から回復し，タスクの成功率を高めることが可能となる．

図3.5　オーケストレータが Life-Style エージェントを操作している画面例

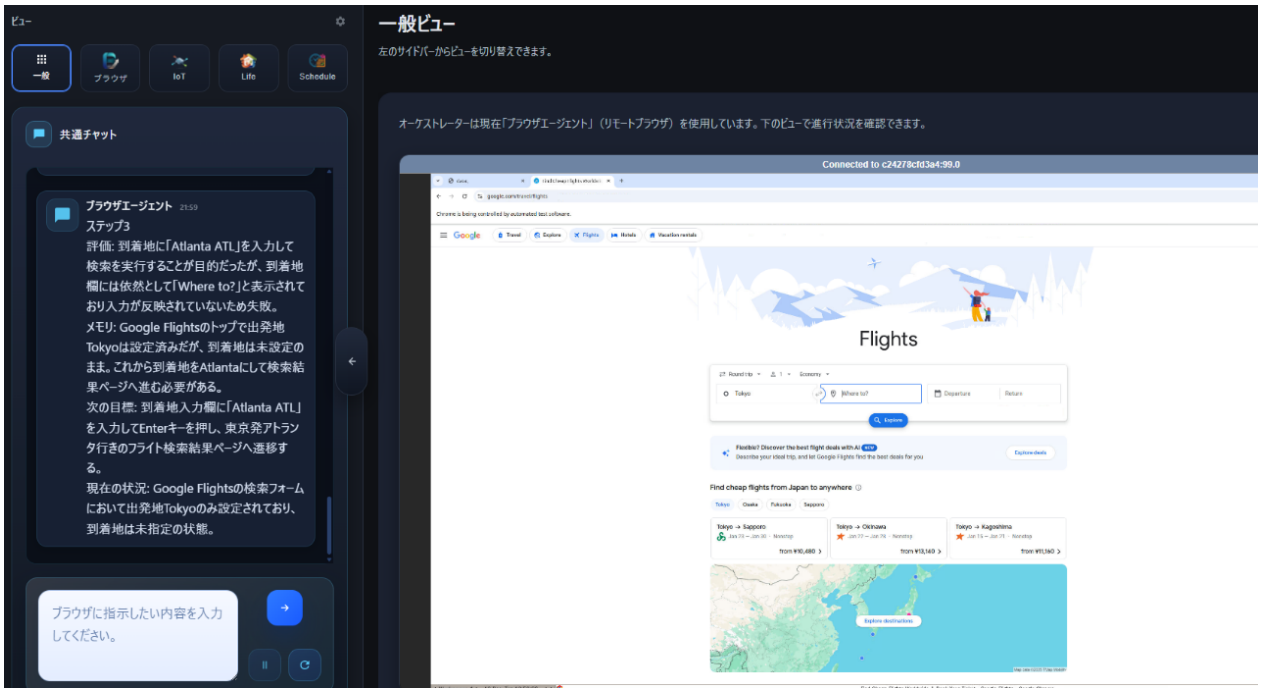

図3.6　オーケストレータが Browser エージェントを操作している画面例

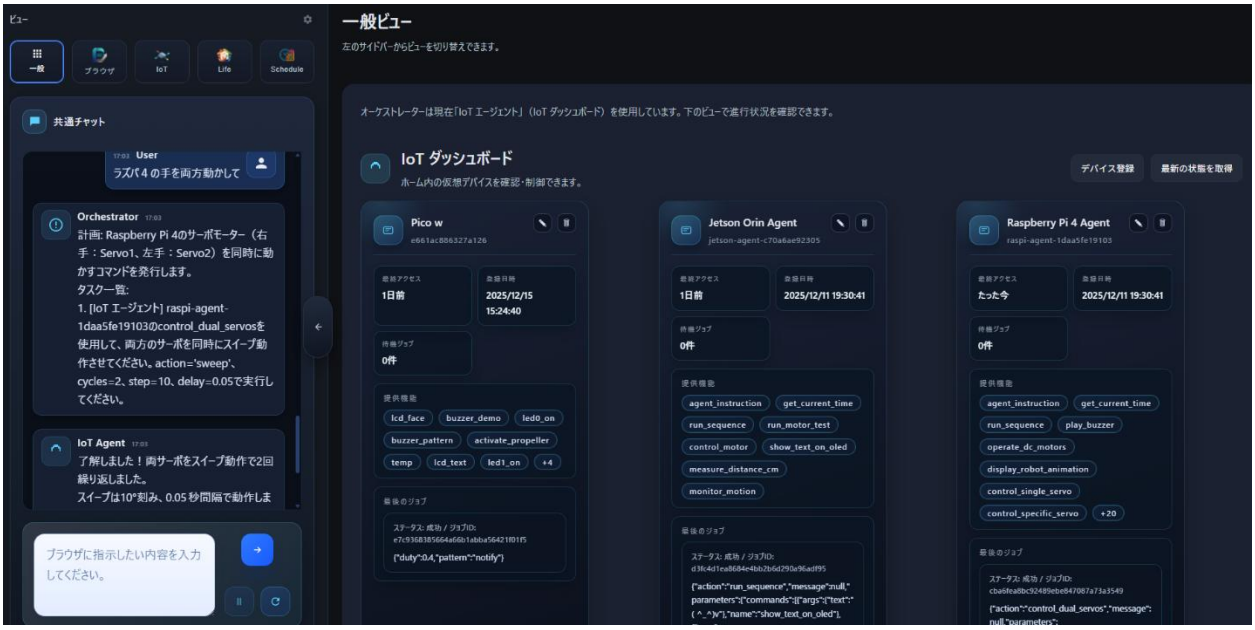

図3.7　オーケストレータが IoT エージェントを操作している画面例

図3.8　オーケストレータが Scheduler エージェントを操作している画面例

3.2　Browser エージェント

Browser エージェントは，Web ブラウザ（Chrome）を人間のように操作し，情報の検索，サイトの閲覧，Web サービスの操作などを自律的に実行する専門エージェントである．本エージェントは，Python の Web フレームワークである Flask を用いて API サーバーとして実装されており，システムの他のコンポーネント（オーケストレータなど）からのリクエストに応じて動作する．Browser エージェントの動作例を図3.9に示す．

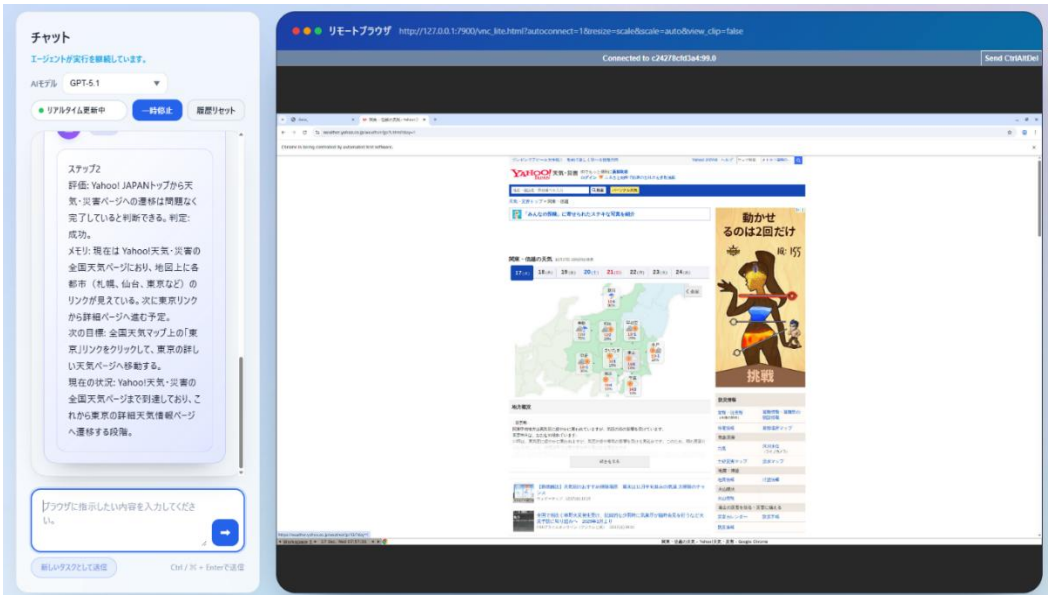

図3.9　Browser エージェントの画面例

3.2.1　概要と技術スタック

本エージェントの中核技術には，LLM（Gemini, Claude, GPT, Groq） browser-use が採用されている[13]．これにより，DOM（Document Object Model）の解析や要素の特定といった従来のスクレイピング技術では困難だった，動的な Web サイトや複雑な UI の操作が可能となっている． ブラウザとの通信には Chrome DevTools Protocol (CDP) を利用しており，Docker コンテナなどの分離された環境で動作するヘッドレスブラウザに対しても，リモートから安定した制御を行うことができる．

Browser エージェントの構成を図3.10に示す．

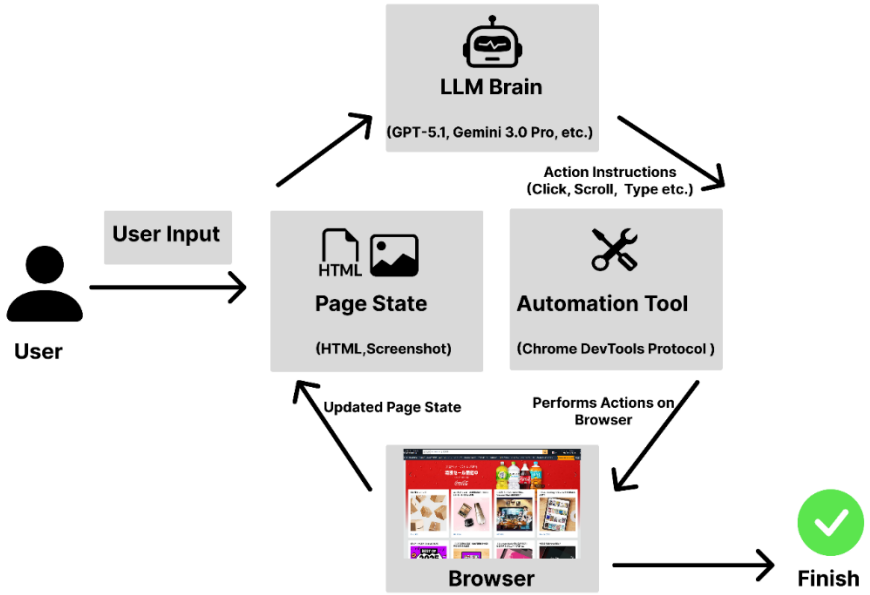

図3.10　Browser エージェントの構成

3.2.2 アーキテクチャと主要機能

内部アーキテクチャは，ブラウザ操作のライフサイクルを管理するコントローラを中心に設計されている．

(1) **エージェント制御とセッション管理:** コントローラは，ブラウザの起動，タスクの実行，終了処理を一元管理する．最新の実装では，セッションの永続化（Keep-Alive）と自動復旧メカニズムが強化されており，タスク完了後もブラウザの状態を維持して次の指示に備えることが可能である．また，内部的に EventBus と Watchdog（監視プロセス）を導入し，ブラウザのクラッシュ，予期せぬポップアップ，ダウンロード完了等を常時監視することで，長時間稼働における高い安定性を確保している．

(2) **リアルタイム進捗配信:** 実行状況は Server-Sent Events (SSE) を用いてリアルタイムに配信される．これにより，ユーザーは，エージェントが現在「どのページを見ているか」「何を入力しているか」といった詳細なステップを逐次確認することができる．

(3) **モデルの柔軟性:** 使用する LLM モデルを動的に切り替える機能を備えており，タスクの難易度やコストに応じて最適なモデル（Gemini 2.5 Flash Lite, GPT-5.1など）を選択して実行できる．

(4) **テキスト応答モード:** ブラウザ操作が不要と判断された場合（例：一般的な知識で回答可能な質問），ブラウザを起動せずに LLM がテキストのみで即座に応答する機能を備え，リソース消費と応答時間を最小化している．

3.2.3 外部システムとの連携（API）

本エージェントは RESTful API を提供し，外部システムとの柔軟な連携を実現している．主なエンドポイントは以下の通りである．

(1) **メインチャット API:** ユーザーやオーケストレータからの自然言語による指示を受け付け，タスクを開始する主要なインタフェース．既にタスクが進行中の場合は，追加の指示（フォローアップ）としてキューに追加され，動的に処理内容が修正される．

(2) **エージェントリレーAPI（「エージェント連携 API」）:** 他のエージェントが一時的にブラウザ操作を利用するための専用エンドポイント．ここでのやり取りはメインの会話履歴には保存されず，独立したサブタスクとして処理されるため，履歴の汚染を防ぎつつ機能を共有できる．

3.3 IoT エージェント

IoT エージェントは，物理空間にある IoT デバイス（RaspberryPi Pico W, RaspberryPi4, Jetson Orin Nano 等）を，AI エージェントシステムと接続するための統合ゲートウェイである．本エージェントは Flask ベースの Web サーバーとして実装されており，自然言語による指示をデバイス固有の制御コマンドへ変換し，非同期で実行させる機能を持つ．IoT エージェントの画面を図3.11に示す．

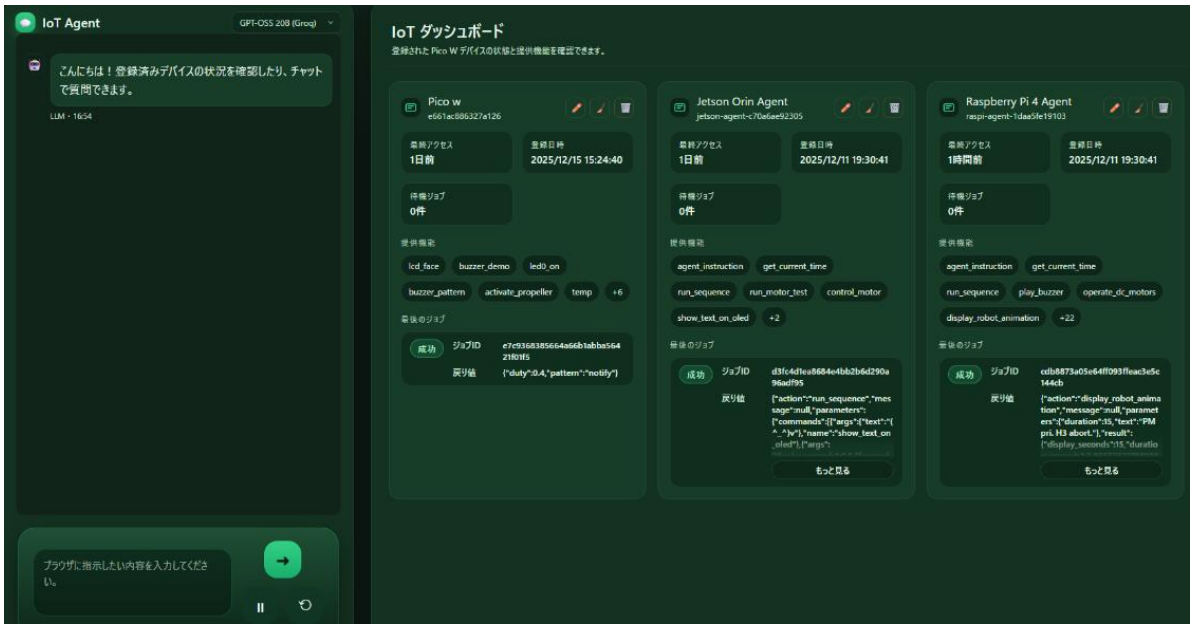

図3.11　IoT エージェントの画面例

3.3.1　概要と役割

IoT エージェントの主な役割は，プロトコルや操作方法が異なる多様なデバイスを抽象化し，統一された API を通じて操作可能にすることである．オーケストレータからの HTTP リクエスト，または Web UI 上のチャットインタフェースからの直接入力を受け取り，背後に登録された適切なデバイスへと命令を振り分ける． また，本エージェントは MCP サーバーとして動作し，外部のエージェントや LLM に対して「デバイス制御ツール」や「デバイス一覧取得ツール」を標準的な形式で公開している．これにより，LLM は自然言語の文脈の中で自律的にこれらのツールを選択・実行することが可能となっている．IoT エージェントの構成を図3.12に示す．

3.3.2　システムアーキテクチャ

IoT エージェントは，以下のコンポーネントで構成される．

(1) **Flask サーバー（「メインサーバープログラム」）:** システムの中核であり，REST API エンドポイントの提供，認証管理，LLM との通信，ジョブ管理を行う．

(2) **MCP サーバーモジュール:** MCP SDK を用いて実装されており，LLM からのツール呼び出しを処理し，具体的なデバイスコマンドへと変換するインタフェースを提供する．

(3) **インメモリデータストア:** デバイスの接続状態，登録された機能，実行待ちのジョブキュー，実行結果ログをメモリ上で管理する．永続化データベースを持たない軽量設計となっており，高速なレスポンスを実現している．

(4) **エッジデバイスクライアント:** Raspberry Pi や Jetson などのエッジデバイス上で動作する Python スクリプト．これらは定期的にサーバーへポーリングを行い，自デバイス宛てのジョブ（コマンド）を取得・実行し，その結果（成功/失敗，センサー値，画像データ等）をサーバーへ返送する．

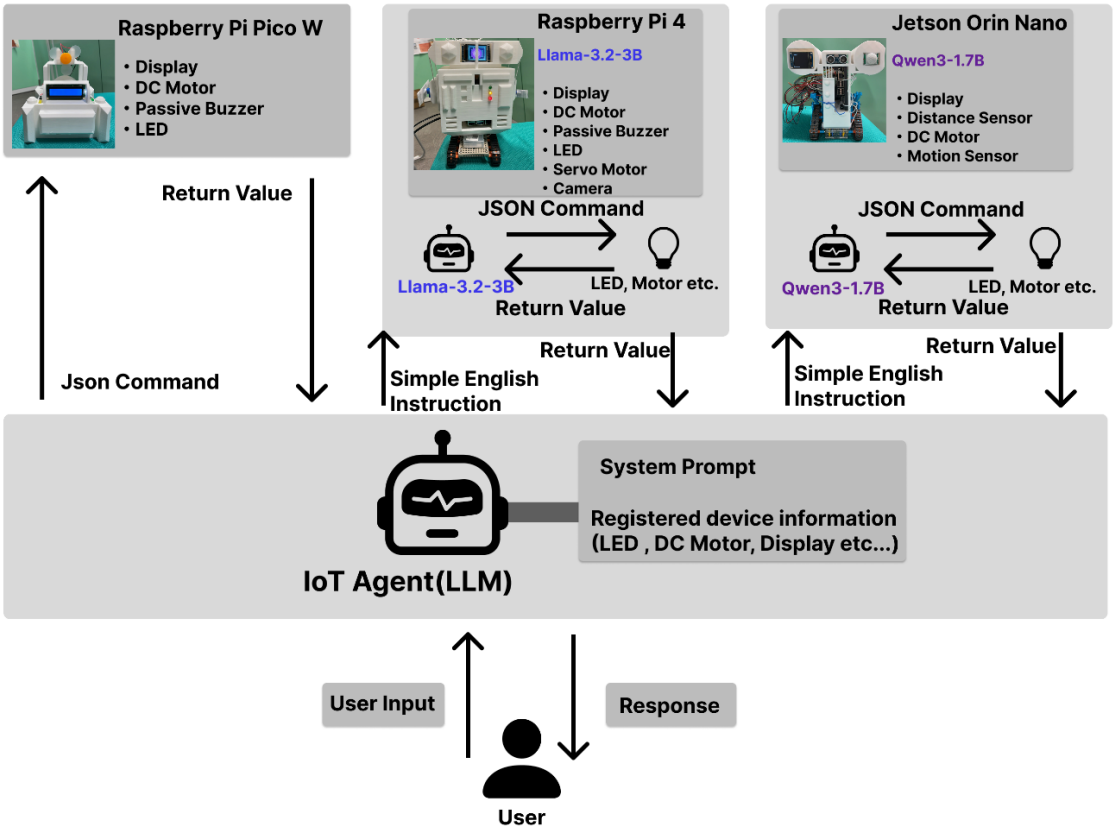

図3.12　IoT エージェントの構成

3.3.3　LLM によるコマンド変換と実行フロー

ユーザーまたはオーケストレータからの指示は，MCP を活用した以下のフローで処理される．

(1) メッセージ解析: 受信した自然言語メッセージは，現在の会話履歴とともに LLM へ送信される．この際，システムは利用可能な MCP ツール定義を LLM に提示する．

(2) ツール呼び出し (Function Calling): LLM は，ユーザーの意図（例：「LED を光らせて」）を解釈し，それを実現するために適切なツールと引数（例：control_device(device_id="raspi-agent-123",command="led1_on",args={"duration": 5})）を決定し，関数呼び出しを生成する．

(3) 実行と検証: 生成されたツール呼び出しは MCP サーバーモジュールによって受け取られ，引数の妥当性（必須パラメータの有無やデバイスの機能サポート状況）が検証される．

(4) キューイング: 検証を通過したリクエストはジョブとしてキューに追加され，対象のデバイスによるポーリングを待つ．

この仕組みにより，従来の正規表現や単純な JSON 生成に比べ，より複雑で柔軟なデバイス操作が高い精度で実現されている．

さらに，LLM に対するシステムプロンプトを拡張し，ユーザーの負担を最小限に抑えるための以下の自律判断ロジックを導入している．

(1) デフォルト値の自動補完: ユーザーが詳細なパラメータ（例：モーターの回転時間

や角度，ブザーの音色）を指定しなかった場合，エージェントは「どのくらいですか？」と聞き返すのではなく，システム側で定義された標準値（例：移動なら5.0秒，サーボなら90度）を自動的に適用して即座に実行に移る．これにより，冗長なやり取りを削減している．

(2) 文脈による対象推定: 登録されているデバイスが1台のみの場合や，直前の文脈から操作対象が明らかな場合は，デバイス指定が省略されていても自動的に対象を特定して，ユーザーへの確認なしにコマンドを発行する（「Read the room」アプローチ）．

(3) デバイス差分の吸収: 同様の機能（例：ディスプレイへの文字表示）を持つデバイス間でのコマンド体系の違い（例：Jetson のディスプレイにテキストを表示する機能と Raspberry Pi のディスプレイにテキストを表示する機能）をプロンプトレベルでの指示により吸収し，LLM がデバイス種別に応じて適切な内部コマンドを選択するように制御している．

3.3.4 デバイス管理とジョブキュー

本システムでは，デバイスの動的な登録と管理が可能である．デバイスは起動時に自身の ID と機能（例：「(電源操作，色変更，撮影などの）機能リスト」）をサーバーに通知し，登録を行う． コマンド実行は非同期のジョブキュー方式を採用している．これにより，ネットワーク不安定な環境にある IoT デバイスや，処理に時間を要するデバイス（例：画像のアップロードを伴うカメラ）に対しても，サーバー側がブロックされることなく安定して命令を発行・管理できる．各ジョブには一意の ID が付与され，オーケストレータは後から実行ステータスや結果を確認（ポーリングまたは待機）することができる．

3.3.5 視覚拡張機能

IoT エージェントは，カメラ付きデバイス（Raspberry Pi + Camera Module 等）と連携した視覚機能も備えている． チャット内で「周りの状況を教えて」「写真を見せて」といったキーワードが検出されると，システムは自動的に視覚対応モデル（GPT-5.1や Gemini 3 Pro 等）を選択し，カメラデバイスに対して撮影コマンド（「カメラ撮影コマンド」）を発行する． 撮影された画像はエッジデバイスからサーバーへアップロードされ，LLM へと渡される．これにより，単なるセンサー数値だけでなく，「部屋が散らかっているか」「電気がついているか」といった視覚的な文脈に基づいた高度な状況判断と回答が可能となっている． IoT エージェントがデバイスから渡された画像から周囲の状況を説明する様子とその仕組みを図3.13，図3.14に示す．

3.3.6 実装されたエッジデバイス

本研究では，システムの有効性を検証するために，以下の3種類のエッジデバイスを実際に構築・接続した．最新の実装では，全デバイスにおいてアクションの「並列実行」がサポートされており，移動しながらの表示や，ブザーを鳴らしながらの動作などが可能となっている．

図3.13　デバイス側から送信された画像を IoT エージェントが説明する様子

(1) Jetson Orin Nano 8GB

(a) **役割**

高度なエッジ AI 処理能力を活かし，ローカル LLM（Qwen3-1.7B-Q4_K_S.gguf）を用いた自律的なコマンド解釈とデバイス制御を行う[17].

(b) 機能（入出力・デバイス）

(i) 移動制御: モーター制御（L293D）による全方向移動

(ii) 表示: ディスプレイ（SH1107）へのテキストメッセージ表示

(iii)　計測: 超音波距離センサー（HC-SR04）による計測

(iv)検知: PIR センサー（SR501）による動体検知

(c) **実行形態（制御・オーケストレーション）**

これらの機能を組み合わせた 並列・順次実行が可能.

(d) **特徴（アーキテクチャ）**

サーバーからの自然言語指示を デバイス内で直接解釈・実行 する 分散インテリジェンスモデル を採用している.

(e) **図表（参照先）**

実際のデバイスの画像とその構成を図3.15，図3.16に示す．また，Jetson Orin Nano の GPU と CPU の両方で AI モデルを動作させたときのトークン生成速度の比較を表3.1に示す.

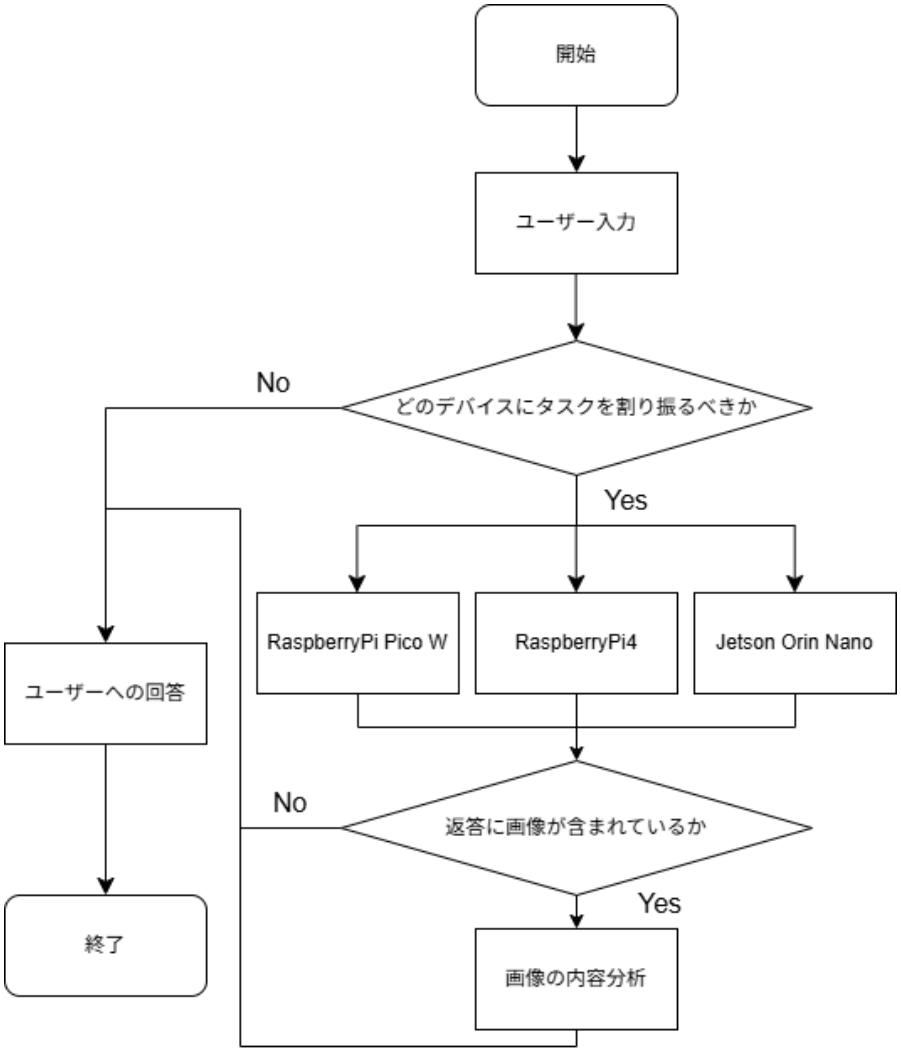

図3.14　IoT エージェントが画像分析するフロー

表3.1　Jetson Orin Nano における Qwen3 1.7B モデルの CPU 推論と GPU 推論のトークン生成速度比較

試行回数	CPU Speed (tokens/sec)	GPU Speed(tokens/sec)	Speedup（倍）
1	8.53	21.95	2.57
2	9.75	24.85	2.55
3	9.78	25.43	2.60
4	9.75	25.13	2.58
5	9.81	24.69	2.52
6	9.87	25.07	2.54
7	9.70	24.97	2.54
8	9.75	25.17	2.58
9	9.73	25.37	2.61
10	9.79	24.92	2.55
平均	**9.64**	**24.76**	**2.57**

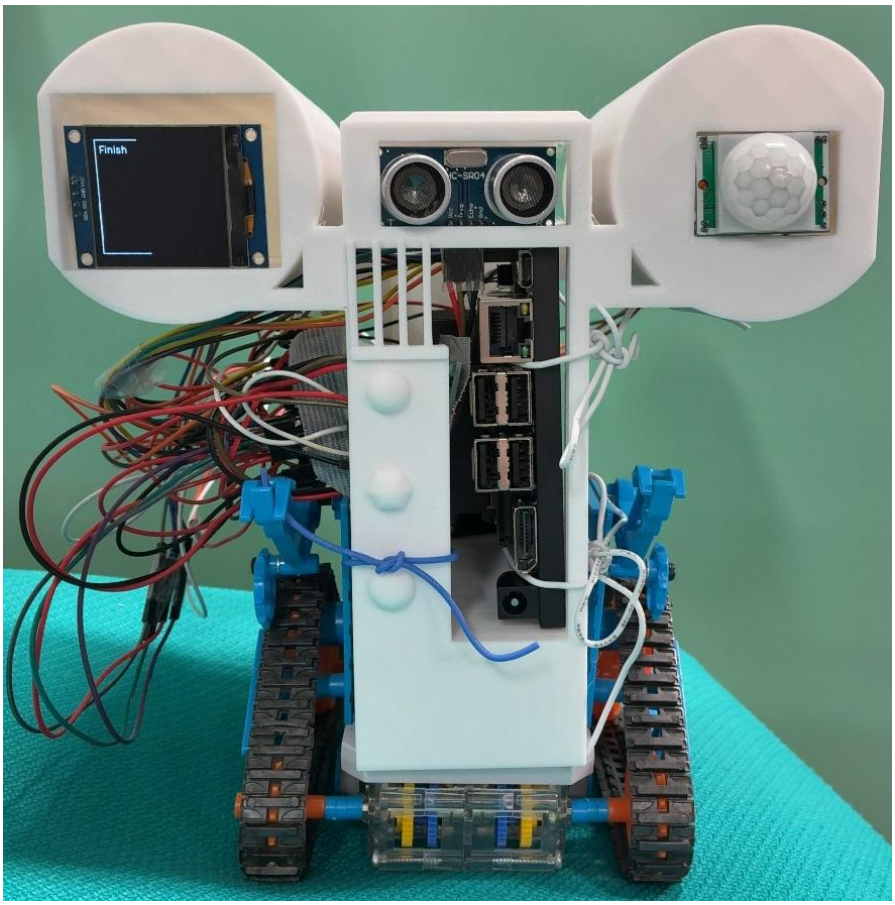

図3.15　Jetson Orin Nano を搭載した IoT デバイス

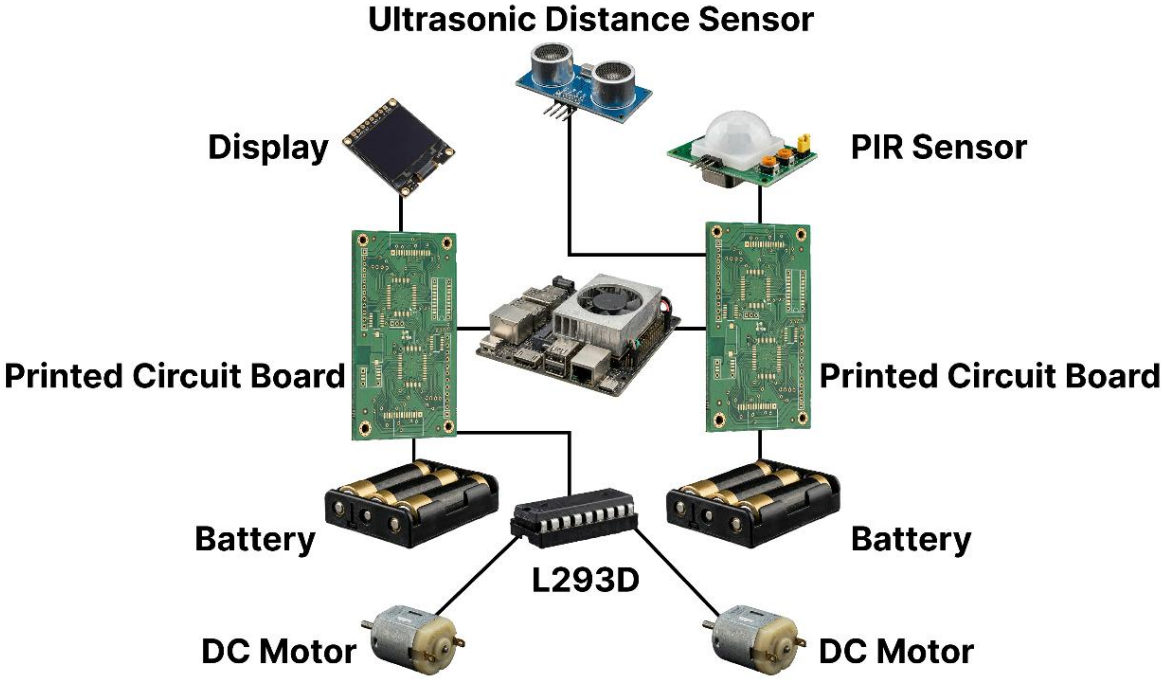

図3.16　Jetson Orin Nano を搭載した IoT デバイスの構成
（Google Imagen 4を一部利用）

表3.1より，GPU 推論は CPU 推論に対して平均2.57倍（CPU: 9.64 tokens/sec，GPU: 24.76 tokens/sec）高速であった．この結果から，本研究ではエッジ側の LLM 推論は GPU を優先して用い，ユーザー対話からデバイス制御までの応答遅延を低減する方針とした．さらに，推論負荷を GPU へ寄せることで CPU 資源をセンサー処理・通信・制御に割り当てやすくなり，システム全体の即応性向上にも寄与する．

(2) Raspberry Pi 4

(a) **役割**

汎用的な処理能力と豊富なインタフェースを持つ標準的なゲートウェイとして機能する．ローカル LLM（Llama-3.2-3B-Instruct-Q4_K_M.gguf）[18]を搭載可能．

(b) 機能（入出力・デバイス）

(i) 視覚: カメラモジュール（Picamera2）を用いた画像撮影と転送

(ii) 制御（運動）: サーボモーター（手）および DC モーター（足）の独立制御

(iii) 表現:

- LED（赤・黄・緑）の点灯パターン制御
- パッシブブザーによるメロディ再生
- ディスプレイ（ST7735）へのロボットの目のアニメーション表示

(c) **制御単位（操作インタフェースの粒度）**

「右手（サーボモーター)」「左手（サーボモーター)」といった 部位ごとの操作が可能．

(d) **特徴（システム上の位置づけ）**

視覚拡張機能の中核を担い，撮影した画像をサーバーへ送信することで，マルチモーダルな状況判断を支援する．

(e) **図表（参照先）**

実際のデバイスの画像とその構成を図3.17，図3.18に示す．

将来的な拡張性

一見すると，IoT エージェントがタスクを SLM（Small Language Model）へ委譲し，さらにエッジ側で推論する構成は，処理経路が増えるため非効率に見える．しかし，本構成（IoT エージェントがタスクを SLM へ委譲し，エッジ側で推論する）は，将来的な拡張を前提にすると合理的である．すなわち，今後，家電クラスの計算資源でも現実的に動作する AI モデルが普及した場合，家庭内データ（会話のテキストデータ，音声，画像，映像）を外部サーバーへアップロードせずに各家庭内に保持したまま，家庭ごとの文脈に合わせた応答や，過去の出来事への質問応答といった高度な支援へ発展させやすい．加えて，通信品質に依存せず最低限の推論・制御を継続できるため，遅延や運用コストの観点でも有利になり得る．

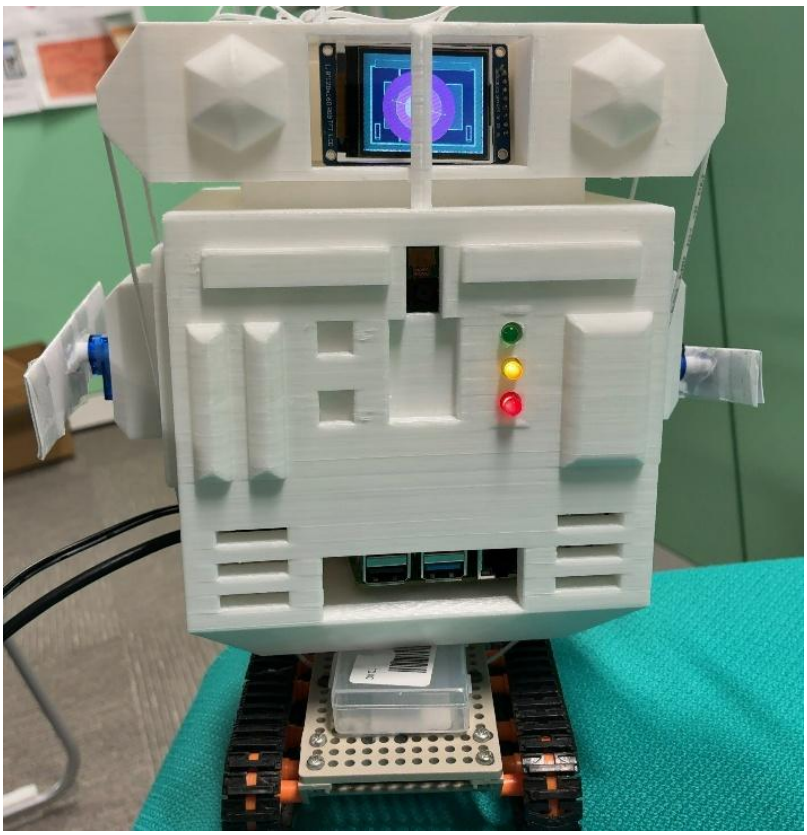

図3.17　RaspberryPi4を搭載した IoT デバイス

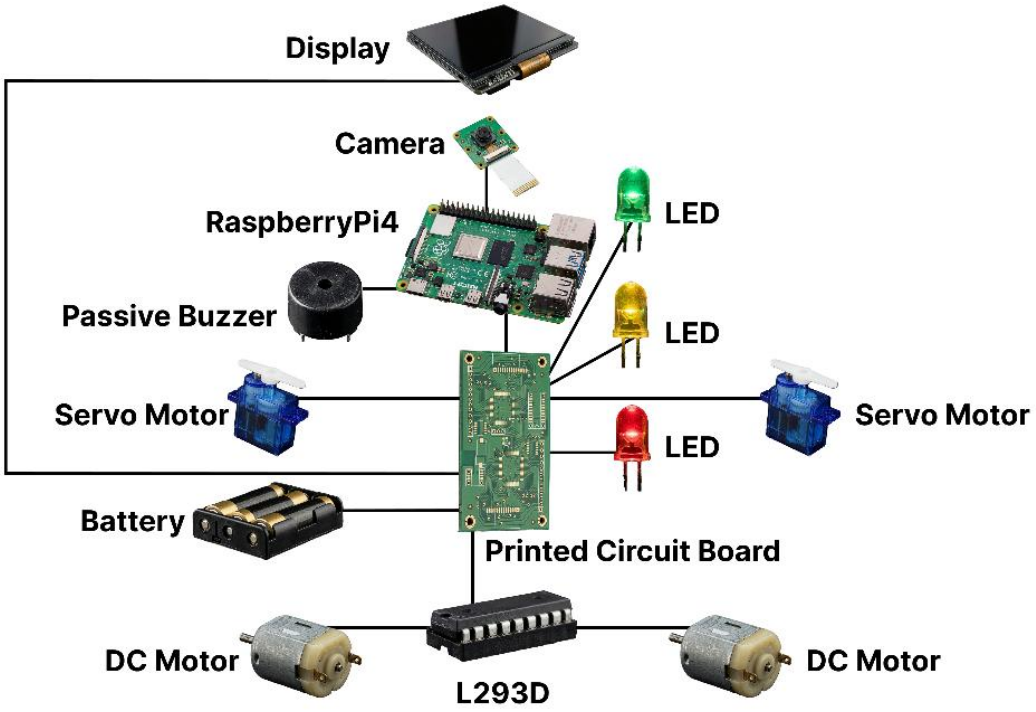

図3.18　RaspberryPi4を搭載した IoT デバイスの構成
（Google Imagen 4を一部利用）

(3) Raspberry Pi Pico W

(a) **役割**

低消費電力・低コストなマイコンベースのエンドポイントデバイス．MicroPython で実装されている．

(b) **機能（入出力・デバイス）**

Web サーバーと通信し，軽量な JSON コマンドに基づいて以下を実行する：

(i) LED 点滅

(ii) ブザー演奏

(iii) DC モーター駆動

(iv)LCD（HD44780）への表情のアニメーション,テキスト表示

(c) **実行形態（制御・オーケストレーション）**

並列実行もサポート．

(d) **特徴（責務分担・制約）**

リソースが限られるため，複雑な推論は行わず，サーバー側で生成された具体的な関数呼び出しを実行することに特化している．このデバイスは，一般家庭にあるエアコンや洗濯機などの高度な OS を搭載していないマイコンで動いている家電の役割を実証している．

(e) **図表（参照先）**

実際のデバイスの画像とその構成を図3.19，図3.20に示す．

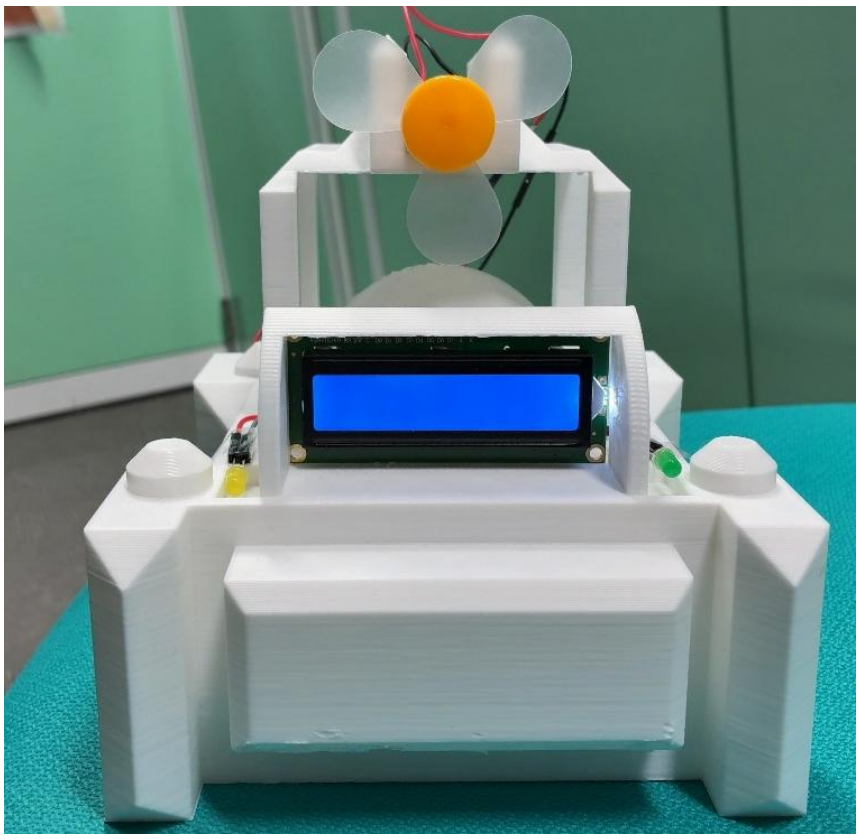

図3.19　RaspberryPi Pico W を搭載した IoT デバイス

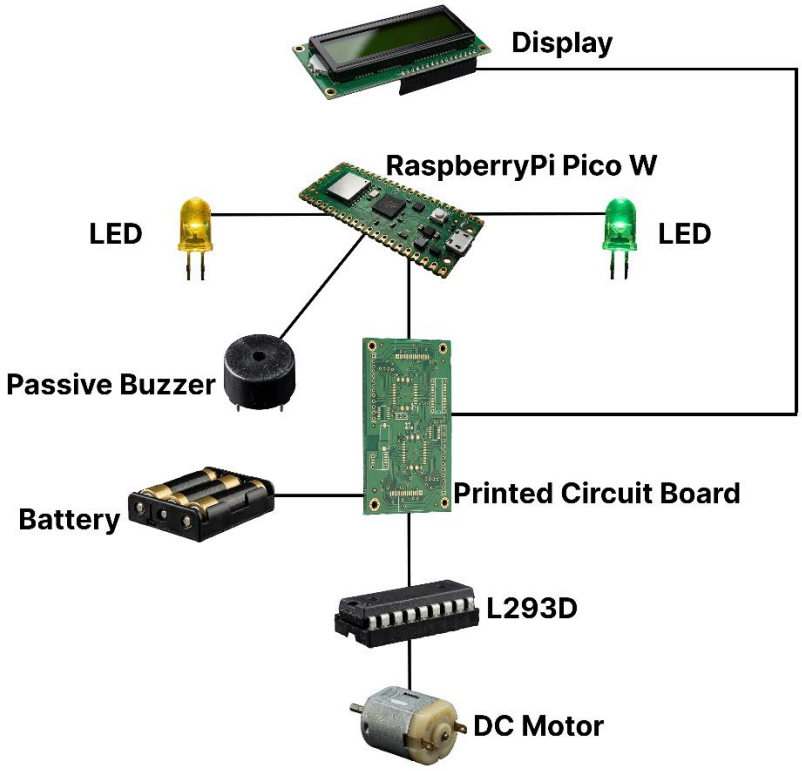

図3.20　RaspberryPi Pico W を搭載した IoT デバイスの構成
（Google Imagen 4を一部利用）

3.4　Life-Style エージェント

Life-Style エージェント（「生活支援エージェント」）は，システムにおける知識ベースとして機能し，家庭内のルール，家電の取扱説明書，生活の知恵（料理，掃除，洗濯など）といったドキュメント情報を専門に扱うエージェントである．オーケストレータからは「知識ベース機能」というエイリアスで参照される． 本エージェントの中核技術は RAG であり，事前にベクトル化された膨大なドキュメントデータベースから，ユーザーの質問や状況に関連する情報を高速に検索し，それを根拠として LLM が正確かつ自然な回答を生成する．これにより，LLM 単体の知識ではカバーしきれない，知識ベースに含まれる範囲で，個別具体的な情報に基づいたサポートを提供する．図3.21に Life-Style エージェントの画面の例を示す．

3.4.1　技術スタックとデータ処理

本エージェントの実装には，LLM アプリケーションフレームワークである「LangChain」[19]と，高速なベクトル検索ライブラリである「FAISS (Facebook AI Similarity Search)」[12]が採用されている．RAG を中心とした，Life-Style エージェントの構成を図3.22に示す．

(1) データ取り込み (Ingestion): JSONL の Question-Answer 形式の構造化データ（{"question": "～", "answer": "～"}）を処理するためのスクリプト群（「データベク

トル化スクリプト群」）を備えている．これらのスクリプトは，ドキュメントを適切なサイズのチャンクに分割し，埋め込みモデル（intfloat/multilingual-e5-large 等）[20]を用いてベクトル化し，FAISS インデックスとして永続化する．

(2) マルチインデックス構造: ベクトルデータベース（「トピック別ベクルデータベース」）は，トピックごとにサブディレクトリで分割管理されており，必要に応じて複数のインデックスを横断的に検索することが可能である．これにより，「生活の知恵」と「家電操作」といった異なるドメインの知識を効率的に管理している．

(3) 動的モデル選択: 他のエージェントと同様に，使用する LLM（Gemini, Claude, GPT, Groq）を動的に切り替えるアーキテクチャを採用している．これにより，コストや応答速度の要件に応じて最適なモデルを選択・運用することが可能である．

図3.21　Life-Style エージェントの画面例

3.4.2　RAG による回答生成プロセス

ユーザーまたはオーケストレータからの問い合わせは，以下のプロセスで処理される．

(1) 検索（Retrieval): ユーザーの質問文をベクトル化し，FAISS インデックス内から意味的に近いドキュメントチャンクを上位数件（Top-K）抽出する．

(2) 生成（Generation): 抽出されたチャンク（検索結果）を「コンテキスト」として，

質問文とともに LLM（Gemini, Claude, GPT, Groq）に提示する．システムプロンプトには「資料に基づき回答すること」「根拠となるファイル名を明示すること」といった制約が含まれており，ハルシネーション（事実に基づかない回答）を抑制している．

(3) 応答: 生成された回答テキストに加えて，参照元のファイル名やページ番号をメタデータとして返却する．これにより，ユーザーは情報の出処を確認することができる．

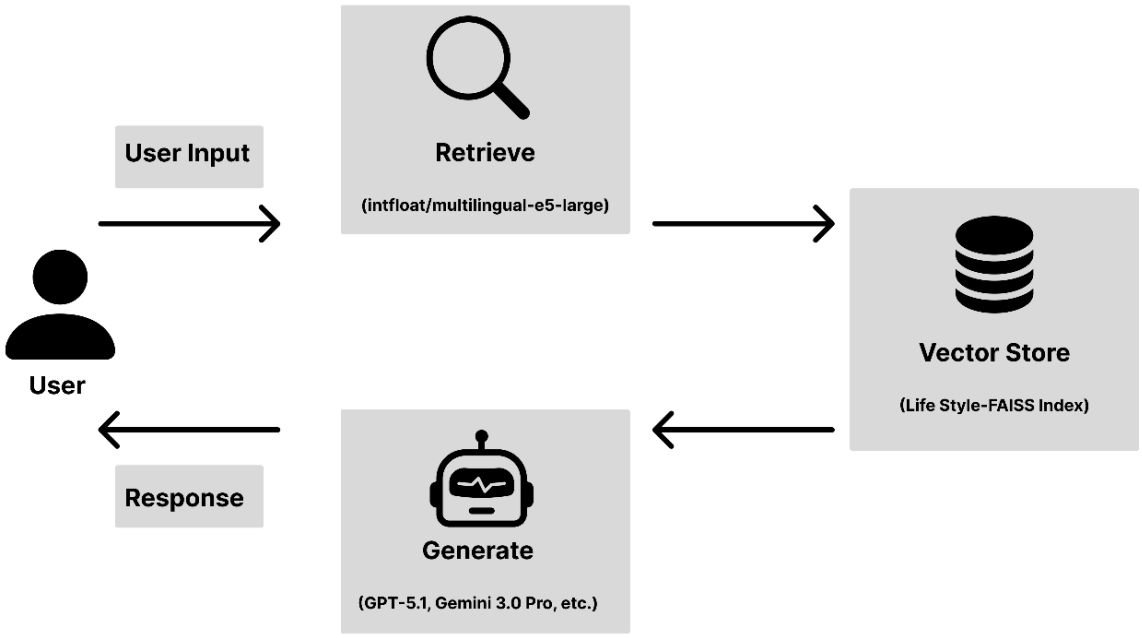

図3.22　Life-Style エージェントの構成

3.4.3　外部インターフェースと MCP

本エージェントは RESTful API に加え，MCP サーバーとしても動作し，以下のツールを外部に公開している．RAG による回答生成では，ユーザーからの生活に関する質問を受け付け，RAG プロセスを経て生成された回答を返す．会話履歴の保存有無も制御可能である．

これにより，MCP 対応のクライアントや他のエージェントは，HTTP API の詳細を知ることなく，統一されたプロトコルで知識ベース機能を利用できる．

3.5　Scheduler エージェント

Scheduler エージェント（「スケジュール管理エージェント」）は，ユーザーのルーティンワークやタスク，日記を管理するための Web アプリケーションベースのエージェントである．自然言語によるチャットインタフェースを備えており，会話を通じてスケジュールの確認やタスクの追加・完了報告を行うことができる．図3.23に Scheduler エージェントの画面の例を示す．

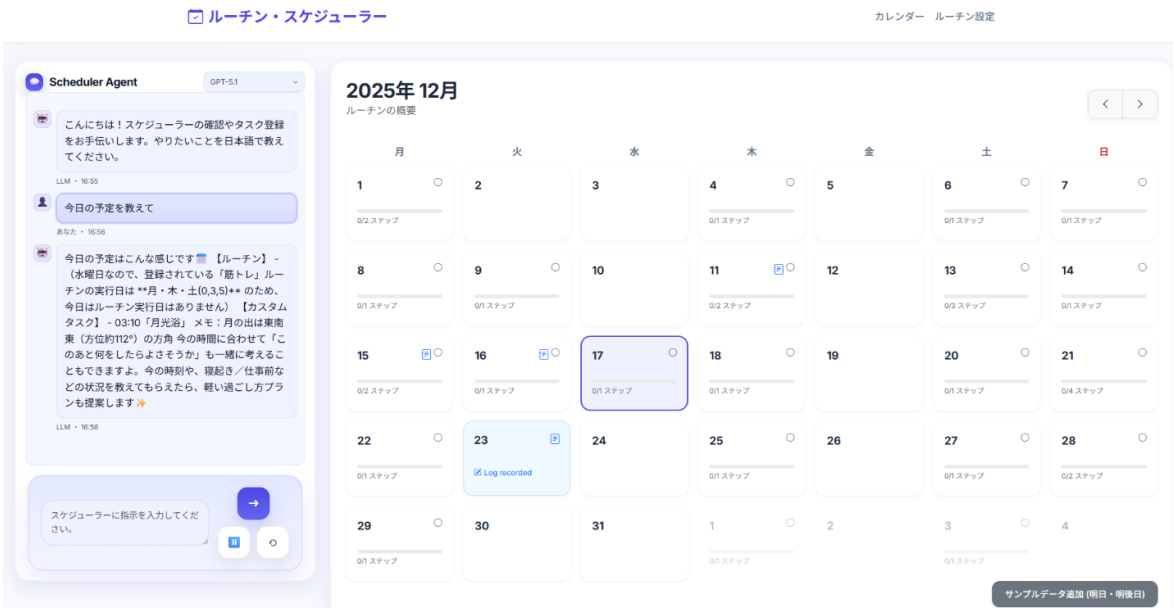

図3.23　Scheduler エージェントの画面例

3.5.1　概要と役割

本エージェントの主な役割は，ユーザーの日常生活における反復的なタスク（ルーティン）と，突発的なタスク（カスタムタスク）を効率的に管理することである．従来の TODO リストアプリとは異なり，LLM（大規模言語モデル）が組み込まれているため，ユーザーは「明日の朝8時に会議を入れたい」「今日の筋トレは終わったよ」といった自然な会話でタスクを操作できる．また，日々の活動を記録する「日記」機能も統合されており，タスクの完了状況と合わせて生活の記録を残すことが可能である．図3.24に Scheduler エージェントの構成を示す．

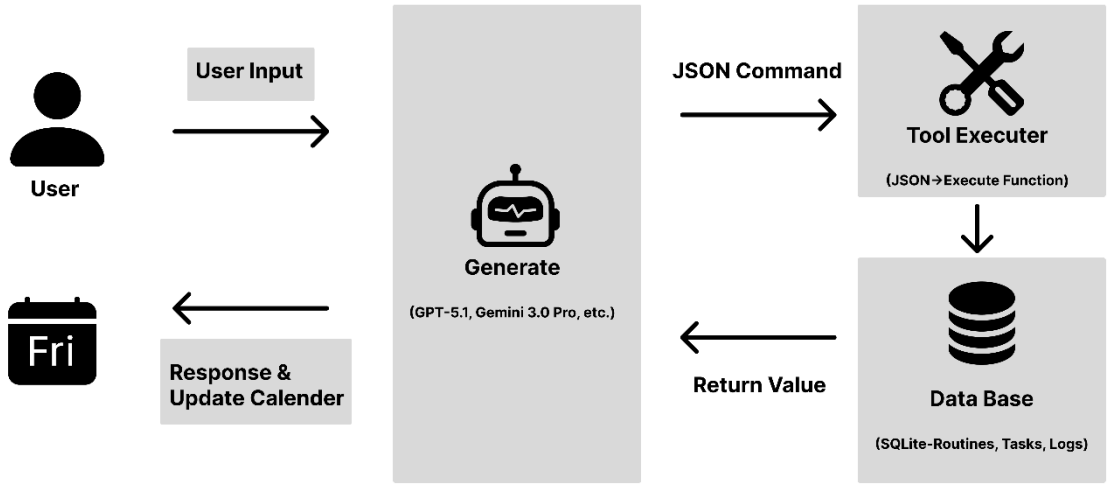

図3.24　Scheduler エージェントの構成

3.5.2　システムアーキテクチャ

本エージェントは Python の Flask フレームワークを用いた Web アプリケーションとして構築されており，データ永続化には軽量なリレーショナルデータベースである SQLite（SQLAlchemy）を採用している． システムは主に以下のコンポーネントで構成されている．

(1) Web サーバー: アプリケーションのエントリーポイントであり，画面描画（HTML テンプレートのレンダリング）と API エンドポイントの提供を行う．データベースモデル（Routine, Step, DailyLog, CustomTask, DayLog）の定義もここに含まれる．また，開発・実験用に，特定のシナリオデータをデータベースに投入する「評価用シード機能」や，モデルの回答精度を記録する「評価モジュール」も統合されている．

(2) LLM クライアント: OpenAI, Gemini, Anthropic 等の多様な LLM プロバイダとの通信を抽象化するモジュール．共通のインタフェースで各モデルを呼び出し，応答に含まれるツール呼び出し（Function Calling）を解析する．

(3) フロントエンド: HTML/CSS と Vanilla JavaScript で実装された SPA（Single Page Application）ライクなインタフェース．チャット画面とタスクリスト画面が統合されており，非同期通信でサーバーとデータをやり取りする． また，外部サイトや他のダッシュボードにカレンダを埋め込むための専用ビュー（Embeddable View）も提供しており，システムの他の画面からスケジュールを参照することも想定した設計となっている．

3.5.3　LLM による自然言語操作とツール呼び出し

Scheduler エージェントの最大の特徴は，LLM の「ツール呼び出し（Function Calling)」機能を活用した確実なタスク操作と，その実行結果に基づく応答生成フローである． ユーザーからメッセージが送信されると，システムは現在の日時，未完了タスク，ルーティン，直近の日報などをコンテキストとして構築し，LLM に送信する．

今回実装されたシステムでは，LLM がツール呼び出しを行った場合，即座にユーザーへ応答を返すのではなく，以下の「実行・要約」プロセスを経るアーキテクチャを採用している．

(1) アクション決定: LLM はユーザーの意図を解釈し，データベース操作に必要なツール呼び出し（JSON）を生成する．

(2) データベース操作: サーバー側でツール呼び出しを受け取り，実際にデータベースの更新（作成・削除・変更）を行う．この際，処理の成功・失敗を判定し，「実行結果ログ」を生成する．

(3) 応答生成（要約）: 実際の「実行結果ログ」を別の LLM（または同一 LLM の別コンテキスト）に入力し，ユーザーへの最終的な報告メッセージを生成する．

これにより，「LLM はやったと言っているが，実際には DB 更新に失敗している」といった不整合を防ぎ，確実な操作担保と，親しみやすい自然言語でのフィードバックを両立している．

利用可能な主要ツールは以下の通り大幅に拡充された．

(1) タスク管理:

(a) create_custom_task: タスクの新規作成．
(b) delete_custom_task: タスクの削除．
(c) toggle_custom_task: 完了/未完了の切り替え．
(d) rename_custom_task, update_custom_task_time, update_custom_task_memo: タスク情報の詳細編集．

(2) ルーティン管理:

(a) add_routine, delete_routine: 習慣（ルーティン）の定義と削除．
(b) add_step, delete_step: ルーティン内の各ステップの編集．
(c) toggle_step: ステップの完了/未完了の切り替え．

(3) ログ・情報取得:

(a) append_day_log: 日報への追記．
(b) update_log: 日報の上書き修正．
(c) get_day_log: 指定日の日報取得．
(d) list_tasks_in_period: 指定期間のタスク一覧取得．
(e) get_daily_summary: その日の活動（タスク・日報）の要約取得．

この多機能なツール群により，単なる予定入力だけでなく，「先週の金曜日に何をしたか確認する」「来週の予定をリストアップする」「毎日の習慣を修正する」といった高度なスケジュール管理対話が可能となっている．

3.5.4 動的なモデル選択

他のエージェント同様，Scheduler エージェントも使用する LLM モデルを動的に切り替える機能を持つ．単純なタスク登録には LLM（Gemini, Claude, GPT, Groq）といった使い分けが可能であり，コストとパフォーマンスの最適化が図られている．

4. 評価

4.1 単体性能評価

本節では，各エージェントをモデル別に単体で評価する．結果をもとに，統合評価で用いる構成やモデル選定に生かす．

4.1.1 モデル性能評価

各エージェントにおいて，使用する LLM の性能がタスク遂行能力にどの程度影響を与えるかを評価するため，パラメータ数や一般的な推論能力ベンチマークである MMLU (Massive Multitask Language Understanding) のスコアを基準に，モデルを以下の3つのクラスに分類した．なお，パラメータ数が非公開のプロプライエタリモデルについては，公表されているベンチマークスコアやプロバイダによる性能区分を参考とした．

(1) **強い (Strong):** 公開ベンチマーク（例：MMLU）で上位に位置し，一般に「フロンティア（最上位）」と見なされるモデル群．複数ステップの推論，長い指示の追従，文脈保持を比較的安定して実行できる．
※ただし MMLU やパラメータ数は万能な性能指標ではなく，ツール操作の正確さや UI 操作の成功率を直接保証しないため，本研究では「強い」は あくまで相対的な区分として扱う．

(2) **普通 (Normal):** 推論性能は十分に高いが，フロンティア級ほどの安定性・難問耐性は期待しにくい 中位（ミドルレンジ）のモデル群．一般に コスト（推論時間・API 料金・メモリ使用量）と性能のバランスが良い．

(3) **弱い (Weak):** 低コスト・低遅延を重視した 軽量（ライトウェイト／エッジ向け）モデル群．単純な分類，定型的な抽出，短い指示への応答，ルーティング（どのエージェントに回すか等）のような 軽い処理に適する．

この分類に基づき，表4.1のようにモデルの選定がシステム全体のロバスト性とコストパフォーマンスに与える影響を検証する．

表4.1 **評価に使用する LLM のモデル構成**

強度	プロバイダー	モデル ID
強い		
	OpenAI	GPT-5.1
	Google	Gemini 3 Pro
	Anthropic	Claude Opus 4.5
普通		
	Anthropic	Claude Haiku 4.5
	Groq	Llama 3.3 70B
	Groq	Qwen3 32B
弱い		
	Google	Gemini 2.5 Flash Lite
	Groq	Llama 3.1 8B
	Groq	GPT-OSS 20B

この評価により，各エージェントのタスクの複雑性に応じた最適な LLM の選択基準を確立し，システム全体のコストパフォーマンスとロバスト性を向上させるための知見を得ることを目的とする．

4.1.2 Browser エージェントの評価

Browser エージェントの評価には，自律型 AI エージェントの Web 操作能力を測定するための標準的なベンチマークである「WebArena」を採用した[21]．WebArena は，カーネギーメロン大学等の研究チームによって開発された，現実的で再現可能な Web 環境を提供するプラットフォームである．静的な HTML 解析とは異なり，実際に機能する Web アプリケーション（EC サイト，フォーラム，GitLab など）をローカル環境にホストし，人間のような複雑なタスク実行能力を「機能的正確性（Functional Correctness)」に基づいて評価する点に特徴がある．

本研究では，WebArena が提供する環境のうち，特にユーザーの利用頻度が高く，商品検索，カート追加，属性フィルタリングといった多段階の操作を要する「Shopping 環境」のみを対象として評価を行った．この環境は，オープンソースの E コマースプラットフォームである Adobe Magento を用いて構築されており，実在の商品データを含むリアルな購買体験をシミュレートしている．

評価手順は以下の通りである．システムに統合された WebArena のタスクローダーを用いて Shopping 環境向けのタスクを抽出し，以下の3つの基準に基づいて成功か失敗かを自動判定した．

(1) URL 一致 (URL Match): 操作終了時にブラウザが特定の URL（例：注文完了画面や特定の商品ページ）に遷移しているか．

(2) 文字列一致 (String Match): 画面内のテキストやエージェントの出力が，正解データ（完全一致または部分一致）を含んでいるか．

(3) DOM 状態検証 (Program HTML Check): ページ内の特定の要素（DOM）に対して JavaScript を実行し，その内容（例：カート内の合計金額，適用されたフィルタの状態）が期待通りに変化しているか．

実験は，モデルの性能差を検証するために，前節で定義した「強いモデル（GPT-5.1)」「普通のモデル（Qwen 32B)」「弱いモデル（GPT-OSS 20B)」の3つを用いて実施し，それぞれのタスク成功率（Success Rate）および完遂までのステップ数を計測した．これにより，E コマース操作における LLM の推論能力とツール操作精度の相関を分析する．
評価の結果は表4.2に示す．

表4.2　WebArena での Browser エージェントの評価

モデル	成功数 / 総数	スコア	所要時間
GPT-5.1	61 / 187	32.6%	267分 48.0秒
Qwen 32B	43 / 187	23%	155分 5.1秒
GPT-OSS 20B	49 / 187	26%	125分 3.9秒

一般にパラメータ数と性能は比例する傾向にあるが，本実験の設定では，GPT-OSS 20B（26%）が Qwen 32B（23%）を上回った．ただし，WebArena のタスク実行は（探索の分岐，ポップアップなど）実行時の揺らぎの影響を受けるため，この差をもってモデルの優劣を断定することはできない． そこで今後は，各モデルを複数回（異なる乱数シード）で実行し，成功率の平均と分散（または95%信頼区間）を算出した上で差の再現性を確認する．差が再現される場合，Web UI 操作では汎用推論能力だけでなく，操作コマンド形式への適合性や指示順守の安定性が成功率に強く寄与する可能性がある．

4.1.3 IoT エージェントの評価

IoT エージェントの単体評価では，自然言語による指示が適切にデバイス制御コマンドに変換されるか，および視覚情報を用いた状況判断が正しく行われるかを検証するために，以下の10個のタスクを作成した．これらのタスクは，単一のデバイス操作から，複数のデバイス（Jetson, Raspberry Pi 4, Raspberry Pi Pico W）を連携させる複合的な操作，さらには「暗闇」や「子供を楽しませる」といった抽象的な指示の解釈能力を問うものまで多岐にわたる．

評価基準は以下の通りである．

〇: タスクのすべてのデバイス操作が完了した場合．

△: タスクが複数デバイスの操作を想定している場合に，1つでもデバイスが操作された場合．

×: すべてのデバイス操作が失敗した場合．

(1) 動きがあるのかを3秒間確認して．

(2) 周囲の状況を確認して，撮影が終わったらブザーで成功音を鳴らして．

(3) Jetson を3秒間前進させて，停止したらディスプレイ画面に「Arrived」と表示して．

(4) Pi4のロボットに funny と表示して，手（サーボ）を縦に振って肯定して．

(5) 部屋の環境調査をして．Pico で温度を測りつつ，プロペラで風を送って．

(6) ウェルカム演出をして．Pico の画面を笑顔（happy）にして，Pi4から歓迎のメロディ（startup）を流して．

(7) 全デバイスで生存確認をして．Jetson はディスプレイに「Ready」，Pi4は「I'm here」を表示，Pico は黄色を点灯させて．

(8) 暗闇でなにかいる気がする．壁までの距離を測って，すべての led を光らせて．すべて同時に行って．

(9) 侵入者警戒モードを開始して．Jetson でディスプレイに「alert」と表示して，Pi4の LED はパトカー（police）のように点滅させて，ピコはブザーを鳴らして．

(10) 子供を楽しませるために，すべてのデバイスで何かしらのアクションをして．

評価結果は,表4.3に示す.

表4.3　IoT エージェント評価タスクにおける各モデルの回答可否

タスクID	GPT-5.1	Gemini 3 Pro	Anthropic Claude Opus 4.5	Anthropic Claude Haiku 4.5	Llama 3.3 70B	Qwen 3 32B	Gemini 2.5 Flash-Lite	Llama 3.1 8B	GPT-OSS 20B
1	○	○	○	○	○	○	○	×	×
2	○	○	○	○	○	○	○	○	○
3	○	○	○	○	×	○	○	○	○
4	△	○	○	○	△	○	○	○	○
5	○	△	○	○	△	△	△	○	○
6	○	○	○	○	○	○	○	△	○
7	△	○	○	△	○	○	○	△	○
8	△	△	○	○	○	○	○	△	△
9	○	○	○	○	△	○	○	○	○
10	○	○	○	○	○	△	△	○	△

実験の結果，全体としてパラメータ数が多い「強い」モデル群は高い指示理解能力を示したが，必ずしも全ての制御タスクにおいて完璧というわけではなかった． 例えば，**GPT-5.1** や **Claude Haiku 4.5** は，一部のタスク（No.7, No.8など）において，LED の点灯色を誤ったり，処理がタイムアウトするなど，物理デバイス特有のフィードバック待ちや複雑な条件分岐で躓くケースが見られた．一方で，**Claude Opus 4.5** や **Gemini 3 Pro** は，抽象的な指示（No.10「子供を楽しませる」）に対し，「hello kids」というメッセージ表示や顔文字の使用など，文脈を汲み取った創造的な振る舞いを見せ，高い表現力を実証した．

興味深いことに，パラメータ数が少ない「弱い」モデル群の一部，特に **Llama 3.1 8B** は，複雑な推論を要しないタスク（No.3, No.10）において，上位モデルよりも迅速かつ安定してコマンドを実行する傾向が確認された．これは，エッジデバイス制御のような即応性が求められるタスクにおいて，必ずしも巨大なモデルが最適とは限らないことを示唆している．

また，**Gemini 2.5 Flash Lite** や **Llama 3.3 70B** では，温度の取得や特定のテキスト表示（No.4, No.5）など，特定のツール呼び出しにおいて失敗が散見された．これはモデル自体の能力不足というよりは，使用したプロンプトやツール定義との相性，あるいは学習データに含まれる制御コードの偏りが影響している可能性がある．

以上のことから，IoT 制御においては，単に高知能なモデルを選ぶだけでなく，タスクの性質（即応性重視か，文脈理解重視か）に応じて適切なサイズのモデルを選定し，プロンプトエンジニアリングでモデルごとの癖を吸収することの重要性が浮き彫りとなった．

4.1.4　Life-Style エージェントの評価

Life-Style エージェントの単体評価では，以下の10個のタスクが用意されている．各タスクは，特定のカテゴリ（家電，料理，金融，メンタル）に関する質問と，それに対する正解（期待される回答）で構成されている．

評価基準は以下の通りである．

〇: 正解の中の核となる単語をすべて含んでいる．
△: 1つ以上正解の中の核となる単語を含んでいる．
×: 1つも正解の中の核となる単語を含んでいない．

(1) [家電] アイロンがけの後，戻りジワを防ぐためにすぐに行うべきことは何ですか？
正解: ハンガーで冷ますこと．

(2) [料理] アボカドの食べ頃を見分ける際，皮の色以外に確認すべきポイントはどこですか？
正解: 指で軽く押して弾力を感じるくらい（がベスト），またはヘタが取れそうなもの（も熟している）．

(3) [金融] 新 NISA において，「つみたて投資枠」の年間投資上限額はいくらですか？
正解: 年間120万円（月額10万円）．

(4) [メンタル] 怒りの感情のピークは何秒間続くとされていますか？
正解: 6秒．

(5) [料理] 鶏胸肉を焼く際，パサつかせずしっとり仕上げるための具体的な下処理方法は？
正解: 削ぎ切りにする，塩麹かマヨネーズを揉み込んで10分置く，または片栗粉をまぶして弱火で焼く．

(6) [メンタル] 不安で胸がざわざわする時に有効な「4-7-8呼吸法」の具体的なやり方は？
正解: 4秒吸って，7秒止めて，8秒かけて吐く．

(7) [料理] 今すごく疲れていて包丁を使いたくありません．冷蔵庫に豚バラと白菜があるのですが，これで作れる簡単な料理はありますか？
正解: 「豚バラと白菜の重ね蒸し」．鍋に敷き詰めて酒と鶏ガラスープの素を入れて放置するだけ．

(8) [料理] 週末に作り置きをしたい．お弁当に入れられて，かつ冷凍保存もできるおかずをいくつか提案してほしい．
正解: 鶏むね肉の南蛮漬け，ひじきと大豆の煮物，無限ピーマンの肉味噌炒めなど．

(9) [金融] 老後資金のために iDeCo と NISA で迷っている．iDeCo 特有のメリットと，逆に気をつけなければならない「制限」は何ですか？
正解: メリットは掛金が全額所得控除になり節税効果が高いこと．制限は原則60歳まで引き出せないこと．

(10) [メンタル] 明日大事なプレゼンがあるのに緊張して眠れません．無理に寝ようとして焦ってしまうのですが，どう対処すればいいですか？
正解: 部屋を暗くして「米軍式睡眠法」の筋弛緩を試す．それでも眠れない場合は「横になって目を閉じているだけでも体は8割程度休まる」と割り切る．

評価結果は,表4.4に示す．

表4.4　Life-Style エージェント評価タスクにおける各モデルの回答可否

タスクID	GPT-5.1	Gemini 3 Pro	Anthropic Claude Opus 4.5	Anthropic Claude Haiku 4.5	Llama 3.3 70B	Qwen 3 32B	Gemini 2.5 Flash-Lite	Llama 3.1 8B	GPT-OSS 20B
1	○	○	○	○	○	○	○	○	○
2	○	○	○	○	○	○	○	○	○
3	○	○	○	○	○	○	○	○	○
4	○	○	○	○	○	○	○	○	○
5	○	○	○	○	○	△	○	○	○
6	○	○	○	○	○	○	○	○	○
7	○	○	○	○	○	○	○	○	○
8	○	○	○	○	○	○	○	○	○
9	○	○	○	○	○	○	○	○	○
10	○	○	○	○	○	○	○	○	○

本評価は「核となる単語の包含」を正否基準としているため,意味的に同等な回答でも特定キーワードが欠落すると減点される．Qwen3 32B の差分は,モデル能力というより表層一致ベースの採点の限界（言い換え耐性の欠如）が影響した可能性がある．今後は同義語許容や意味類似度に基づく補助評価を導入する．

しかし，Life-Style エージェントのベクトル検索部分においては，すべての評価タスクで同一の埋め込みモデル（intfloat/multilingual-e5-large）が使用されたため，RAG の「検索（Retrieval)」フェーズにおいて LLM に渡されるコンテキストは同一であった．その結果，モデル間での回答精度に大きな差は見られなかった．これは，本評価タスクにおいては，LLM の生成能力よりも，検索フェーズ（埋め込みモデル）の質が結果に支配的な影響を与えていたことを示唆している．

4.1.5　Scheduler エージェントの評価

Scheduler エージェントの単体評価では，以下の10個のタスクが用意されている．
評価基準は以下の通りである．

○: 3回試行して全て成功．
△: 1～2回成功．
×: 全て失敗．

(1) 「洗剤を買う」というタスクを追加して．
(2) 明日から明後日までの予定を教えて．
(3) 「洗剤を買う」を「柔軟剤を買う」に名前を変えて．
(4) 「柔軟剤を買う」タスク，完了した．
(5) 明日の15:30分から「歯医者」の予定を入れて．
(6) やっぱり「柔軟剤を買う」は削除して．そして,「歯医者」のタスクに「保険証を忘れない」というメモを追加しておいて．
(7) 「筋トレ」という新しいルーティンを作って．月曜と木曜にやるよ．

(8) さっき作った「筋トレ」ルーティンに,「スクワット」というステップ（10分）を追加して.

(9) 「筋トレ」ルーティン，やっぱり土曜日もやることにする.

(10) 先週の金曜日の日報を見せて.

評価結果は,表4.5に示す.

表4.5　Scheduler エージェント評価タスクにおける各モデルの回答可否

タスク ID	GPT-5.1	Gemini 3 Pro	Claude Opus 4.5	Claude Haiku 4.5	Llama 3.3 70B	Qwen3 32B	Gemini 2.5 Flash Lite	Llama 3.1 8B	GPT-OSS 20B
1	○	○	○	○	○	○	×	○	○
2	○	○	○	○	○	○	×	○	○
3	○	○	○	○	○	○	×	△	○
4	○	○	○	×	○	○	×	○	○
5	○	○	○	○	○	○	×	△	○
6	○	○	○	×	○	○	×	○	×
7	○	○	○	×	○	○	△	○	○
8	○	○	○	×	○	○	×	○	○
9	○	○	○	×	×	○	×	△	○
10	×	○	○	○	×	○	△	×	×

各モデルの主な失敗要因:

(1) GPT-5.1: タスク10において日付計算ミスが発生した．検証日（2025年12月9日・火曜日）に対し,「先週の金曜日」を2025年11月28日と誤認した（正解は12月5日）．曜日や今日の日付自体は正しく認識していたものの，週を跨ぐ計算で誤りが見られた.

(2) Claude Haiku 4.5: タスク4, 6, 7, 8, 9において，ツール呼び出し（JSON 生成）が行われず，ただのテキスト応答のみとなるケースが多発した（No tool calls）.

(3) Llama 3.3 70B: タスク9で曜日指定を誤り（土曜日ではなく日曜日），タスク10では日付を1日ずれて計算（12月6日）するミスが見られた.

(4) Gemini 2.5 Flash Lite: 全体的に指示に従った JSON 生成ができず，失敗が多かった．システムプロンプトに含まれる『ユーザー向け応答内での JSON 出力禁止』という制約を,ツール呼び出しにおける JSON 生成にまで拡大解釈して適用したと考えられる．(ユーザーへの返答:コマンド実行のための JSON)という出力をすることが正解だが，JSON の生成を不正な出力，あるいは禁止された動作であると判定した可能性が高い.

(5) Llama 3.1 8B: タスク3や5において，日本語の引数生成時に「演溪角を起ん」のような意味不明な文字列が混入する現象が見られた．トークナイザまたは学習データの日本語カバレッジの問題と考えられる．タスク10では日付計算を誤った（12月4日を参照）.

(6) GPT-OSS 20B: タスク6のような複合指示（削除と追加）において，片方の指示（削除のみ実行し，追加を忘れる）が無視される傾向があった．タスク10では大幅に日付を誤った（12月2日を参照）．

(7) Qwen3 32B: 「普通」クラスのモデルでありながら，全ての問題に正解した．Scheduler エージェントのタスクにおいて高い適性を示している．

4.2 シナリオベースの統合評価

本節では，システム全体が連携してユーザーの曖昧な指示をどの程度適切に実行できるかを評価する．特に，パーソナライズされた「メモリ」の有無が，タスクの成功率や実行効率に与える影響を検証する．

4.2.1 実験設定と評価指標

本評価実験で使用する各エージェントの LLM モデルは，4．1節の単体性能評価の結果に基づき，各タスクにおいて最もコストパフォーマンスと性能のバランスが優れていた，あるいは特定のエラーが少なかったモデルを選定している．各エージェントの使用するモデルは表4.6に示す．

(1) 実験設定:

(a) **ベースライン（メモリなし):** ユーザーの個人情報を保持していない状態．

(b) **メモリあり:** 個人データ（住所，趣味，健康状態など）に最適化された行動をとる状態．

(i) **サブタスク:** 「ユーザーの曖昧な指示」をどれだけ適切に具体化できたかを評価する．

(ii) なお，本評価で用いる住所・趣味・健康状態などの個人データは評価用に作成した架空データであり，実在個人を特定し得る情報は含まない．

(2) 評価指標:

(a) **達成度:** 各シナリオに設定された複数の条件（サブゴール）ごとに，達成（○）か未達成（×）を判定する．

(b) **スコア算出:**

(i) 達成（○）1つにつき1点を加算する．

(ii) エージェントからの質問1回につき1点を減点する．

※ この重み付けは，ユーザーへの追加負担（問い合わせ対応による認知的負荷や時間的コスト）を，1つのサブゴール達成というポジティブな価値と等価なネガティブ要因として定義したものである．

(iii) 各シナリオの素点 G = ○の数

(iv) 質問回数 Q

(v) 本研究における「質問回数」とは，エージェントがタスク遂行に必要な追加情報（例：日時・場所・対象の特定）をユーザーに要求し

た発話を1回としてカウントしたものである．

(vi)スコア $= \max(0, G - Q)$

補足情報: ブラウザのステップ数（あくまで参考情報）．

表4.6　使用モデル構成

エージェント	モデル
オーケストレータ	Claude Opus 4.5
IoT エージェント	Claude Haiku 4.5
Browser エージェント	GPT-5.1
Scheduler エージェント	Qwen3 32B
Life-Style エージェント	Llama 3.3 70B

4.2.2　評価シナリオと基準

以下の10のシナリオを用いて評価を行う．

(1) シナリオ1

(a) **ユーザー指示:**「今日も元気に頑張ろう！」

(b) **タスク内容:** ユーザーの抽象的な発話に対し，オーケストレータが適切な返答を選択・生成するタスク．

(c) **基準1:** オーケストレータ単体での回答を選択できるか．

(2) シナリオ2

(a) **ユーザー指示:**「一週間後の天気予報を確認して，その日の天気情報を記録に残して」

(b) **タスク内容:** メモリ上の居住地情報に基づく週間天気予報の検索，および取得した情報の記録を行うタスク．

(c) **基準1:** メモリからユーザーの居住地域近くの天気を検索できるか．

(d) **基準2:** 適切に記録できるか．

(3) シナリオ3

(a) **ユーザー指示:**「東京からアトランタまでの片道航空券を，2026年1月中の任意の平日出発で探して，その中の最安値の便をカレンダーに登録して．」

(b) **タスク内容:** 航空券検索サイトを用いた特定路線の最安値便の調査，およびカレンダーへの予定登録を行うタスク．

(c) **基準1:** 飛行機の情報を調査したか．

(d) **基準2:** その情報をカレンダーに登録したか．

(4) シナリオ4

(a) **ユーザー指示:**「今夜の月が見える方角と時間を調べて，その時間に『月光浴』の予定を入れたら，ライトを消して．」

(b) **タスク内容:** 居住地における月の出の方角と時間の調査，スケジュールへの予定登録，および IoT デバイスによる照明消灯を連携して行うタスク．

(c) **基準1:** ユーザーの居住地における月の方角を調査したか．

(d) **基準2:** 予定を登録したか．

(e) **基準3:** デバイスのライトを消灯したか.

(5) シナリオ5

(a) **ユーザー指示:** 「ネットで地震が起きた時の避難マニュアルを確認した後に，避難所での人間関係の注意点を把握したら，避難訓練として警告灯（赤色LED）を点滅させて.」

(b) **タスク内容:** オンライン上の避難マニュアルの確認，Life-Style エージェントによる避難所生活の助言取得，および IoT デバイスによる警告灯（赤色LED）の点灯制御を順次実行するタスク.

(c) **基準1:** 避難マニュアルを確認できるか.

(d) **基準2:** Life-Style エージェントで適切に人間関係の確認ができるか.

(e) **基準3:** IoT デバイスを操作できるか.

(6) シナリオ6

(a) **ユーザー指示:** 「夕飯のレシピ考えて，それを今日のタスクに追加したら，デバイスに”complete”と表示して.」

(b) **タスク内容:** ユーザーの健康状態や嗜好を考慮したレシピ提案，タスクリストへの追加，およびデバイスディスプレイへの完了通知表示を行うタスク.

(c) **基準1:** 健康志向やアレルギーに配慮したレシピ提案ができるか.

(d) **基準2:** 今日のタスクに追加されたか.

(e) **基準3:** いずれかのディスプレイに”complete”と表示されたか.

(7) シナリオ7

(a) **ユーザー指示:** 「私の趣味に関連するイベントを探して，教えて．そして，そのイベントがある日の予定にスケジュールを追加して.」

(b) **タスク内容:** ユーザーの趣味に関連するイベント情報の検索（居住地周辺），および該当イベントのスケジュール登録を行うタスク.

(c) **基準1:** 趣味に関連し，かつ評価時点で今後開催予定のイベントを検索できたか.

(d) **基準2:** 住所（鎌倉，中目黒，大宮）周辺の情報を検索できたか.

(e) **基準3:** スケジュールに登録できたか.

(8) シナリオ8

(a) **ユーザー指示:** 「大好きなあの食べ物があるお店は，近くにないかな？あれば今日のメモに残した後に，ディスプレイに英語でそのお店の名前を表示して.」

(b) **タスク内容:** ユーザーの好物を提供する近隣店舗の検索，メモへの記録，および店舗名のディスプレイ表示を行うタスク.

(c) **基準1:** 趣味（好物）に関連する検索ができたか.

(d) **基準2:** メモに残せたか.

(e) **基準3:** 店の名前を表示できたか.

(9) シナリオ9

(a) **ユーザー指示:** 「近くでピクニックができそうな場所を調べてほしい．その

あと，家族でどのように予定を話し合って立てればよいのかを教えてほしい．そして，ピクニックの予定を来週の日曜日に入れて，完了後にブザーを鳴らして教えて.」

(b) **タスク内容:** 近隣の行楽地情報の検索，家族会議の進め方に関する助言取得，スケジュール登録，および完了通知としてのブザー鳴動を複合的に行うタスク．

(c) **基準1:** 住んでいる近くのピクニックスポットを調査できたか．

(d) **基準2:** Life-Style エージェントが回答したか．

(e) **基準3:** スケジュールに登録したか．

(f) **基準4:** ブザーが鳴動したか．

(10) シナリオ10

(a) **ユーザー指示:** 「いつものルーティンを実行して.」

(b) **タスク内容:** ユーザーが定義した定型処理（ルーティン）の認識と，それに含まれる複数のアクションを順次実行するタスク．

(c) **基準1:** いつものルーティンを認識できるか．

(d) **基準2:** ルーティンに記載されている1つ目の動作を実行できるか．

(e) **基準3:** ルーティンに記載されている2つ目の動作を実行できるか．

メモリ有り無しの評価結果は，表4.7，表4.8，表4.9，表4.10に示す．

4.2.3 評価結果

表4.7　ベースライン（メモリなし）評価結果

シナリオ	基準1	基準2	基準3	基準4	質問回数	ステップ数(ブラウザ)	スコア
1	○	—	—	—	0	—	1
2	×	○	—	—	0	19	1
3	×	×	—	—	0	19	0
4	×	×	○	—	0	16	1
5	○	○	○	—	0	17	3
6	×	○	○	—	0	—	2
7	○	×	○	—	1	15	1
8	×	○	○	—	0	3	2
9	×	○	○	○	0	7	3
10	×	○	○	—	1	—	1
合計						96	15

(1) シナリオ2: 2025年の12月24日を検索できた．ただし，東京の天気を検索していた．

(2) シナリオ4: 質問されることなく，東京の情報が検索された．

(3) シナリオ6: 健康志向や好みを聞いてくることはなく，Life-Style エージェントに登録されているのレシピを1つ提案したのみであった．

(4) シナリオ7: 質問で，趣味に関することは質問できたが，住所については聞いてこ

なかった．ただし，2025年のイベントを検索できていた．これにより，年次の推定（日時解釈）にばらつきが見られることが分かった．

(5) シナリオ8: 一般的に人気が高いと推定されるラーメン店を検索していた．ただし，すぐに完了した．

表4.8　ペルソナ1（メモリあり）評価結果

シナリオ	基準1	基準2	基準3	基準4	質問回数	ステップ数(ブラウザ)	スコア
1	○	—	—	—	0	—	1
2	○	○	—	—	0	20	2
3	×	×	—	—	0	20	0
4	○	○	○	—	0	4	3
5	○	○	○	—	0	14	3
6	○	○	○	—	0	—	3
7	×	○	○	—	0	20	2
8	○	○	○	—	0	18	3
9	○	○	○	○	0	19	4
10	○	○	○	—	0	—	3
合計						115	24

(1) シナリオ3: 飛行機の日付選択でつまずき，そのまま失敗した．

(2) シナリオ5: 適切にエージェントの選択ができていた．

(3) シナリオ6: 血圧・そばアレルギーに配慮した塩分控えめの夕飯レシピを提案した．

(4) シナリオ7: 鎌倉周辺の②12/7「三浦三崎マグロ争奪将棋大会」③12/9 9:15「俳句（北風）（鎌倉婦人子供会館)」をスケジュールに登録したが，いずれも評価時点では開催済みであったため基準1を×とした．

(5) シナリオ10: 正しくルーティンを認識して実行した．

表4.9　ペルソナ2（メモリあり）評価結果

シナリオ	基準1	基準2	基準3	基準4	質問回数	ステップ数(ブラウザ)	スコア
1	○	—	—	—	0	—	1
2	○	○	—	—	0	11	2
3	○	○	—	—	0	17	2
4	○	○	○	—	0	9	3
5	○	○	○	—	0	19	3
6	○	○	○	—	0	—	3
7	×	○	○	—	0	19	2
8	○	○	○	—	0	16	3
9	○	○	○	○	0	12	4
10	○	○	○	—	0	—	3
合計						103	26

(1) **シナリオ7:** 現在が2025年の12月だと理解しているが，既に過ぎている,2024年12月 ～ 2025年1月 のイベントを検索してしまっていた．そして，2025年の1月に予定を入れてしまった．
(2) **シナリオ9:** 適切に中目黒の公園を検索できていた．
(3) **シナリオ10:** メモリから情報を取得し，正確に実行できていた．

表4.10　ペルソナ3（メモリあり）評価結果

シナリオ	基準1	基準2	基準3	基準4	質問回数	ステップ数(ブラウザ)	スコア
1	○	—	—	—	0	—	1
2	○	○	—	—	0	7	2
3	○	○	—	—	0	18	2
4	○	○	○	—	0	4	3
5	○	×	○	—	0	10	2
6	○	○	○	—	0	—	3
7	×	○	○	—	0	17	2
8	○	○	○	—	0	19	3
9	○	○	○	○	0	3	4
10	○	○	○	—	0	—	3
合計						78	25

(1) **シナリオ2:** ユーザーの居住地域の情報を取得できた．
(2) **シナリオ4:** 視認性が高く構造が平易なサイトに最初にアクセスしたため，ステップ数が少なかった．
(3) **シナリオ5:** 人間関係のことは Life-Style エージェントに割り振ることが正解であるが，基準2も Web 検索をしてしまった（通常なら成功としてもよいが，適切にタスクを意図したとおりに割り振れていないため，失敗とした）．
(4) **シナリオ6:** アレルギー，好みを配慮した回答ができていた．
(5) **シナリオ7:** 2024年の1月のイベントを検索してしまっていた．オーケストレータも Browser エージェントも両方とも今が何年なのかを理解しているはずであるが，ミスが発生した．
(6) **シナリオ8:** 近くのハンバーグの店を適切に検索していた．
(7) **シナリオ9:** 近くの公園を検索して，詳細を調べずに即答したためステップ数が減少した．

Browser エージェントによる現在日時の取得が正確に行われておらず，2024年の情報を調べてしまう場合がある．これは Browser エージェントのシステムプロンプトが膨大であるため，日時の情報の重要性が薄れてしまっている可能性がある．オーケストレータから Browser エージェントへの情報の受け渡しにおいて，2025年の12月某日と渡すと正確な情報が取得できるが，単に12月某日と渡すと2024年の情報を検索してしまう．これは，モデルの知識カットオフによるものと考えられる．

また，ブラウザ操作の効率性（ステップ数）についても考察を加える．実験前は，メモリ機能によりユーザーの嗜好や前提条件が明確化されることで，Browserエージェントが迷わず目的の情報へアクセスでき，結果としてステップ数が削減されると仮説を立てていた．しかし，実際の結果において，メモリの有無によるステップ数の有意な差は確認されなかった．これに対し，エージェントの行動ログを詳細に分析すると，ステップの「質」に変化が生じていることが示唆された．

メモリなしの場合，エージェントは一般的な情報を広く浅く探索する傾向が見られた．対してメモリありの場合，ユーザーの「好み」や「具体的な居住地」といった制約条件との整合性を確認するため，詳細ページの閲覧や情報の照合といった「検証的なステップ」が増加していた．この結果は，メモリ機能が単なる操作ステップの短縮（効率化）をもたらすのではなく，多少のコストを払ってでも出力結果の適合性を最大化しようとする「行動変容」を LLM に促したことを示唆している．これは，失敗が許されない実生活支援タスクにおいて望ましい挙動である．

各評価ケースにおける総合スコアを比較すると表4.11の通りである．

表4.11　メモリの有無を比較した評価結果

評価対象	総合スコア（点）
1.　メモリなし (Baseline)	15
2.　ペルソナ1 (メモリあり)	24
3.　ペルソナ2 (メモリあり)	26
4.　ペルソナ3 (メモリあり)	25

この結果から，メモリ機能（ユーザープロファイルやコンテキストの保持）を有効にすることで，ベースラインと比較して大幅にスコアが向上することが確認された．特に曖昧な指示に対する解釈精度や，文脈に応じた適切なツール選択において改善が見られる．

Evaluation target	Overall score (points)
1.　No memory (Baseline)	15
2.　Persona 1 (with memory)	24
3.　Persona 2 (with memory)	26
4.　Persona 3 (with memory)	25

5. 考察

本章では，第4章で得られた評価結果に基づき，提案システムの有効性，現状の課題，および将来の展望について考察する．

5.1 メモリ機能によるパーソナライズと効率化

シナリオベースの統合評価（4.2節）において，メモリ機能を有効にした場合のスコア（平均25.0点）は，無効にした場合（15点）と比較して著しく高い結果となった．これは，本システムの中核である「オーケストレータ」と「メモリ」が，ユーザーの曖昧な指示を具体的かつ適切なタスクに変換する上で極めて有効に機能したことを示している．

特筆すべきは，追加質問の減少である．メモリなしのベースラインでは，場所や好みの特定ができず，一般的すぎる回答（例：全国的な天気，一般的なレシピ）に終始するか，あるいはユーザーへの確認が必要となる場面が想定された．一方，メモリありのケースでは，「いつもの」「近くの」といった指示に対し，住所情報や過去の履歴（アレルギー情報，趣味など）を自律的に参照することで，ユーザーの意図を正確に補完できている．例えばシナリオ6において，ユーザーの「そばアレルギー」や「高血圧」といった健康情報を考慮したレシピ提案が行われた点は，単なる自動化を超えた「パーソナライズされた執事」としての価値を示している．

この成功は，オーケストレータが持つ「長期メモリ（ユーザーの住所，健康状態，趣味などの永続的な属性情報)」と「短期メモリ（直近の興味関心や一時的なコンテキスト)」が協調して機能した結果である．具体的には，以下の種類のメモリがタスク成功に特に寄与した．

(1) **住所情報:** シナリオ2「一週間後の天気予報を確認して，その日の天気情報を記録に残して」およびシナリオ4「今夜の月が見える方角と時間を調べて，その時間に『月光浴』の予定を入れたら，ライトを消して.」では，メモリに保存されたユーザーの居住地域の住所情報（例：鎌倉，中目黒，大宮）を参照することで，Browser エージェントが「東京の天気」や「一般的な月の方角」ではなく，ユーザーにとって関連性の高い地域の情報を正確に検索できた．これにより，曖昧な指示に対する解釈精度が向上し，ユーザーは具体的な場所を指定する手間を省くことができた．

(2) **健康状態（アレルギーや持病など）：** シナリオ6「夕飯のレシピ考えて，それを今日のタスクに追加したら，デバイスに”complete”と表示して.」において，メモリに登録されていたユーザーの「そばアレルギー」や「高血圧」といった健康状態を考慮し，Life-Style エージェントが塩分控えめでアレルゲンを含まないレシピを提案した．これは，単に一般的なレシピを提示するだけでなく，個人の健康状態に合わせたパーソナライズされた情報提供が可能であることを示しており，ユーザーの満足度向上に直結する．

(3) **好み・趣味:** シナリオ7「私の趣味に関連するイベントを探して，教えて．そして，そのイベントがある日の予定にスケジュールを追加して.」およびシナリオ8「大

好きなあの食べ物があるお店は，近くにないかな？あれば今日のメモに残した後に，ディスプレイに英語でそのお店の名前を表示して.」では，長期メモリに保存されたユーザーの趣味（例：将棋，俳句）や好物（例：ラーメン，ハンバーグ）を参照することで，Browser エージェントが関連イベントや店舗情報を効率的に検索し，Scheduler エージェントがスケジュール登録やメモへの追記を行った．メモリがない場合，システムはユーザーに趣味や好物を尋ねる必要があり，タスクの完遂までに余計なステップを要する．メモリによってこれらの情報が事前に把握されていることで，タスクの実行がよりスムーズかつ直接的になり，ユーザーの体験が向上した．

これらの事例から，単にメモリの有無だけではなく，どのような種類の個人情報がどのエージェントに利用されるかを適切に設計し，保持することで，システムのパーソナライズ能力と効率性が飛躍的に向上することが明らかになった．

5.2 LLM の知識カットオフと時間認識の課題

評価実験を通じて浮き彫りになった主要な課題の一つは，LLM の「知識カットオフ」と「現在日時の認識」との乖離である．システムプロンプトで現在日時（2025年12月）を明示しているにもかかわらず，エージェントが「2024年」の情報（イベントやカレンダー）を優先して検索・参照してしまう事象が複数回確認された（シナリオ2, 7）．
これは，LLM の学習データに含まれる期間の重みが，プロンプトで与えられた短期的なコンテキストよりも強く出力に影響したためと考えられる．特に Web 検索クエリの生成において，明示的に「2025年」と含めない限り，モデルが「現在（学習データ上の最新）」と認識している2024年以前の情報を探索してしまう傾向が見られた．この問題は，LLM を実社会のリアルタイムなタスクに適用する際の共通課題であり，プロンプトエンジニアリングによる強制（クエリへの年号付与の徹底）や，RAG による直近情報の重み付け強化といった対策が必要である．

本課題に対する具体的な解決策として，オーケストレータから各専門エージェントへタスクを割り振る際，あるいはユーザーからのプロンプト入力の都度，システム側で現在の日時情報（タイムスタンプ）を明示的に付与する手法が考えられる．これにより，エージェントは常に最新の時間軸を認識した状態で推論を行うことが可能となり，学習データ内の古い情報に引きずられることなく，正確な日時情報に基づいた検索やスケジュール管理を実行できると期待される．

5.3 複雑な Web UI 操作の限界と可能性

Browser エージェントの評価（4.1.2節およびシナリオ3）からは，現在の LLM ベースのエージェントにとって，複雑な Web UI 操作が依然として高いハードルであることが確認された．本実験条件（WebArena）では，使用モデル（GPT-5.1）でも成功率は約25%に留まったことや，飛行機予約サイトの日付選択ウィジェットの操作に失敗した事例は，DOM 解析ベースのアプローチの限界と LLM の状況判断能力の限界を示唆している．

動的に変化する DOM 構造や，視覚的な配置に依存した操作（カレンダーのクリックな

ど）は，テキストベースの HTML 解析だけでは文脈を捉えきれない場合が多い．一方で，シンプルな検索や情報抽出（シナリオ2, 4）では高い成功率を示しており，定型的なタスクにおいては十分実用的である．今後は，GPT-5.1や Gemini 3 Pro のようなマルチモーダルモデルを活用し，DOM だけでなく「画面のスクリーンショット」を併用して視覚的に UI を理解させるアプローチが，突破口になると考えられる[22].

5.4　エッジ AI とクラウド AI の役割分担

IoT エージェントの評価（4.1.3節）において，Jetson や Raspberry Pi 上で動作する数 B（数十億）パラメータクラスの軽量 LLM（Qwen3 1.7B[17], Llama-3.2 3B[18]）が，JSON 構造の記述誤り（シンタックスエラー）を起こすことなく,デバイスの動作に成功した点は注目に値する.

これは，適切なツール定義（Function Calling）とシステムプロンプトを与えれば，エッジ側の小規模モデルでも十分に「推論器」として機能することを示している．すべての処理をクラウドの超高性能モデル（GPT-5.1等）に依存するのではなく，プライバシーに関わるデータの利活用，センサー処理や即応性が求められるデバイス制御はエッジで行い，高度な推論が必要な計画立案はクラウドで行うという，本システムの「階層型アーキテクチャ」の妥当性が裏付けられたと言える.

6. むすび

本研究では，分断された Web サービスと物理デバイス（IoT）を統合し，ユーザーの曖昧な自然言語指示から自律的にタスクを実行する「協調型マルチエージェントシステム」を設計・開発した． 提案システムは，個人の嗜好や文脈を保持する「メモリ付きオーケストレータ」を中心に，ブラウザ操作，IoT 制御，生活知識検索，スケジュール管理を担う専門エージェント群を MCP によって標準化して接続する構成とした．

評価実験の結果，以下の知見が得られた．

第一に，長期・短期メモリを用いたコンテキスト認識の有効性である．ユーザーの属性や過去の履歴を記憶することで，ベースラインと比較してタスク実行の成功率とスコアが大幅に向上し（約1.7倍），ユーザーへの質問回数は，ベースラインの合計2回に対し，メモリありの3ケースでは合計0回となった（表4.7～表4.10）．これにより，「いつもの」といった人間らしい曖昧な指示に対しても，システムが意図を汲み取り，パーソナライズされた行動をとることが可能となった．

第二に，エッジ AI とクラウド AI の協調動作の実用性である．Jetson や Raspberry Pi などのエッジデバイス上で動作する軽量 LLM（1.7B～3B パラメータ）が，適切なプロンプトエンジニアリングの下で，抽象的な指示を物理的な制御コマンドへ正確に変換できることを実証した．これにより，プライバシーや即応性が求められる処理をエッジ側で完結させつつ，高度な判断のみをクラウドに委ねる階層的な知能処理が有効に機能することを確認した．

一方で，課題も明らかとなった．LLM の学習データに起因する時間認識のズレ（知識カットオフ問題）や，複雑な GUI を持つ Web サイト操作における DOM 解析の限界である．これらは，LLM を実世界の動的な環境に適用する際の共通の壁であり，今後はマルチモーダルモデルによる視覚的理解の導入や，RAG システムのリアルタイム性強化が不可欠である．

総括として，本研究は，単なるツールの自動化に留まらず，AI が個人の文脈を理解し，Web と現実世界を横断して生活をサポートする「パートナー」となり得る可能性を示した．今後は，本システムの実運用を通じて長期的な記憶の変容プロセスを検証するとともに，より多様なサービスやデバイスとの接続を進め，人と AI が共生するスマートな生活環境の実現に貢献することが期待される．

参考文献

[1] Schick, T. et al.: Toolformer: Language Models Can Teach Themselves to Use Tools, arXiv:2302.04761, https://arxiv.org/abs/2302.04761（2025/12/18参照）.

[2] Home Assistant：2024.6: Dipping our toes in the world of AI using LLMs , Home Assistant Blog, https://www.home-assistant.io/blog/2024/06/05/release-20246/（2025/12/18参照）.

[3] Wu, Q. et al.: AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation, arXiv:2308.08155, https://arxiv.org/abs/2308.08155（2025/12/18参照）.

[4] Fourney, A. et al.: Magentic-One: A Generalist Multi-Agent System for Solving Complex Tasks, arXiv:2411.04468, https://arxiv.org/abs/2411.04468（2025/12/18参照）.

[5] Mozannar, H. et al.: Magentic-UI: Towards Human-in-the-loop Agentic Systems, arXiv:2507.22358, https://arxiv.org/abs/2507.22358（2025/12/18参照）.

[6] Cui, H. et al.: LLMind: Orchestrating AI and IoT with LLM for Complex Task Execution, arXiv:2312.09007, https://arxiv.org/abs/2312.09007（2025/12/18参照）.

[7] Shen, Y. et al.: HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face, arXiv:2303.17580, https://arxiv.org/abs/2303.17580（2025/12/18参照）.

[8] agno-agi：Agno (AgentOS), GitHub, https://github.com/agno-agi/agno（2025/12/18参照）.

[9] King, E. et al.: Sasha: Creative Goal-Oriented Reasoning in Smart Homes with Large Language Models, arXiv:2305.09802, https://arxiv.org/abs/2305.09802（2025/12/18参照）.

[10] Anthropic：Model Context Protocol, https://modelcontextprotocol.io/（2025/12/18参照）.

[11] Rawat, M. et al.: Pre-Act: Multi-Step Planning and Reasoning Improves Acting in LLM Agents, arXiv:2505.09970, https://arxiv.org/abs/2505.09970（2025/12/18参照）.

[12] Facebook Research：FAISS, GitHub, https://github.com/facebookresearch/faiss（2025/12/18参照）.

[13] browser-use：browser-use, GitHub, https://github.com/browser-use/browser-use（2025/12/08参照）.

[14] LangChain：LangGraph, https://python.langchain.com/docs/langgraph/（2025/12/18参照）.

[15] Adimi, Alaa Dania：Augmenting LLMs with Retrieval, Tools, and Long-term Memory, Medium, https://medium.com/infinitgraph/augmenting-llms-with-retrieval-tools-and-long-term-memory-b9e1e6b2fc28（2025/12/19参照）.

[16] Zhong, W. et al.: MemoryBank: Enhancing Large Language Models with Long-Term Memory, arXiv:2305.10250, https://arxiv.org/abs/2305.10250（2025/12/19参照）.

[17] Unsloth：Qwen3-1.7B-Q4_K_S.gguf, Hugging Face, https://huggingface.co/unsloth/Qwen3-1.7B-GGUF/blob/main/Qwen3-1.7B-Q4_K_S.gguf（2025/12/08参照）.

[18] Unsloth：Llama-3.2-3B-Instruct-Q4_K_M.gguf, Hugging Face, https://huggingface.co/unsloth/Llama-3.2-3B-Instruct-GGUF/blob/main/Llama-3.2-3B-Instruct-Q4_K_M.gguf（2025/12/08参照）.

[19] LangChain：LangChain, https://python.langchain.com/（2025/12/18参照）.

[20] intfloat：multilingual-e5-large, Hugging Face, https://huggingface.co/intfloat/multilingual-e5-large（2025/12/10参照）.

[21] WebArena：WebArena: A Realistic Web Environment for Building Autonomous Agents, https://webarena.dev/（2025/12/18参照）.

[22] Chae, H. et al.: Web Agents with World Models: Learning and Leveraging Environment Dynamics in Web Navigation, arXiv:2410.13232, https://arxiv.org/abs/2410.13232（2025/12/18参照）.