Mobile Computing Final Project
CAPMA: Context-Aware Personalized Mobile Agent

Chanbin Lee

June 24, 2025

1 Motivation

Traditional mobile virtual assistants such as Siri and Bixby, while deeply integrated into mobile operating systems, exhibit limited flexibility in handling complex, personalized tasks. Their capabilities are typically confined to predefined commands or shallow contextual understanding, making them less effective for dynamic, user-driven interactions that demand deeper semantic reasoning.

Recent advancements in Multimodal Large Language Models (MLLMs) have led to the development of powerful mobile agent systems capable of interpreting open-ended user instructions. However, these systems primarily focus on instruction-following behavior and often lack a user-friendly, personalized response mechanism. They typically operate in a vacuum, unaware of the user's real-time context or past interactions, which limits their ability to deliver truly meaningful assistance.

To address this gap, our project proposes a real-time context-aware mobile agent pipeline that enhances MLLM-based agents with live sensor data and personalized user knowledge. By integrating environmental signals (e.g., user activity, location, and audio context) with semantically relevant user data retrieved through a retrieval-augmented generation (RAG) pipeline, we aim to significantly improve the quality, relevance, and usability of agent responses.

This approach enables a more intelligent and responsive assistant that can not only understand what the user wants, but also why and under what circumstances—leading to more effective device control, natural interaction, and ultimately, a more human-centric mobile agent experience.

2 Development Environment

Android Development Setup

- **IDE:** Android Studio
- **Build System:** Gradle
- **Minimum SDK Version:** Android 10 (API level 29)
- **Target SDK Version:** Android 14 (API level 34)
- **Compile SDK Version:** API level 34
- **Programming Language:** Java
- **Architecture:** AndroidX-based (e.g., `androidx.appcompat`)

Key Features and Components

- **Activity Recognition:** via Google Play Services
- **Location Awareness:** using Google Places API
- **Speech-to-Text:** OpenAI Whisper API integration
- **Semantic Search:** Custom text embedding module (`SentenceEncoder`)

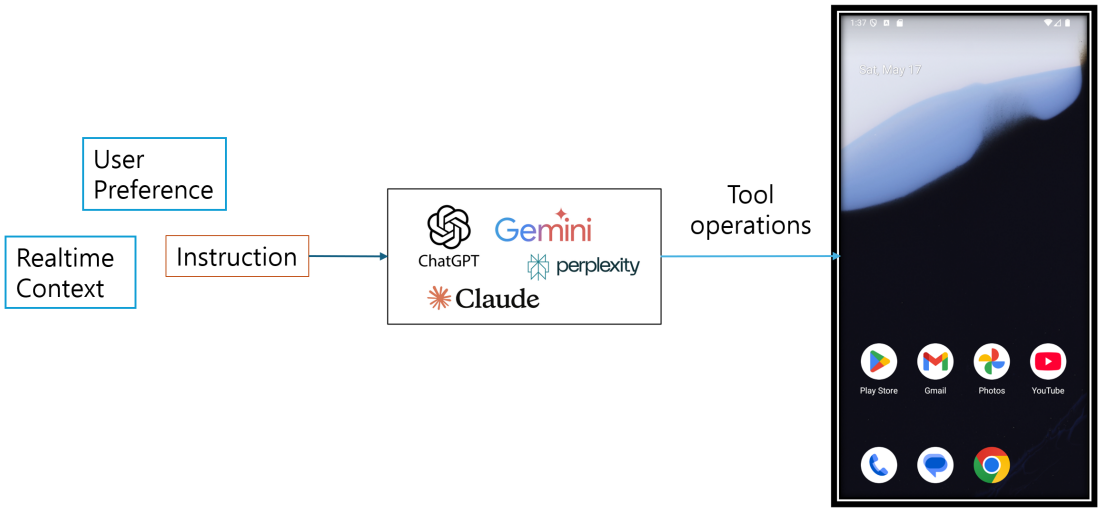

Figure 1: The simplified view of our system.

- **Local Storage:** Room database with note embeddings
- **UI:** Material Components, RecyclerView
- **Text-to-Speech:** Android `TextToSpeech` API

Device Capabilities Required

- Microphone (audio recording)
- Location services (GPS or network-based)
- Internet connection (for API communication)

Permissions

- `RECORD_AUDIO`
- `ACCESS_FINE_LOCATION`, `ACCESS_COARSE_LOCATION`
- `ACTIVITY_RECOGNITION`
- `INTERNET`
- `FOREGROUND_SERVICE`, `FOREGROUND_SERVICE_DATA_SYNC`
- `POST_NOTIFICATIONS`

Backend Integration

- Custom API Client (`AgentApiClient`)
- Communicates with external server (configurable URL)
- Sends contextual data and receives responses

Background Operation

- Foreground service with persistent notification
- Restart via broadcast receiver
- Auto-start on device boot

Libraries and Dependencies

Core Android Libraries

- `androidx.appcompat:appcompat`
- `androidx.core:core-ktx`
- `androidx.activity:activity`
- `androidx.recyclerview:recyclerview`
- `androidx.constraintlayout:constraintlayout`

Material Design

- `com.google.android.material:material`

Google Services

- `com.google.android.gms:play-services-location`
- `com.google.android.gms:play-services-activity-recognition`
- `com.google.android.libraries.places:places`

Room Database

- `androidx.room:room-runtime`
- `androidx.room:room-compiler`
- `androidx.room:room-ktx`

Networking and JSON

- Java HTTP client or OkHttp
- Gson or Jackson for JSON parsing

AI and NLP Integration

- OpenAI Whisper API for speech recognition
- Custom `SentenceEncoder` for embedding and semantic search
- TensorFlow Lite backend
- OpenAI Embedding API available

Concurrency and Threads

- `java.util.concurrent.ExecutorService`
- `android.os.Handler`

External Resources

- Material Design icons (vector drawables)
- Google Places API key (in app resources)
- OpenAI API key (in app resources)

Build Tools

- Gradle (build automation and dependency management)
- Android SDK tools and plugins

3 Service and Application Design

System Overview

CAPMA (Context-Aware Personalized Mobile Agent) is designed as a modular mobile system that senses environmental context, processes multi-modal input data, and provides very useful additional information to the MLLM agent to make intelligent, personalized control possible. The system architecture emphasizes modularity, resilience, and user-centered design, enabling it to operate robustly in diverse mobile conditions.

User Interface Design

The CAPMA application provides a modular interface aligned with its architecture. It consists of three major UI components: the main control screen, the note management screen, and the settings screen. Each screen provides direct interaction with a different system module.

Main Control Screen

As shown in Figure 2, the main screen acts as the central hub for initiating sensing and navigating the app. It includes three core buttons:

- **Start Sensing:** Begins real-time sensing of activity, location, and audio.
- **Notes:** Navigates to the personal note database.
- **Settings:** Opens configuration options for sensing and retrieval.

Below these buttons, the system displays the current server URL and sensing status. Once sensing begins, this area is also used to present sensing results and responses from the agent server.

Notes Management Screen

Figure 3 shows the interface for user note management. This screen supports:

- **Add/Edit/Delete:** Basic note operations
- **Pinning:** Pinned notes are always included in retrieval, regardless of the user query

Pinned notes represent persistent semantic knowledge that can enhance contextual responses in a broad range of scenarios.

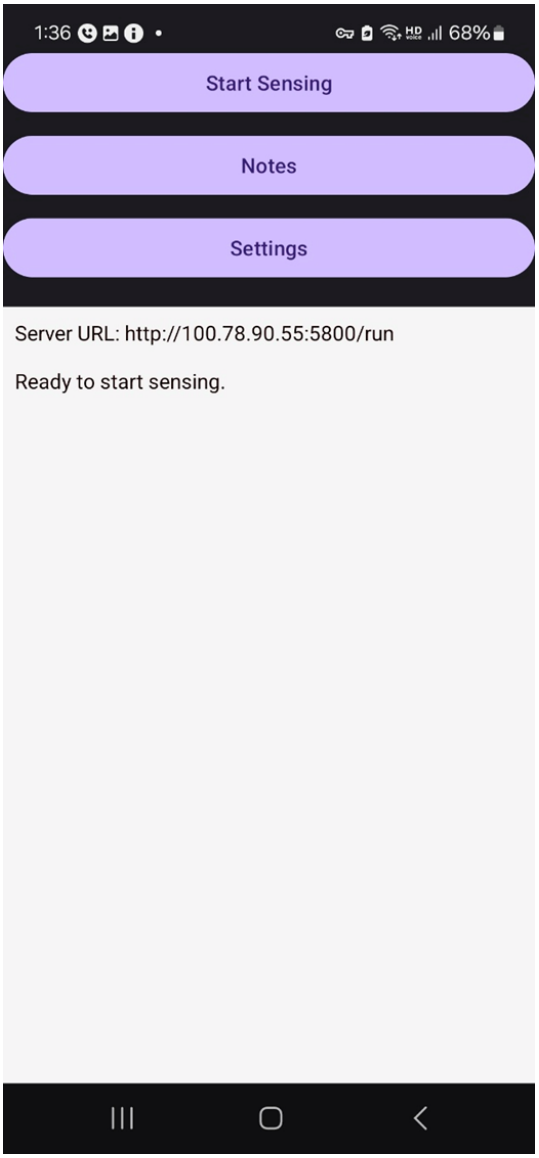

Figure 2: Main interface with sensing control and navigation buttons

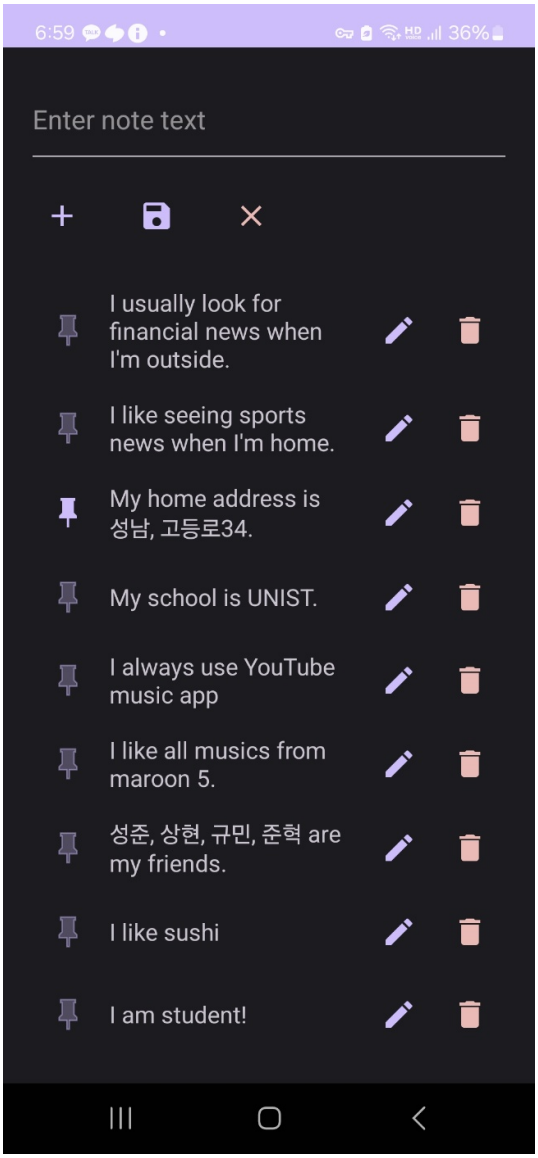

Figure 3: Note management interface with support for editing, deleting, and pinning notes

Settings – Server and Sensing

In Figure 4, users can configure the connection to the external agent server and toggle sensing modules. This includes:

- **Server IP/Port:** Manually configure the target server for agent communication
- **Sensing Options:** Toggle activity recognition, place awareness, and audio noisiness detection

Settings – Retrieval Configuration

As illustrated in Figure 5, the retrieval configuration screen allows the user to fine-tune semantic search behavior:

- Enable or disable note retrieval
- Set the maximum number of results

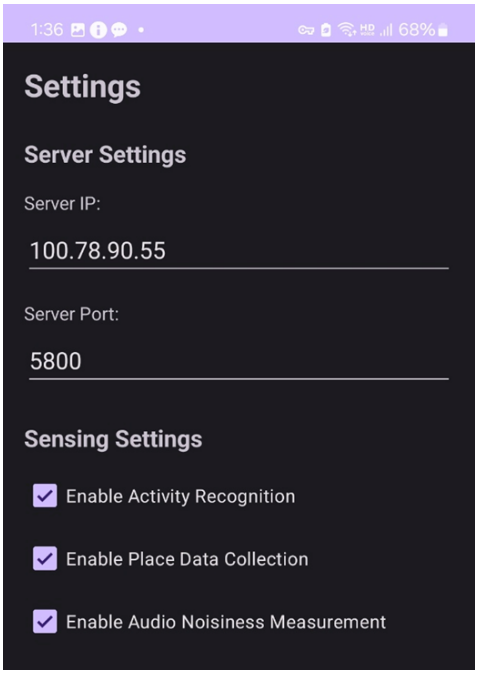

Figure 4: Settings screen for server address and sensing module control

- Choose an embedding model: USE, BERT, or OpenAI Embeddings API

These options give the user control over personalization and retrieval precision.

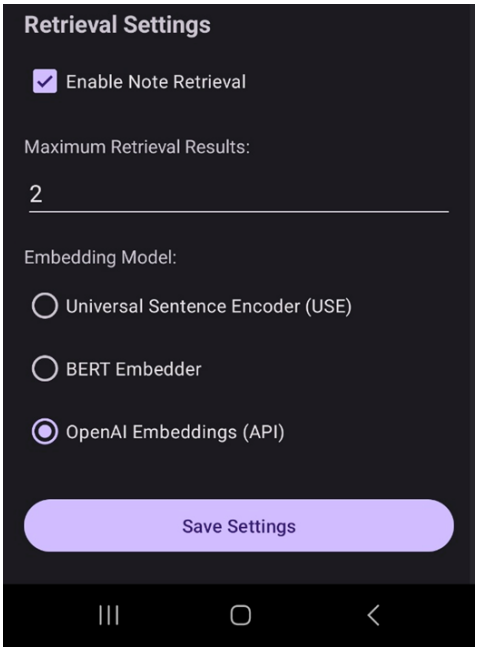

Figure 5: Retrieval settings screen with embedding model and result limit options

System Architecture

1. Context Collection Engine

- **Purpose:** Collects real-world contextual data from device sensors and APIs.

- **Subcomponents:**
  - *Activity Recognition Module:* Detects user activity states (e.g., walking, stationary).
  - *Location Awareness Module:* Identifies user's semantic location using Google Places API.
  - *Audio Sensing Module:* Captures environmental audio for transcription and noisiness estimation.

2. Natural Language Processing Pipeline

- **Purpose:** Transforms audio queries into semantically meaningful representations.
- **Subcomponents:**
  - *Speech-to-Text Service:* Converts recorded speech to text via OpenAI Whisper API.
  - *Embedding Generator:* Converts query text into vector embeddings.
  - *Retrieval System:* Performs semantic matching against user knowledge base.

3. Knowledge Management System

- **Purpose:** Stores, retrieves, and indexes user-authored notes.
- **Subcomponents:**
  - *Note Storage:* Room database for persistent storage.
  - *Vector Database:* Embedding vectors for semantic search.
  - *Retrieval Engine:* Matches embeddings with query context.

4. Agent Communication Interface

- **Purpose:** Bridges mobile device context with external intelligent services.
- **Subcomponents:**
  - *Context Formatter:* Aggregates and formats context data into API-compatible payloads.
  - *Agent API Client:* Manages HTTP communication with the external agent.
  - *Response Processor:* Parses responses and routes them to the UI.

5. Persistence and Background Operation

- **Purpose:** Maintains continuous operation regardless of foreground state.
- **Subcomponents:**
  - *Foreground Service:* Ensures persistence with notification interface.
  - *Service Recovery:* Automatically restarts on termination.
  - *Boot Receiver:* Launches service on device startup.

6. User Interface Layer

- **Purpose:** Provides user access to sensing controls, responses, and knowledge management.
- **Subcomponents:**
  - *Main Interface:* Allows initiation of sensing and displays contextual data.
  - *Notes Interface:* Enables creation and review of personal notes.
  - *Settings Interface:* Controls API endpoint, permissions, and retrieval parameters.

7. MLLM Agent Server Integration

- **Purpose:** Executes intelligent actions on the device based on user queries and context, using a Multimodal Large Language Model (MLLM).
- **Architecture:** The agent server runs externally on a computer and acts as the execution layer for commands generated by the MLLM.
- **Functionality:**
  - Receives structured context and user queries from the CAPMA mobile app.
  - Utilizes a MLLM-based backend to interpret the intent and generate appropriate device control commands.
  - Sends resulting ADB (Android Debug Bridge) commands to the connected mobile device for execution.
- **Codebase:**
  - `https://github.com/mcp-use/mcp-use` — Framework for connecting the mobile app with the agent server.
  - `https://github.com/mobile-next/mobile-mcp` — Backend MLLM-powered MCP (Mobile Control Platform) agent server implementation.

Data Flow Architecture

1. Context Collection Flow

`User Environment` → `Sensors` → `Raw Data` → `Processing Modules` → `Structured Context Data`

2. Query Processing Flow

`User Speech` → `Audio Recording` → `Speech-to-Text API` → `Text Query` → `Embedding Generation` → `Context Enhancement`

3. Knowledge Retrieval Flow

`Query Embedding` → `Similarity Search` → `Ranked Notes` → `Context Augmentation`

4. Agent Interaction Flow

`Enhanced Context` → `API Request` → `Agent Service` → `Response` → `Presentation`

Service Integration Model

- **External Services:**
  - OpenAI Whisper API (speech recognition)
  - Agent API (intelligent contextual responses)
  - Google Places API (semantic location data)
- **Internal Services:**
  - Embedding Service (vector representation of queries and notes)
  - Database Service (Room storage and access)
  - Background Service (persistent sensing operation)

User Interaction Model

Explicit Interaction

- User initiates sensing manually
- User speaks a query
- User manages stored notes

Implicit Interaction

- Activity recognition runs continuously in the background
- Places are identified automatically
- Ambient noise levels are used for context estimation

Response Presentation

- Automatic device control by the MLLM agent's ADB commands
- Text-to-speech output for mobile agent's text output

4 Implementation Details

Main Application Components

CAPMAApplication.java

- Initializes global application state
- Starts background service on app launch
- Handles notification permission for Android 13+

MainActivity.java

- Central hub for context sensing and user interaction
- Requests permissions for location, audio, and activity recognition
- Manages data collection and API communication
- Handles audio transcription and text-to-speech
- Binds to background service for persistent operation

Context Sensing Components

Activity Recognition

ActivityRecognitionManager.java

- Uses Google Play Services for activity updates
- Provides formatted activity data and callbacks

ActivityRecognitionService.java

- Background service receiving activity transition events

Location and Places

PlacesManager.java

- Interfaces with Google Places API
- Retrieves and formats current place data
- Manages location permission checks

Audio Processing

AudioRecorder.java

- Manages audio recording sessions and files
- Provides callbacks for completion and error handling

WhisperApiClient.java

- Uploads audio to OpenAI Whisper API for transcription
- Handles API responses and errors

Data Management Components

Note Management

TextNote.java

- Represents a note with text, timestamp, and vector embedding

TextNoteDao.java

- Defines Room database operations (CRUD, search, etc.)

AppDatabase.java

- Singleton Room database configuration and access

Embedding and Retrieval

SentenceEncoder.java

- Generates vector embeddings for input text
- **USE:** https://storage.googleapis.com/mediapipe-models/text_embedder/universal_sentence_encoder/float32/latest/universal_sentence_encoder.tflite
- **BERT:** https://storage.googleapis.com/mediapipe-models/text_embedder/bert_embedder/float32/latest/bert_embedder.tflite

Manages underlying model and resources

RetrievalManager.java

- Performs semantic search over stored notes
- Ranks results based on similarity

API Communication

AgentApiClient.java

- Formats context data and sends to external agent API
- Manages HTTP requests and handles responses

UI Components

NotesActivity.java

- Manages note creation, editing, deletion
- Interfaces with embedding and database
- Updates RecyclerView

NoteAdapter.java

- RecyclerView adapter for note display
- Handles UI interactions (edit/delete)

SettingsActivity.java

- Manages server URL and feature toggles
- Stores user preferences

Background Services

BackgroundService.java

- Foreground service with persistent notification
- Handles lifecycle, status updates, and self-restart

ServiceRestartReceiver.java

- Listens for boot and kill events to restart service

Layout Files

- **activity_main.xml**: Main screen with controls and result display
- **activity_notes.xml**: Interface for note input and listing
- **activity_settings.xml**: Configuration screen with toggles and URL input
- **item_note.xml**: Individual note layout with edit/delete buttons

Resource Files

- **Drawable Resources:** Icons (edit, delete, add, save, cancel), button shapes
- **String Resources:** UI texts, default settings
- **Network Configuration:** Security policies, cleartext traffic settings

5 Result

5.1 Embeddings

We provide three options for text embeddings. USE and BERT models are saved in TFLite format and runs locally on the device. We found that it generates embedding very quickly, allowing it possible to use in real scenarios. In contrast, the OpenAI embeddings api provides more accurate and distinctive embeddings, that gives better retrieval results in many cases. However it has cost per call, and the embedding generation time is relatively slow comapred to the local models.

Embedding Model	Latency (ms)
USE (Universal Sentence Encoder)	29
BERT	82
OpenAI API	900

Table 1: Average Latency by Embedding Model, generating single embedding vector.

5.2 Real-Use Cases

To validate the effectiveness of CAPMA and its MLLM-based control pipeline, we present several real-world use cases that demonstrate the agent's behavior under various conditions. Each case highlights a distinct capability of the system, including normal command execution, personalization, contextual adaptation, and failure modes.

The vidoes are available with sound, and be careful for the loud sounds when playing.

5.2.1 Normal Device Control

In `simple_working.mp4`, the user gives a basic instruction: *"Open Google Chrome"*. The system successfully executes this command, and the connected laptop's agent server can be seen processing the request and sending the ADB command.

In `complex_working.mp4`, a more advanced instruction is given: *"Open YouTube, search for recent US political news, play any video from the result, and subscribe to the channel."* The agent manages this multi-step request effectively, assuming that the instruction is clearly structured.

These examples demonstrate that the base MLLM agent is capable of executing both simple and moderately complex commands accurately.

5.2.2 Context-Aware Control

In `adjust_volume.mp4`, the user asks to adjust the volume appropriate to the current environment. As the sensing module provides data that the environment is very quiet, the agent lowers the media volume to 30.

5.2.3 Personalized Control

In `personalized_control.mp4`, the user asks: *"Turn on my favorite music."* No explicit music name is mentioned. However, the system retrieves two relevant user notes:

- `User always uses YouTube Music.`
- `User's favorite group is Maroon 5.`

These retrieved notes are included in the prompt sent to the agent. As a result, the system successfully plays *"Payphone" by Maroon 5* on YouTube Music.

This case highlights the effectiveness of the personalization pipeline—contextually relevant user data is retrieved and used to enrich vague commands.

5.2.4 Context-Aware Personalized Control

The video `integrated_control.mp4` demonstrates a highly context-sensitive interaction. When the recording took place, the user was **walking around their home**, and their home address had been **pinned in the notes**.

The user repeatedly, simply asks for the news. The system retrieves relevant knowledge:

- `User likes sports news when at home.`
- `User looks for financial news when outside.`

Because the current GPS location matched the pinned home address, the system inferred that the user was at home and displayed **sports news**.

Later, when **place data was disabled** in the settings, and the user was still physically at home but **walking**, the system instead inferred the user was outside based on activity recognition. As a result, it displayed **financial news**.

This example showcases how the agent makes use of all available contextual information (location, activity, user info) to make the best decision. It also emphasizes the importance of accurate and complete user-provided knowledge.

6 Discussion

6.1 Improving MLLM agents

As we show in the results, our app functions well as the bridge of the user and the MLLM agent. But the performance of the entire system depends hardly on the performance of the MLLM agent, that actually controls the device.

When we just use the vanilla version of the agents, it often behaves wierd, or does not execute the correct operations. MLLM agents are research field that is still ongoing actively, and it has many limitations in the performance, currently. For our system, we tried to gain the performance by giving some well-designed system prompts and some control tips. These control tips provide some human-like ways to control the device in many scenarios, so it could give the results we want.

System Prompt

You are a helpful assistant that can help with tasks on a mobile device. User's instruction, current activity, place data, audio data, and user's information related to the instruction will be provided. When given user preferences or context, act decisively on their behalf. Don't offer options—make the best choice and execute it.
If user wants something, you should do it.

Control Tips

1. You can swipe down from the top to see the notifications.
2. To play certain music in music app, click the search icon and search keyword and click the song to play.
3. Use Google Chrome to search for information.
...

6.2 Future Works

This project was an experimental system, based on the interest of combining useful user information and the ability of MLLM agents. We showed the possibility that this kind of integrated system could make the device control more user-friendly. We used basic contexts like activity, location, audio. But in our device, there are many other information that we can get. Not only about the user, the current state of device itself can be an useful information that can be given to agents.

Also, for the user preference data, we used internal notes that user can write down in the app. But in real world, data about the user is not only in fixed-form text, and could be a multimodal data that can be collected from various sources. Improved system may use these data for better control, inside the fence of user's permission about the privacy.