

Thiết kế Companion Android: Nghe – Nhìn – Học (Phase 1)

Kiến trúc Hệ thống (Android-First)

Tổng quan kiến trúc: Hệ thống gồm smartphone Android (Samsung Note 20 Ultra) đóng vai trò trung tâm xử lý **AI on-device**, giao tiếp với một Arduino Mega (điều khiển robot) qua Bluetooth SPP. Ứng dụng Android đảm nhiệm *nghe và hiểu giọng nói, nhìn và nhận biết hình ảnh, học có giám sát và ra quyết định hành vi*, đồng thời gửi lệnh điều khiển phần cứng robot. Mô hình kiến trúc **Android-first** giúp xử lý thời gian thực ngay trên thiết bị di động mà không phụ thuộc đám mây, đảm bảo độ trễ thấp và bảo vệ dữ liệu người dùng ¹. Dưới đây là sơ đồ các thành phần chính và luồng dữ liệu (các tiến trình Nghe/Nhìn chạy song song):



Mô tả luồng dữ liệu: Người dùng nhấn và giữ nút **PTT (Push-to-Talk)** trên giao diện để ra lệnh bằng giọng nói. Âm thanh thu từ **Mic** được **ASR** (ví dụ Android Speech API) xử lý hoàn toàn *on-device* để chuyển thành văn bản (nhằm giảm độ trễ và không phụ thuộc mạng ¹). Văn bản sau nhận dạng được gửi tới **Intent Router**, thành phần này phân tích câu nói và xác định ý định (intent) của người dùng theo

luật (MVP dùng kịch bản rule-based cho các intent như chào hỏi, yêu cầu mô tả, học tên, truy xuất trí nhớ, điều khiển, v.v.). **Dialogue Manager** quản lý trạng thái hội thoại (ví dụ biết liệu đang trong chế độ “dạy” hay không, hoặc theo dõi lượt nói) và có cơ chế *điều phối lượt thoại*, cho phép người dùng **barge-in** – tức ngắt lời trợ lý bất kỳ lúc nào bằng cách nói, hệ thống sẽ dừng phát TTS để lắng nghe lệnh mới

2 . 3 .

Song song với kênh âm thanh, camera điện thoại kích hoạt **Vision module** (dựa trên **CameraX**), liên tục xử lý khung hình. Bộ xử lý hình ảnh này gồm **Face/Person Detection** (phát hiện khuôn mặt/người) và **Object Detection** (nhận dạng vật thể thông dụng) thực thi bằng model nhẹ trên máy (TFLite MobileNet SSD hoặc MediaPipe) đạt tốc độ ~15 FPS ⁴. Với mỗi khung hình, **Vision Processor** sẽ: (1) Phát hiện **khuôn mặt** và gán ID theo dõi tạm thời (để biết vị trí người di chuyển giữa các khung), (2) Phát hiện **vật thể** xung quanh và gắn nhãn (sử dụng mô hình học máy nhúng đã huấn luyện trên bộ dữ liệu COCO hoặc tương tự). Kết quả nhận diện (danh sách khuôn mặt/vật thể kèm vị trí, độ tin cậy, nhãn nếu đã biết) được gửi tới **Behavior Planner** và đồng thời cập nhật lên **UI** (ví dụ về **bounding box** và nhãn trên khung hình preview). Toàn bộ pipeline *nhin* này chạy liên tục trên GPU/CPU điện thoại, *không bị tạm dừng khi robot nói hoặc nghe*, đảm bảo Companion luôn “mở mắt” quan sát môi trường.

Brain - Xử lý ngôn ngữ & Hoạch định hành vi: **Intent Router** phân loại câu lệnh đầu vào thành các intent cụ thể. Ví dụ: “Xin chào” -> intent `greet`; “Bạn đang thấy gì?” -> intent `describe_scene`; “Đi tới” -> intent `move(forward)`; “Quay trái” -> `move(left)`; “Nhìn theo mình” -> intent `track_person`; “Dạy tên đồ vật này là bình sữa” -> intent `teach(object, label="bình sữa")`; “Nhớ mặt mình là Ba” -> intent `teach(face, label="Ba")`; “Ai đây?” -> intent `recall(face)`; “Cái này là gì?” -> intent `recall(object)`; “Quên Ba” -> intent `forget("Ba")`. Sau khi Router xác định được intent và tham số, **Dialogue Manager** sẽ xét ngữ cảnh hiện tại (ví dụ đang trong phiên dạy hay bình thường) và quyết định phản hồi hoặc hành động phù hợp.

Tiếp đó, **Behavior Planner** ánh xạ intent sang tập *hành vi* cụ thể cho robot: bao gồm phản hồi lời nói (qua **TTS**), cử chỉ (quay đầu camera theo người, nháy đèn LED), hoặc di chuyển thân robot (qua lệnh động cơ). Planner có thể kết hợp cả thông tin từ kênh nhìn – ví dụ nếu intent `track_person` (theo dõi người gần nhất) thì Planner sẽ lấy vị trí khuôn mặt từ Vision để tính toán góc servo phải quay, sau đó phát lệnh tương ứng. Đối với intent hỏi thoại, Planner chuẩn bị câu trả lời để **TTS** phát, và có thể kèm hiệu ứng LED (như nhấp nháy màu vui nhộn khi chào hỏi).

Memory Store (bộ nhớ): Đây là thành phần lưu trữ lâu dài các **đối tượng** (vật thể) và **khuôn mặt** mà người dùng đã “dạy” cho robot. Mỗi mục nhớ bao gồm: loại (FACE/OBJECT), **nhãn** do người dùng đặt, **đặc trưng nhận dạng** (ví dụ vector embedding khuôn mặt hoặc đặc trưng ảnh vật thể), và siêu dữ liệu (ngày giờ học, tần suất gặp lại, v.v.). Khi Vision phát hiện một khuôn mặt/vật thể mới, hệ thống có thể so khớp đặc trưng với cơ sở dữ liệu này để xem có nhận ra đối tượng quen thuộc không ⁵. Nếu độ tương đồng vượt ngưỡng, Companion sẽ nhớ tên và có thể chào hoặc mô tả đúng tên đã học; nếu thấp, có thể yêu cầu người dùng xác nhận (tránh nhận nhầm). Tất cả dữ liệu này được lưu **on-device** (dùng Room hoặc Proto DataStore), không gửi ra ngoài, đảm bảo tính riêng tư. Cách tiếp cận **so khớp embedding on-device** này đã được chứng minh hiệu quả: ví dụ, app nhận diện khuôn mặt on-device sử dụng FaceNet TFLite lưu trữ vector 128-D rồi so sánh *k-lâng giềng gần nhất* để nhận dạng danh tính mà không cần máy chủ ⁵ ⁶.

Device Bridge (Bluetooth SPP): Cầu nối Bluetooth Classic (SPP profile) chịu trách nhiệm gửi lệnh thời gian thực từ điện thoại tới Arduino, và nhận lại phản hồi/telemetry. Behavior Planner tạo ra các lệnh dạng **DRIVE/ SERVO/ LED** và đẩy vào Bridge; Bridge sẽ đóng gói thành khung nhị phân theo giao thức (mô tả chi tiết ở mục sau) để truyền qua Bluetooth với tần số 20-50 Hz. Ngược lại, Arduino có thể gửi dữ

liệu cảm biến (như khoảng cách, IMU) hoặc ACK/Ping, Bridge giải mã và chuyển tới thành phần tương ứng (ví dụ TelemetryRepo để log hoặc Behavior Planner nếu cần ra phản xạ khẩn cấp).

App UI (Jetpack Compose): Giao diện người dùng được xây dựng bằng Compose, cung cấp các **màn hình** và **nút chức năng** để tương tác với Companion. Người dùng có thể: Chọn và **kết nối Bluetooth** tới module (HC-05) của robot; giữ **nút Mic (PTT)** để nói (kích hoạt ASR); theo dõi **camera preview** kèm khung phát hiện (Vision sẽ cung cấp dữ liệu để vẽ overlay qua Canvas); nhấn nút “**Dạy**” (Teach) để vào chế độ học thủ công nếu cần (MVP chủ yếu qua lệnh thoại nhưng UI có thể hỗ trợ nút này). UI cũng hiển thị một **log hội thoại** tóm tắt các lệnh gần nhất và phản hồi của Companion (giúp debug và UX).

Tính năng đa nhiệm & thời gian thực: Kiến trúc sử dụng lập trình bất đồng bộ (Coroutines + Flow) để đảm bảo **camera** và **microphone** hoạt động đồng thời mà không chẵn nhau. Quá trình nhận diện hình ảnh chạy trên luồng riêng (sử dụng khả năng binding lifecycles của CameraX), trong khi ASR lắng nghe trên một luồng khác. Khi Companion đang **nói** (TTS phát tiếng) thì camera vẫn tiếp tục nhận dạng người/vật (chỉ ASR tạm ngừng thu âm để tránh feedback âm thanh chính nó). Nếu người dùng nói chen (PTT hoặc VAD kích hoạt) **trong lúc TTS đang phát, barge-in policy** sẽ lập tức dừng TTS và chuyển sang chế độ nghe để tiếp nhận lệnh mới ³. Cách thiết kế này giúp tương tác tự nhiên: người dùng không cần đợi robot nói xong hoàn toàn mới được ra lệnh. Hệ thống cũng có logic **VAD (Voice Activity Detection)** dự phòng để tự động phát hiện khi nào người dùng bắt đầu nói, tuy nhiên MVP sẽ đơn giản dùng PTT để tránh phức tạp ban đầu.

Tóm lại, kiến trúc trên đảm bảo Companion Android có thể “**vừa nghe vừa nhìn**” một cách mượt mà, xử lý hầu hết tác vụ AI ngay trên thiết bị. Điều này tối ưu **độ trễ end-to-end ~<700ms** cho phản hồi lời nói (theo khuyến nghị ~800ms để tạo trải nghiệm thoại tốt ²), cũng như đạt **tốc độ khung hình ~15-30 FPS** cho camera. Hệ thống sử dụng triệt để sức mạnh của điện thoại (NPU/GPU) để chạy mô hình ML nhẹ trong thời gian thực, và chỉ dùng Arduino cho điều khiển phần cứng.

Tính năng MVP và User Stories

Bảng dưới đây liệt kê các **tính năng chính của MVP** cùng mô tả ngắn gọn và một số **user story** minh họa cách người dùng tương tác:

- **Nghe & Đáp (Voice Interaction):** Hỗ trợ hội thoại cơ bản bằng tiếng Việt. Người dùng nhấn **PTT** và nói, câu nói sẽ được nhận dạng giọng nói ngay trên máy (Android Speech API hoặc Vosk). Hệ thống phân tích ý định và phản hồi bằng giọng nói qua TTS. **Thời gian phản hồi** được tối ưu để trong vòng ~0.7 giây robot bắt đầu nói ², tạo cảm giác gần như tức thì.
User Story: Là một người dùng, tôi muốn chào robot “Xin chào” và nghe nó chào lại vui vẻ ngay lập tức. (Robot phản hồi: “Xin chào, rất vui được gặp bạn!” kèm LED nhấp nháy màu thân thiện).
- **Nhìn & Theo dõi (Vision & Tracking):** Robot nhận biết sự hiện diện của con người và vật thể qua camera. Khi phát hiện **người** hoặc **khuôn mặt**, robot có thể tự động xoay **servo camera** hướng về người gần nhất như đang “nhìn theo” (tín hiệu servo được tính toán từ tọa độ khuôn mặt phát hiện). Các **vật thể phổ biến** (như chai nước, bóng, gấu bông) cũng được nhận diện bằng mô hình TFLite (MobileNet SSD đã huấn luyện trên COCO) chạy on-device ⁴, cho phép phân loại khoảng 80 loại vật thể phổ thông. Giao diện sẽ vẽ khung chữ nhật quanh mặt người/vật thể cùng nhãn (nếu nhận dạng được).
User Story: Là người dùng, tôi di chuyển trong phòng và thấy robot camera luôn xoay theo hướng tôi. Khi tôi hỏi “Bạn đang thấy gì?”, robot sẽ mô tả: “Mình đang nhìn thấy 1 người ở phía trước, 1 chai nước bên phải...” (thông tin dựa trên những gì camera nhận dạng được).

- **Học có Giám sát (Teach & Remember):** Chế độ cho phép người dùng “dạy” robot nhận biết khuôn mặt cá nhân hoặc đồ vật cụ thể và lưu vào bộ nhớ. Kích hoạt qua câu lệnh thoại (hoặc nút Teach): Robot sẽ lấy một loạt khung hình của mục tiêu, tính **embedding vector** đặc trưng (128-D hoặc 512-D đối với khuôn mặt ⁷, hoặc đặc trưng ảnh cho vật thể) và lưu vào **Memory Store** cùng với **nhãn** do người dùng đặt. Mỗi lần chỉ học **một khuôn mặt hoặc một vật** để đảm bảo tập trung và chính xác. Sau khi học, robot sẽ nhớ lâu dài: có API để *truy vấn* (“Ai đây?” để hỏi tên người trước mặt, “Cái này là gì?” để hỏi tên vật thể đang cầm trước camera) – robot tìm vector gần nhất trong **Memory DB** để trả lời ⁵. Cũng có lệnh *quên* (“Quên <tên>”) để xóa mục nhớ. Chính sách độ tin cậy: nếu độ tương đồng thấp hơn ngưỡng khi nhận diện, robot sẽ hỏi lại “Có phải <tên> không?” để xác nhận, tránh nhầm lẫn.

User Story: Một phụ huynh chỉ camera vào mặt mình và nói: “Nhớ mặt mình là Ba”. Robot chụp khuôn mặt người đó vài lần, lưu đặc trưng và nhãn “Ba”. Ngày hôm sau, khi người đó xuất hiện và hỏi “Ai đây?”, robot nhận ra khuôn mặt và đáp: “Đây là Ba”. Tương tự, phụ huynh có thể cầm một món đồ và nói “Dạy đồ vật này là bình sữa”, rồi sau đó hỏi “Cái này là gì?” – robot sẽ trả lời “Đó là bình sữa” khi nhận ra vật đã học.

- **Mô tả Cảnh & Ngữ cảnh:** Robot có thể tóm tắt những gì nó “thấy” qua camera theo yêu cầu. Lệnh “Bạn đang thấy gì?” (intent `describe_scene`) sẽ kích hoạt Behavior Planner truy vấn kết quả nhận dạng gần nhất từ Vision (danh sách người & vật thể cùng số lượng), sau đó tạo câu mô tả. Để giảm độ trễ và tránh nội dung vượt khả năng, từ vựng mô tả trong MVP có thể giới hạn ở các danh mục phổ biến (người, trẻ em, đồ chơi, chai lọ, thú bông...).

User Story: Người dùng hỏi “Bạn thấy gì bây giờ?”, robot đáp: “Mình thấy 2 người và 1 quả bóng phía trước”. (cảnh trước camera có 2 người và 1 quả bóng, được nhận ra qua model). Nếu camera không thấy ai/vật gì quen thuộc: “Mình thấy không có ai và đồ vật lạ xung quanh.”

- **Điều khiển Thân Robot:** Người dùng ra lệnh di chuyển: “Đi tới” (tiến thẳng), “Lùi lại”, “Quay trái/phải” – tương ứng intent `move(direction)` – Behavior Planner sẽ gửi lệnh **DRIVE** với tốc độ bánh xe trái/phải qua Bluetooth. Lệnh “Đừng lại” lập tức gửi tốc độ 0 (STOP). Ngoài ra, lệnh “Nhìn lên/xuống/trung tâm” điều khiển góc camera (servo) – mapping sang gói **SERVO (id, pos)** tương ứng. Lệnh “Đổi đèn màu đỏ/xanh...” map sang gói **LED (r,g,b)**. Tất cả lệnh được truyền với tần số cao (20-50Hz) để robot phản ứng gần như ngay (<50ms) kể từ lúc người dùng nói ². Hệ thống cũng có cờ an toàn: nếu quá 300ms không nhận lệnh mới, Arduino sẽ tự động dừng động cơ (**watchdog**).

User Story: Một đứa trẻ nói “Đi tới!” – robot lập tức tiến về trước vài bước. Đứa trẻ tiếp: “Đừng lại... quay trái!” – robot ngừng lại rồi xoay tại chỗ sang trái. Sau đó, đứa trẻ nói “Nhìn theo mình” rồi di chuyển – robot xoay đầu (servo camera) để giữ người đó trong khung hình. Cuối cùng đứa trẻ bảo “Nháy đèn xanh đi” – đèn LED trên robot chuyển sang màu xanh da trời nhấp nháy.

Các tính năng trên được xây dựng với mục tiêu **MVP tối giản nhưng đủ hữu ích**: tập trung vào tương tác trực quan (giọng nói + thị giác) và điều khiển thời gian thực, tránh mở rộng quá rộng ngay từ đầu.

Giao thức Bluetooth SPP (Android ↔ Arduino)

Để đảm bảo **truyền thông thời gian thực ổn định** giữa ứng dụng Android và vi điều khiển Arduino, ta thiết kế một **giao thức khung nhị phân** đơn giản chạy trên **Bluetooth SPP (Serial Port Profile)**. Ưu điểm: giao thức nhị phân có độ trễ thấp, khối lượng dữ liệu nhỏ gọn, và dễ parse ở cả hai phía.

Cấu trúc khung gói: Mỗi gói tin có cấu trúc cố định gồm header, trường loại, độ dài, payload và checksum:

[0xAA] [0x55] [type] [seq] [len] [... payload bytes ...] [xor_checksum]

• **Header 0xAA55:** Hai byte cố định đánh dấu bắt đầu khung (giúp Arduino/Android nhận biết khi nào một gói mới bắt đầu trong dòng byte liên tục; sử dụng cặp byte đặc biệt 0xAA 0x55 – phổ biến trong các giao thức nhúng để đánh dấu khung⁸).

• **type:** 1 byte xác định *loại lệnh/dữ liệu* của gói. Các mã loại dự kiến:

- 0x01 – **DRIVE:** lệnh điều khiển động cơ (vd. robot di chuyển).
- 0x02 – **SERVO:** lệnh điều khiển servo (vd. góc camera).
- 0x03 – **LED:** lệnh điều khiển đèn LED.
- 0x10 – **SENSOR_REQ:** yêu cầu Arduino gửi dữ liệu cảm biến.
- 0x11 – **SENSOR_DATA:** gói Arduino trả dữ liệu cảm biến (khoảng cách, IMU, nhiệt độ...).
- 0xFE – **PING:** gói tín hiệu kiểm tra kết nối (Android gửi).
- 0xFF – **ACK:** gói phản hồi xác nhận đã nhận (Arduino gửi).
- 0xE0 – **ERROR:** gói báo lỗi (có thể dùng khi checksum sai, lệnh không hợp lệ,...).

• **seq:** 1 byte *sequence number* (0–255) để đánh số thứ tự gói, hỗ trợ theo dõi và phát hiện mất gói. Mỗi lần gửi tăng seq, phía nhận phản hồi ACK kèm cùng seq.

• **len:** 1 byte *number of payload bytes* theo sau. Các gói cố định độ dài cũng vẫn điền giá trị này để dễ kiểm tra.

• **payload:** dữ liệu nội dung gói, độ dài tùy theo type.

• **xor_checksum:** 1 byte checksum tính bằng XOR từ tất cả bytes từ header 0xAA đến cuối payload.

Phía nhận cũng XOR các byte nhận được (trừ checksum) để kiểm tra khớp giá trị checksum đi kèm – nếu lệch thì bỏ gói (hoặc gửi gói ERROR yêu cầu gửi lại).

Định dạng payload cho từng loại chính: - **DRIVE (0x01):** 2 byte – gồm **vL** và **vR** (kiểu *int8_t*, giá trị -100...100) biểu diễn tốc độ bánh trái và phải (% hoặc đơn vị tuỳ hiệu chỉnh). Ví dụ: **vL=50, vR=50** robot sẽ chạy tới nhanh vừa; **vL=30, vR=-30** robot quay tại chỗ sang phải. - **SERVO (0x02):** 3 byte – gồm **id** (*uint8*) chỉ định servo nào (nếu robot có nhiều servo) và **pos** (*uint16*) là giá trị xung (microseconds) cho góc servo đó (500–2500 µs tương ứng góc giới hạn servo, ví dụ 1500 là góc giữa). - **LED (0x03):** 3 byte – **r, g, b** (mỗi byte 0–255) biểu thị giá trị màu RGB cho đèn LED. Robot có thể chỉ có 1 LED RGB, giá trị này đặt màu trực tiếp. - **SENSOR_DATA (0x11):** Dữ liệu từ cảm biến robot gửi lên. Dự kiến 12 byte: - 2 byte **dist_mm** (*uint16*) – khoảng cách đo (vd cảm biến siêu âm hoặc LiDAR, đơn vị mm). - 6 byte IMU gồm **ax, ay, az, gx, gy, gz** (mỗi cái *int8* hoặc *int16* tùy độ chính xác; ở đây có thể dùng *int16* cho gia tốc và gyroscope). - 2 byte **t_c10** (*int16*) – nhiệt độ (nhân 10, ví dụ 253 tức 25.3°C). - 2 byte **h_p10** (*uint16*) – độ ẩm (% nhân 10).

(Lưu ý: Cấu trúc trên có thể điều chỉnh theo cảm biến thực tế; mục tiêu chính là minh họa khung dữ liệu.)

Ví dụ gói lệnh: Giả sử robot nhận lệnh đi với tốc độ 50%, gói DRIVE sẽ là: AA 55 01 3A 02 32 32 61. Giải: 0xAA55 header, type=0x01, seq=0x3A, len=0x02, payload "0x32 0x32" (50,50 thập phân), checksum=0x61 (XOR của AA^55^01^3A^02^32^32). Arduino khi nhận sẽ XOR kiểm tra 0x61 khớp, sau đó giải mã payload thành **vL=50, vR=50**.

Tần số và Cơ chế an toàn: Ứng dụng Android gửi các gói điều khiển **liên tục 20–50 lần/giây** (tùy tốc độ cập nhật cần thiết) để đảm bảo robot di chuyển mượt và có thể dừng nhanh khi cần. Đồng thời, Android định kỳ gửi **PING (0xFE)** mỗi 100ms; Arduino khi nhận sẽ lập tức đáp **ACK (0xFF)**. Nếu quá 300ms không thấy bất kỳ gói nào từ Android (cả lệnh lẫn ping), Arduino sẽ hiểu kết nối gián đoạn và **tự động ngừng động cơ** (đảm bảo an toàn, tránh trường hợp robot tiếp tục chạy mất kiểm soát). Giao thức đơn giản

này sử dụng checksum XOR thay vì CRC để tiết kiệm tính toán, và header cố định giúp Arduino dễ dàng bắt đầu đọc đúng điểm (trường hợp byte dữ liệu trùng pattern header rất hiếm, có thể bổ sung escaping nếu cần thiết để tránh hiểu nhầm 0xAA trong payload ⁹). Với băng thông SPP (~<100 KB/s), giao thức này thừa khả năng truyền lệnh/telemetry ở tần số cao, độ trễ truyền ước tính <20ms cho mỗi gói, đáp ứng tốt yêu cầu điều khiển realtime.

Lựa chọn Công nghệ & Tiêu chí Kỹ thuật

Nền tảng Android (ngôn ngữ Kotlin): Sử dụng **Kotlin** với Android SDK hiện đại cho tốc độ phát triển nhanh và tích hợp tốt với các thư viện Jetpack (Compose, CameraX, etc.). Kiến trúc app theo mô hình **MVVM + Clean Architecture**, tách biệt rõ business logic (core) và UI (feature), giúp dễ mở rộng về sau ¹⁰. Dùng **Hilt (Dagger)** để quản lý dependency injection cho các service (ASR, TTS, Vision) – cho phép thay thế dễ dàng (ví dụ chuyển từ Android Speech API sang Vosk mà không ảnh hưởng code sử dụng).

Giao diện Compose & CameraX: Giao diện dùng **Jetpack Compose** để xây dựng nhanh UI khai báo, đồng thời tận dụng Compose Preview và tính linh hoạt cho UI động (vd. hiển thị bounding box nhận dạng). Camera truy xuất qua **CameraX API** ở chế độ preview + analyze. CameraX cho phép lấy từng khung hình dưới dạng **ImageProxy** – ta sẽ chuyển vào pipeline ML để phân tích (phát hiện khuôn mặt/vật thể) ¹¹. Compose không trực tiếp render camera, thay vào đó ta dùng **PreviewView** kết hợp với **AndroidView** trong Compose, hoặc thư viện Accompanist nếu cần, để hiển thị feed camera nền. Các **bounding box** và nhãn được vẽ bằng Compose Canvas trên overlay. (Thực tế có mẫu app minh họa kết hợp Compose + TFLite SSD cho nhận diện real-time khá tốt ¹²).

Xử lý Giọng nói (ASR): MVP dùng **Android SpeechRecognizer API** (hoặc **RecognizerIntent**) để nhận dạng tiếng Việt offline. Lý do: dễ tích hợp, hỗ trợ tiếng Việt tốt, và có chế độ offline (nếu cài đặt gói ngôn ngữ) với độ chính xác chấp nhận được. Độ trễ nhận dạng thường <500ms cho câu ngắn (nhờ chạy cục bộ) – phù hợp yêu cầu <700ms cho phản hồi nhanh ². Hơn nữa, API này tích hợp sẵn *Voice Activity Detection*, có thể giúp phát hiện khi người dùng ngừng nói để trả kết quả sớm (trong PTT thì người dùng sẽ thả nút khi nói xong). Mặc dù vậy, thiết kế vẫn dự trù interface **ASRProvider** trừu tượng; sau này có thể **thay engine** dễ dàng (như thư viện **Vosk** hoặc **Whisper.cpp** cho nhận dạng hoàn toàn offline) chỉ bằng cách viết một lớp **VoskAsrImpl** thực thi interface này và cung cấp qua DI.

Xử lý Ngôn ngữ Tự nhiên (NLU): Trước mắt triển khai đơn giản **rule-based Intent Router** – sử dụng so khớp từ khóa và cú pháp mẫu để xác định intent (do phạm vi lệnh hẹp, rule-based sẽ đủ hiệu quả ở MVP). Ví dụ: câu chứa “xin chào” -> greet, chứa “thấy gì” -> describe_scene, chứa “đi tới/trái/phải/lùi” -> move, chứa “nhớ mặt... là X” hoặc “dạy khuôn mặt” -> teach(face), chứa “đồ vật ... là X” -> teach(object), chứa “ai đây” -> recall(face), “cái này là gì” -> recall(object), “quên X” -> forget, v.v. Mặc dù dùng luật cứng, module IntentRouter vẫn tách riêng để có thể thay bằng NLU học máy sau này (VD: dùng TensorFlow Lite model hoặc thậm chí gọi API GPT khi online). **Dialogue Manager** thì quản lý trạng thái cho các phiên *teach* hay *confirmation*, cũng như cờ cho phép *barge-in*.

Thị giác Máy tính (Vision): Sử dụng **CameraX** kết hợp **ML Kit** hoặc model TFLite tự có để xử lý khung hình. Cụ thể, **ML Kit Face Detection** (on-device) để lấy vị trí khuôn mặt và các điểm đặc trưng cơ bản (ML Kit trả về thông tin pose, eyes open, v.v., nhưng quan trọng nhất là bounding box và *tracking ID* liên tục giữa các frame ¹³). Face Detection chạy rất nhanh trên thiết bị hiện đại (có thể ~30FPS, nhưng ta xử lý ~15FPS để tiết kiệm pin). Với **object detection**, sử dụng mô hình **SSD MobileNet** đã được TensorFlow tối ưu cho thiết bị di động ⁴ – có thể dùng phiên bản TFLite 300x300 COCO để nhận biết ~80 vật thể phổ biến; tốc độ trên GPU điện thoại đạt ~15-20FPS. Thư viện **TensorFlow Lite** sẽ được tích hợp để chạy model này. Đối với **nhận dạng khuôn mặt** (face recognition – gán tên cho khuôn mặt đã gấp), ML Kit không hỗ trợ sẵn nên ta tích hợp một model nhúng như **FaceNet** hoặc **MobileFaceNet** TFLite để trích

xuất **vector embedding** 128 chiều cho mỗi khuôn mặt ⁷. Khi user kích hoạt “nhớ mặt X”, hệ thống sẽ phát hiện khuôn mặt bằng ML Kit, cắt ảnh khuôn mặt, đưa qua model FaceNet TFLite để lấy embedding, rồi lưu embedding đó vào database cùng label X. Lúc nhận lệnh “Ai đây?”, hệ thống tính embedding khuôn mặt hiện tại và so sánh với các embedding đã lưu (dùng khoảng cách cosine hoặc L2); nếu gần nhất dưới ngưỡng cho trước -> nhận diện đó là X ⁵. Cách này cho phép nhận diện on-device trong thời gian thực và đã được chứng minh bởi nhiều ứng dụng mẫu ^{5 11}. Tương tự cho vật thể: nếu dạy vật thể mới, ta có thể lưu **feature vector** trích từ một lớp trước softmax của model nhận dạng (transfer learning) để so sánh về sau.

TTS (Text-to-Speech): Sử dụng **Android TTS (TextToSpeech)** engine mặc định của hệ thống (ví dụ Samsung TTS hoặc Google TTS) để phát tiếng Việt. Ưu điểm: cài sẵn, giọng tương đối tự nhiên và phản hồi nhanh (khởi tạo TTS trước để tránh trễ khi nói lần đầu). Gói thiết kế **TtsProvider** trừu tượng; nếu sau này muốn dùng giọng cloud (Google, AWS Polly, v.v.) thì chỉ cần triển khai **CloudTtsImpl** gọi API và thay thế qua DI. Thời gian bắt đầu phát TTS sau khi có text thường <200ms. Ngoài ra, implement chính sách **barge-in**: khi ASR phát hiện người dùng nói (hoặc PTT được nhấn) thì TtsProvider sẽ *ngắt* ngay câu nói hiện tại (dùng **stop()** hoặc quản lý luồng audio).

Memory & Lưu trữ: Sử dụng **Room** (SQLite ORM của Android) để lưu dữ liệu khuôn mặt/vật thể đã học. Tạo một entity **MemoryItem** với các field: **id** (khóa chính), **type** (FACE/OBJECT), **label** (tên do người dùng đặt), **embedding** (mảng float - có thể lưu dưới dạng BLOB), **metadata** (JSON hoặc các cột: ví dụ **taughtAt**, **lastSeenAt**, **timesSeen** ...), **imageRef** (tùy chọn: URI ảnh mẫu). Việc lưu embedding vector trong SQLite BLOB là khả thi do kích thước nhỏ (128 float ~ 512 byte). Ngoài ra có thể dùng **DataStore (Proto)** để lưu config hoặc các thông số (như ngưỡng nhận dạng, danh sách từ khóa). Tất cả dữ liệu sẽ nằm trong bộ nhớ cục bộ của app; nếu cần backup hay đồng bộ cloud có thể cân nhắc sau nhưng Phase 1 không làm.

Tiêu chí hiệu năng: Đặt mục tiêu các chỉ số chính: - **Độ trễ hội thoại:** Thời gian từ khi người dùng **bắt đầu nói** đến khi robot **bắt đầu phản hồi âm thanh** $\leq 700\text{ms}$. Trong đó ASR xử lý ~200-500ms tùy độ dài câu; NLU/Planning gần như tức thì; TTS bắt đầu nói sau ~100-200ms chuẩn bị. Mục tiêu $<700\text{ms}$ nằm trong ngưỡng trải nghiệm người dùng chấp nhận được, gần mức hội thoại tự nhiên ². - **Tốc độ khung hình camera:** $\geq 15 \text{ FPS}$ với pipeline nhận dạng chạy song song (camera 30fps, nhưng ML phân tích mỗi 2 khung để tiết kiệm). 15fps đủ để theo dõi người di chuyển mượt và cập nhật thông tin cảnh liên tục. - **Độ trễ điều khiển:** Lệnh di chuyển/servo từ app đến robot $<50\text{ms}$. Bluetooth SPP thường <20ms, Arduino xử lý tức thì, động cơ có quán tính nhưng phản ứng lệnh điện hầu như ngay lập tức. Cảm giác người dùng sẽ gần như đồng bộ giữa giọng nói và hành động robot. - **Độ ổn định kết nối:** Bluetooth phải duy trì ổn định trong phạm vi ~5-10m. Sử dụng cơ chế ping/ack để phát hiện nhanh mất kết nối. Nếu mất kết nối, app giao diện sẽ thông báo và cố gắng kết nối lại tự động, đồng thời Arduino dừng an toàn. - **An toàn & Riêng tư:** Ứng dụng **không ghi lại** âm thanh thô hoặc hình ảnh thô vào bộ nhớ. Chỉ lưu thông tin đã xử lý (như embedding, nhấn). Quyền truy cập **Camera/Mic** được thông báo rõ ràng và chỉ bật khi cần (Mic chỉ nghe khi PTT, Camera chỉ hoạt động trong màn hình chính của app). Có thể tích hợp **Kid Mode**: chặn một số lệnh nguy hiểm (như “đi nhanh”, “đâm vào tường” – nếu có) hoặc giới hạn vùng hoạt động, nhằm đảm bảo trẻ em sử dụng an toàn. Ngoài ra, robot sẽ *không kết nối Internet* ở phase này, loại bỏ nguy cơ rò rỉ dữ liệu.

Cấu trúc Mã nguồn & Module hóa

Dự án Android sẽ được tổ chức thành các module rõ ràng để dễ phát triển độc lập và tái sử dụng. Cấu trúc thư mục (theo gợi ý Clean Architecture và phân tầng core/feature) như sau:

```

app/
  |- core/                      # Layer core logic, independent of UI
    |- asr/                      # Xử lý tiếng nói
      |- ASRProvider.kt          (interface chung cho ASR)
      |- GoogleAsrImpl.kt        (triển khai dùng SpeechRecognizer)
      |- VoskImpl.kt              (triển khai dùng Vosk offline, optional)
    |- vision/                  # Xử lý tầm nhìn máy tính
      |- CameraController.kt     (quản lý CameraX)
      |- FaceTracker.kt          (theo dõi khuôn mặt, dùng ML Kit)
      |- ObjectDetector.kt       (nhận diện vật thể TFLite)
      |- TeachFlow.kt             (quy trình thu thập dữ liệu khi dạy)
    |- nlu/                      # Xử lý ngôn ngữ & hội thoại
      |- IntentRouter.kt          (rule-based parser → Intent)
      |- DialogueManager.kt       (quản lý trạng thái hội thoại, barge-in)
      |- IntentModels.kt          (định nghĩa các intent, dữ liệu phụ)
    |- memory/                  # Lưu trữ bộ nhớ đối tượng/face
      |- MemoryStore.kt           (interface quản lý memory chung)
      |- FaceDao.kt, ObjectDao.kt (DAO Room cho bảng face/object)
      |- MemoryDatabase.kt         (Room database config)
      |- MemoryModels.kt           (định nghĩa entity: FaceEntity,
ObjectEntity,...)
      |- behavior/                # Hoạch định hành vi & điều khiển
        |- BehaviorPlanner.kt     (logic chọn action dựa trên intent &
context)
        |- PolicyRules.kt          (các rule cho mapping intent->hành vi)
        |- Expressions.kt           (kịch bản câu thoại phản hồi mẫu, hiệu ứng
LED,...)
      |- tts/                      # Xử lý tổng hợp giọng nói
        |- TtsProvider.kt           (interface chung TTS)
        |- AndroidTtsImpl.kt        (triển khai dùng TextToSpeech có sẵn)
      |- device/                  # Giao tiếp thiết bị phần cứng
        |- BtClient.kt              (quản lý kết nối Bluetooth SPP)
        |- PacketCodec.kt            (mã hóa/giải mã gói theo giao thức)
        |- CommandBus.kt             (quản lý hàng đợi lệnh gửi đi ở tần số cố
định)
        |- TelemetryRepo.kt          (xử lý dữ liệu cảm biến nhận về, lưu log)
  |- feature/companion/          # Layer UI + trình bày cho Companion
    |- ui/                       (Compose UI màn hình)
      |- ConnectScreen.kt          (UI ghép nối Bluetooth, chọn thiết bị)
      |- CompanionScreen.kt        (UI chính: camera preview, nút PTT, nút
Teach,...)
      |- components/              (các Composable nhỏ: ví dụ mic button,
overlay box,...)
      |- vm/                      (ViewModel cho màn hình, quản lý state và
luồng data)
        |- ConnectViewModel.kt
        |- CompanionViewModel.kt
  |- di/                        # Module DI Hilt
    |- AppModule.kt               (cung cấp singletons: ASRProvider,

```

```
TtsProvider, v.v.)  
└ ... (có thể thêm module khác cho Vision, etc.)
```

Trong thiết kế này, **mỗi chức năng chính** (ASR, Vision, NLU, Memory, Behavior, TTS, Device) đều có module riêng trong `core/`. Mỗi module định nghĩa **interface** để trừu tượng hóa chức năng (ví dụ `ASRProvider`, `TtsProvider`, `MemoryStore`...) và các implementation cụ thể (như `GoogleAsrImpl`). Điều này hỗ trợ *thay thế dễ dàng*: ví dụ chuyển từ `SpeechRecognizer` sang dịch vụ cloud chỉ cần viết lớp mới mà không phải sửa chỗ khác. Kiến trúc này tuân thủ nguyên tắc **Dependency Inversion** (UI phụ thuộc vào abstraction của core, core không phụ thuộc UI). UI layer (`feature/companion`) chỉ tương tác với `ViewModel`, bên trong `ViewModel` dùng các **UseCase/Service** từ core (có thể cung cấp qua Hilt).

Để rõ hơn, sau đây là **ví dụ một vài interface quan trọng** trong core layer:

```
// ASRProvider - interface cho module nhận dạng giọng nói  
interface ASRProvider {  
    fun startListening(onResult: (String) -> Unit, onError: (Exception) ->  
        Unit)  
    fun stopListening()  
}  
// Triển khai Google ASR sử dụng SpeechRecognizer API  
class GoogleAsrImpl(context: Context): ASRProvider { ... }  
  
// TtsProvider - interface cho module tổng hợp giọng nói  
interface TtsProvider {  
    fun speak(text: String, interruptIfSpeaking: Boolean = true)  
    fun stopSpeaking()  
}  
// Triển khai Android TTS  
class AndroidTtsImpl(context: Context): TtsProvider { ... }  
  
// FaceTracker - theo dõi khuôn mặt qua camera  
interface FaceTracker {  
    val facesFlow: Flow<List<DetectedFace>> // luồng output các khuôn mặt  
    phát hiện  
    fun startTracking(cameraProvider: ProcessCameraProvider)  
    fun stopTracking()  
}  
  
// BtClient - quản lý kết nối Bluetooth SPP  
interface BtClient {  
    suspend fun connect(device: BluetoothDevice): Boolean  
    fun sendPacket(packet: ByteArray)  
    fun readPackets(): Flow<ByteArray> // luồng gói nhận  
    fun disconnect()  
}
```

(Ghi chú: Mã trên chỉ minh họa phương thức chính yếu, chi tiết triển khai nằm trong các lớp `Impl` tương ứng.
Các interface khác như `MemoryStore` hay `BehaviorPlanner` cũng sẽ được định nghĩa tương tự.)

Cấu trúc mã như trên giúp nhóm lập trình có thể phân chia công việc *theo module*: một người có thể tập trung xây dựng **Vision** (xử lý CameraX, ML), người khác lo **ASR/TTS**, người khác lo **Bluetooth & Device**, v.v. Các module tương tác qua interface rõ ràng, giảm thiểu phụ thuộc lẫn nhau. Việc **test** cũng thuận lợi hơn: có thể mock các provider (ví dụ mock ASR để test logic DialogueManager). Kiến trúc sử dụng triệt để lập trình bất đồng bộ (suspend function, Flow) để tránh chặn UI thread: ví dụ Camera frames xử lý trên Dispatchers.Default, ASR chạy trong service riêng, Bluetooth I/O trên thread IO... Compose UI hầu như chỉ collect Flow từ ViewModel và vẽ trạng thái.

Schema Bộ nhớ & Vòng đời Học/Quên

Thiết kế cơ sở dữ liệu MemoryStore: Sử dụng **Room** tạo database lưu thông tin các khuôn mặt và đồ vật đã học. Ta có thể dùng một bảng chung cho mọi MemoryItem, hoặc tách 2 bảng Face và Object. Ở đây chọn bảng chung đơn giản với kiểu phân biệt.

Bảng `memory_items` (nếu dùng Room sẽ là một data class `MemoryItemEntity`):

- `id` (Primary Key, auto-generate) – ID duy nhất cho mỗi mục nhớ.
- `type` (Integer hoặc String) – loại mục: "FACE" hoặc "OBJECT" (hoặc enum mapping).
- `label` (Text) – nhãn do người dùng đặt, ví dụ "Ba", "bình sữa".
- `embedding` (BLOB) – mảng nhị phân chứa vector đặc trưng nhận dạng. Với khuôn mặt, vector 128 float (512 byte) hoặc 512 float; với vật thể, có thể dùng vector đặc trưng 32-128 float từ model nhận dạng.
- `metadata` (Text hoặc cột phụ) – thông tin phụ JSON: ví dụ `source` (loại nguồn: face hay object, có thể trùng type), `taughtAt` (timestamp lúc học), `lastSeenAt` (timestamp lần cuối nhận ra), `timesSeen` (đếm số lần đã nhận ra), `confidence` (độ tự tin lần nhận ra gần nhất)...
- `imagePath` (Text, optional) – đường dẫn tới hình ảnh mẫu (nếu lưu trong storage để debug, không bắt buộc).
- `reserved1, reserved2` – các trường dự phòng (nếu muốn lưu thêm thông tin sau này như vector trung bình nếu có nhiều mẫu per label...).

Lifecycle "Học - Ghi nhớ - Quên": 1. *Teach (Học):* Khi người dùng kích hoạt dạy, ví dụ "Dạy khuôn mặt này là Ba": ứng dụng sẽ vào **TeachFlow** – bật camera (nếu chưa), lấy liên tục một vài ảnh khuôn mặt trước camera (có thể hướng dẫn người dạy nhìn thẳng). Với mỗi ảnh, chạy FaceNet model để trích xuất embedding. Có thể lấy trung bình hoặc chọn embedding tốt nhất (rõ nét) làm đại diện. Tạo một bản ghi MemoryItem với `type=FACE`, `label="Ba"`, lưu embedding và metadata (ví dụ thời gian, có thể lưu luôn embedding gốc khác nếu cần). Quá trình này nên thực hiện nhanh (1-2 giây) để không làm người dùng chờ lâu. Tương tự cho vật thể: chụp vài ảnh vật thể từ camera, có thể dùng chính model nhận dạng vật thể để lấy đặc trưng (ví dụ đầu ra lớp áp chót) rồi lưu. Sau khi lưu xong, TTS có thể phản hồi: "Đã nhớ <label>." để xác nhận.

2. *Recall (Nhớ lại):* Khi cần nhận dạng, chẳng hạn user hỏi "Ai đây?" trong khi một người đứng trước camera: hệ thống lấy khuôn mặt phát hiện lớn nhất, tính embedding hiện thời, rồi so sánh với tất cả embedding trong DB loại FACE. Có thể dùng khoảng cách cosine; tìm mục có khoảng cách nhỏ nhất. Nếu khoảng cách < threshold (ví dụ 0.5) thì xem như *match*: giả sử match với label "Ba" độ tương đồng cao, robot TTS đáp "Đây là Ba." ⁵. Nếu không mục nào dưới ngưỡng, robot đáp "Mình không biết người này" hoặc "Bạn có thể dạy mình người này tên là gì không?". Tương tự với vật thể: "Cái này là gì?" → tìm trong DB OBJECT. Trường hợp có nhiều mục gần giống, có thể ưu tiên mục có khoảng cách nhỏ nhất nhưng nếu hai khoảng cách gần nhau, robot có thể hỏi lại để chắc chắn.

3. *Forget (Quên):* Khi người dùng ra lệnh quên một mục ("Quên Ba"), hệ thống sẽ tìm trong DB mục có label khớp (và type tương ứng, nếu biết). Nếu tìm thấy, xóa bản ghi đó khỏi DB. Xử lý này cần xác nhận: có thể robot hỏi lại "Bạn có chắc muốn quên Ba không?" – nếu user xác nhận "đồng ý" thì mới xóa. Sau khi xóa, robot sẽ không nhận ra đối tượng đó nữa cho đến khi được dạy lại.

Quản lý độ tin cậy & cập nhật: Mỗi khi robot nhận dạng đúng một mục đã nhớ, nó có thể cập nhật `lastSeenAt` và tăng `timesSeen`. Nếu lần nhận không chắc chắn (gần ngưỡng), robot có thể tạm *không tự động chào tên* mà thay vào đó hỏi người dùng xác nhận. Nếu người dùng xác nhận đúng, có thể cập nhật embedding: ví dụ tính trung bình với embedding mới để tăng độ chính xác (nhưng MVP có thể chưa làm để đơn giản). Ngoài ra, MemoryStore có thể cài giới hạn số mục nhớ (vd. tối đa 50 mục) để tránh đầy bộ nhớ – nếu đạt giới hạn, có thể thông báo không nhận thêm hoặc yêu cầu người dùng xóa bớt.

Skeleton Intents & Hành vi (MVP)

Để hệ thống hoạt động theo yêu cầu, ta xác định một tập *intent* chính và ánh xạ chúng tới các hành động cụ thể của robot. Dưới đây là bảng tóm tắt **các intent MVP** và xử lý tương ứng (trong Behavior Planner):

- `greet()` - **Chào hỏi:** Kích hoạt khi người dùng chào robot ("Xin chào", "Hello robot"...). *Hành vi:* Robot đáp lại bằng lời chào thân thiện qua TTS (ví dụ: "Xin chào, tôi có thể giúp gì?") và có thể nháy LED màu xanh lá cây vài lần (thể hiện "vui vẻ"). Nếu có camera nhìn thấy người, có thể hướng mặt về phía người đó khi chào.
- `describe_scene()` - **Mô tả cảnh:** Kích hoạt khi user hỏi robot thấy gì. *Hành vi:* Behavior Planner lấy danh sách các đối tượng phát hiện được gần nhất từ Vision (ví dụ: 1 người, 2 vật thể – loại gì nếu biết tên hoặc danh mục). Xâu chuỗi thành câu mô tả ngắn rồi gọi TTS nói ra. Ví dụ: nếu memory có tên người quen thuộc trong khung hình, có thể nói "Có anh Nam ở trước mặt, và 1 cái ghế bên phải."; nếu không biết ai: "Mình đang thấy 1 người lạ và 1 quả bóng."
- `teach(name, target)` - **Học ghi nhớ:** Kích hoạt khi user dùng câu "Dạy... là ...". Có hai biến thế: `target=face` (dạy khuôn mặt) hoặc `target=object` (dạy đồ vật), và `name` là nhãn. *Hành vi:* DialogueManager chuyển trạng thái sang `TEACH_MODE` để thực hiện quy trình TeachFlow (như mô tả ở phần trước). Trong trạng thái này, robot có thể hướng dẫn user nếu cần: "Đang học, hãy giữ mục tiêu trước camera". Sau khi thu thập và lưu memory, thoát chế độ dạy, TTS xác nhận "Đã nhớ <name>.".
- `recall(target)` - **Nhớ lại:** Kích hoạt khi user hỏi về đối tượng trước mặt. `target` có thể là `face` (ví dụ "Ai đây?") hoặc `object` ("Cái này là gì?"). *Hành vi:* Behavior Planner truy vấn MemoryStore: nếu `target=face`, lấy khuôn mặt hiện tại -> tìm label; nếu `target=object`, lấy vật thể lớn nhất đang focus -> tìm label. Sau đó TTS trả lời: nếu tìm thấy label => "Đây là <name>.", nếu không => "Mình chưa biết <đối tượng> này." hoặc gợi ý học.
- `forget(name)` - **Quên:** Kích hoạt khi user yêu cầu xóa một mục đã học ("Quên <Tên>"). *Hành vi:* Robot sẽ kiểm tra có memory nào khớp label đó không. Nếu có, DialogueManager có thể yêu cầu xác nhận: "Bạn có chắc muốn quên <name> không?". Nếu user xác nhận (có) -> xóa khỏi DB, TTS phản hồi "Đã quên <name>.". Nếu user huỷ -> không xóa, TTS: "OK, vẫn giữ <name> trong bộ nhớ".
- `move(direction)` - **Điều khiển di chuyển:** Kích hoạt khi user ra lệnh di chuyển. `direction` có thể là các hướng: tiến ("đi tới"), lùi, trái, phải, hoặc dừng. *Hành vi:* Behavior Planner ánh xạ:
- `forward / backward`: gửi gói DRIVE với $vL=vR=+V$ hoặc $-V$ (tốc độ phù hợp, V có thể 50/100%).
- `left / right`: có hai cách, nếu robot có bánh xe vi sai: quay tại chỗ ($vL=-vR$), nếu robot kiểu xe hơi: có thể dùng servo góc lái (nhưng ở đây giả định xe tank/vi sai).
- `stop`: gửi $vL=0, vR=0$.
- Ngoài ra, lệnh "nhanh hơn / chậm lại" có thể điều chỉnh V . (MVP có thể chưa cần).
- Không phản hồi TTS cho lệnh di chuyển, để tránh ồn ào; robot chỉ thực thi. (Trừ khi cần xác nhận nguy hiểm: ví dụ "không thể, phía trước có vật cản" – nếu tích hợp sensor).
- `head(action)` - **Điều khiển đầu (camera):** Kích hoạt khi user nói "nhìn lên/xuống/trước" hoặc "nhìn theo <ai đó>". *Hành vi:* Nếu cụ thể "nhìn lên/xuống/trước", Planner gửi lệnh SERVO với

id servo đầu, pos tương ứng (vd nhìn lên: pos=500 (góc cao); nhìn xuống: pos=2500; nhìn thẳng: pos=1500). Nếu là “nhìn theo <người>” hoặc “nhìn theo mình”, Planner kích hoạt **tracking mode**: ghim đối tượng đó (nếu “mình” thì người gần camera nhất) và mỗi khung hình Vision sẽ điều chỉnh servo để giữ đối tượng ở trung tâm (liên tục gửi SERVO).

- **light(color)** - **Đổi màu đèn**: Kích hoạt khi user nói đổi LED (ví dụ “đèn màu đỏ”, “nháy đèn xanh”). **Hành vi**: Parser lấy **color**, Planner chuyển thành giá trị RGB (ví dụ đỏ: 255,0,0), gửi gói LED. Nếu “nháy” thì có thể gửi nhiều lần on/off hoặc Arduino có chế độ nháy sẵn. TTS không cần phản hồi trừ phi muốn xác nhận (“Đã chuyển đèn màu đỏ”).

Trên đây chỉ là skeleton intents chính cho MVP. Hệ thống có thể mở rộng thêm như **acknowledge()** (khi user nói “tốt lắm” robot phản ứng vui vẻ), **idle()** (khi không có lệnh thì làm gì – có thể nhìn quanh hoặc phát ra âm thanh ngẫu nhiên nhẹ nhàng), nhưng giai đoạn này có thể chưa cần. Quan trọng là thiết kế module và cấu trúc **Policy/BehaviorPlanner** linh hoạt để dễ bổ sung intent mới. Mỗi intent thực thi nên **không chặn** (sử dụng coroutine cho các tác vụ như gửi Bluetooth, hoặc delay ngắn giữa các lệnh servo nếu cần tạo chuyển động mượt).

Tiêu chí Chất lượng & Kiểm thử

Để đảm bảo thiết kế trên hoạt động đúng mong đợi, nhóm phát triển sẽ kiểm thử theo một **checklist** các hạng mục sau, cũng như đặt tiêu chí chất lượng cụ thể:

• Kết nối & An toàn Bluetooth:

- **Pair & Connect**: Thử ghép nối điện thoại với module Bluetooth (HC-05). Mở app, vào màn hình Connect, đảm bảo thiết bị xuất hiện và kết nối thành công.
- **Reconnect**: Kiểm tra kịch bản mất kết nối (tắt HC-05 hoặc đi xa ngoài tầm) -> app phát hiện ngắt kết nối trong ~1s, robot dừng an toàn (do watchdog). Khi thiết bị quay lại phạm vi, app tự động kết nối lại hoặc người dùng nhấn connect lại.
- **Ping/Timeout**: Giả lập tình huống ứng dụng treo hoặc mất tín hiệu radio, Arduino không nhận lệnh trong >300ms -> kiểm tra động cơ dừng.
- **Throughput**: Gửi lệnh liên tục 50Hz, xác nhận Arduino nhận đầy đủ (có thể check seq và ack).

• Đa nhiệm Camera & Mic:

- Khởi chạy camera preview, sau đó nhấn giữ PTT nói chuyện -> đảm bảo **ASR** hoạt động song song, camera preview không bị gián đoạn (vẫn phát hiện khuôn mặt/đồ vật trong lúc nghe).
- Ngược lại, khi camera đang theo dõi một người di chuyển, thử robot phát TTS một câu dài -> camera vẫn cập nhật khung hình (đảm bảo thread tách biệt).
- **Barge-in**: Trong khi robot đang nói TTS, người dùng nhấn PTT nói lệnh mới -> robot phải ngừng nói ngay và lắng nghe. Kiểm tra log để chắc rằng TTS bị cancel kịp thời và ASR thu âm không nhiễu.

• Tính năng Học & Nhớ:

- **Teach Face**: Cho một người mới đứng trước camera, ra lệnh “Nhớ mặt X là <Tên>”. Đảm bảo quy trình lưu diễn ra: sau khi lệnh, robot có feedback (“Đã nhớ <Tên>”). Kiểm tra DB có bản ghi mới với label đó, embedding không rỗng. Sau đó thử tắt app mở lại (đảm bảo DB lưu bền).
- **Recall Face**: Với khuôn mặt đã lưu, người đó quay lại, hỏi “Ai đây?” -> robot nhận ra đúng tên >90% trường hợp khi mặt rõ ràng. Thủ trường hợp robot nhận sai hoặc không chắc (dùng người chưa lưu nhưng có nét giống) -> robot nên trả lời “Mình không biết” hoặc hỏi lại xác nhận, không được khẳng định sai.
- **Teach Object**: Tương tự, dạy một vật (“Dạy đồ vật này là ly nước”), kiểm tra DB lưu. Sau đó đưa vật đó vào khung và hỏi “Cái này là gì?” -> robot nhận đúng tên với độ chính xác chấp nhận (nếu vật na ná vật khác có trong model COCO, có thể nhầm; chấp nhận vì MVP).
- **Forget**: Dạy thử vài khuôn mặt/vật, sau đó ra lệnh “Quên <Tên>” cho một cái. Kiểm tra DB mục đó bị xóa, robot không còn nhận ra sau khi quên.

- **Điều khiển & Phản xạ:**

- **Drive Commands:** Thủ tất cả lệnh thoại di chuyển: tiến, lùi, trái, phải, dừng. Quan sát robot phản ứng lập tức và đúng hướng. Đặc biệt kiểm tra “đừng lại” có hiệu lực nhanh (nếu đang chạy tốc độ cao phải dừng trong <0.5s).
- **Servo Track:** Thủ lệnh “nhìn theo mình” hoặc tự động track khi phát hiện người: di chuyển trước camera, xem servo có quay mượt, độ trễ nhỏ. Kiểm tra biên độ servo không vượt giới hạn vật lý.
- **LED:** Thủ đổi màu LED qua lệnh, LED đổi đúng màu. Thủ “nhấp nháy” nếu có, LED nhấp nháy rõ ràng.
- **Sensor Feed:** (Nếu có sensor) kiểm tra app nhận được gói SENSOR_DATA liên tục (ví dụ mỗi 100ms), hiển thị đúng (có thể log hoặc vẽ khoảng cách).

- **Hiệu năng & Độ ổn định:**

- Đo thời gian từ lúc **user nói xong** đến lúc **bắt đầu nghe tiếng TTS** – mục tiêu ~0.5-0.7s. Thủ với các câu ngắn vs dài, offline vs online để tinh chỉnh (có thể bật mạng xem có trễ hơn không).
- Đo **FPS camera** khi chạy đồng thời mọi thứ (ASR, detection): kỳ vọng >=15FPS với Note 20 Ultra. Nếu thấp, xem xét dùng GPU delegate cho TFLite hoặc giảm kích thước ảnh đầu vào.
- Chạy thử liên tục >30 phút xem có rò rỉ bộ nhớ hoặc crash không (nhiệt độ thiết bị cũng nên theo dõi, nếu quá nóng có thể giảm tốc).
- Kiểm tra **pin**: chạy 15 phút xem tụt bao nhiêu %, đảm bảo có thể hoạt động ít nhất vài giờ trên điện thoại.

- **UX & An toàn:**

- Đảm bảo app chỉ thu âm khi người dùng PTT (biểu tượng mic sáng lên). Khi thả nút thì ngừng, TTS không tự ý nói khi không có lệnh để tránh làm phiền.
- Quyền truy cập: lần đầu mở app phải xin phép camera, mic, location (cho Bluetooth) – kiểm tra luồng request permission OK. Nếu từ chối, app thông báo hợp lý.
- Thủ “Kid mode”: nếu bật cờ Kid Mode, thủ ra lệnh nguy hiểm kiểu “đi nhanh tối đa” – robot có thể phớt lờ hoặc đáp “Xin lỗi, mình không thể làm vậy.”.
- Cuối cùng, thử trong bối cảnh thật với trẻ em: xem ngôn ngữ có phù hợp, robot có hành vi an toàn (không di chuyển quá mạnh, nhận dạng sai có gây hiểu lầm không).

Sau khi các bài test trên đều đạt kết quả tốt, chúng ta có thể tự tin rằng thiết kế MVP của Companion Android đã **đầy đủ tính năng, hoạt động ổn định và an toàn**. Các kết quả thử nghiệm và phản hồi người dùng thực tế sẽ định hướng cho Phase 2 (mở rộng tính năng, tối ưu hơn, có thể tích hợp AI đàm thoại nâng cao hoặc điều hướng SLAM nếu cần).

Phạm vi & Giới hạn MVP

Để giữ cho Phase 1 tập trung và khả thi, một số chức năng được **giới hạn phạm vi hoặc hoãn** sang giai đoạn sau:

- **Không phụ thuộc Cloud:** Tất cả xử lý (ASR, TTS, Vision) đều nội bộ. Wi-Fi hoặc mô hình ngôn ngữ lớn (LLM) không bắt buộc trong MVP. Tuy nhiên, thiết kế đã chứa sẵn giao diện mở rộng để sau này có thể tùy chọn tích hợp (ví dụ chuyển TTS sang cloud để có giọng tự nhiên hơn, hoặc gọi API chatbot GPT để có hội thoại phong phú hơn).
- **Chưa có Điều hướng thông minh:** MVP chưa làm về SLAM, lập bản đồ hay tránh vật cản nâng cao. Robot di chuyển đơn giản theo lệnh, dựa vào người dùng quan sát. Phase sau có thể thêm camera depth hoặc dùng sensor để tự tránh chướng ngại, nhưng Phase 1 không bao gồm.
- **Một đối tượng mỗi lượt học:** Khi vào chế độ “Dạy”, người dùng dạy **từng khuôn mặt hoặc vật thể một**. Chưa hỗ trợ dạy hàng loạt hoặc từ ảnh có sẵn. Điều này giúp đơn giản luồng UX và đảm bảo độ chính xác cho mỗi lần học. Người dùng có thể dạy nhiều mục nhưng tuần tự.
- **Giới hạn nhận dạng:** Robot chỉ nhận dạng được những gì model được huấn luyện hoặc đã dạy. Với vật thể, model MobileNet-SSD chỉ biết 80 danh mục chung (chó, mèo, chai, ghế, v.v.).

Những vật thể ngoài danh mục có thể chỉ được mô tả là “đồ vật”. Tương tự, face recognition chỉ phân biệt những người đã học; người lạ sẽ coi như “người lạ” chung chung.

- **Từ vựng & Ngôn ngữ:** MVP chủ yếu hoạt động tiếng Việt (có thể bổ sung vài câu lệnh tiếng Anh nếu cần cho test). Vốn từ được quy định sẵn trong Intent Router, chưa có NLP phức tạp. Người dùng cần nói đúng mẫu ở mức độ nào đó. Giai đoạn sau có thể dùng mô hình NLP cho hiểu ngôn ngữ tự nhiên linh hoạt hơn.
- **Giao diện Đơn giản:** UI MVP chỉ có những nút cơ bản (Connect, PTT, Teach) và preview camera. Chưa đầu tư nhiều vào đồ họa sinh động hay nhân vật hoạt hình, v.v. Mục tiêu là chức năng chạy đúng. Sau khi xong chức năng, có thể cải thiện UI/UX (thêm avatar hiển thị trạng thái, hướng dẫn trên màn hình, v.v.).

Với những giới hạn trên, MVP sẽ gọn nhẹ và tập trung, sau đó dựa trên phản hồi sử dụng thực tế để định hướng tính năng mở rộng. Mục tiêu cuối cùng là xây dựng một **Companion Robot** thông minh, tin cậy, có thể *nghe-hiểu, nhìn-nhớ và tương tác* tự nhiên với con người – Phase 1 đặt nền móng vững chắc cho những bước tiến đó.

Tài liệu Tham khảo:

1. Nikhil R, *How to optimise latency for voice agents* – Tầm quan trọng của độ trễ <800ms và tính năng barge-in trong trợ lý giọng nói [2](#).
2. Axel Sirota, *Implementing On-device Facial Recognition* – Phương pháp trích xuất **embedding** và so sánh để nhận dạng khuôn mặt trên Android (dùng model pre-trained, so khớp nearest-neighbor) [5](#) [11](#).
3. Dự án mẫu *On-Device Face Recognition Android* – Sử dụng FaceNet TFLite tạo 128-D embedding lưu trong cơ sở dữ liệu vector để nhận diện khuôn mặt người dùng [6](#) [14](#).
4. Blog TensorFlow, *Realtime Object Detection on Mobile* – Mô hình **SSD MobileNet** tối ưu cho thiết bị di động, cho phép nhận dạng vật thể thường gặp với hiệu năng thời gian thực [4](#).
5. ML Kit & Firebase Docs – API phát hiện khuôn mặt on-device, cung cấp vị trí và **ID theo dõi** để theo dấu khuôn mặt liên tục giữa các khung hình [13](#).
6. Cộng đồng StackOverflow & Arduino – Thảo luận về thiết kế giao thức nối tiếp đáng tin cậy (ví dụ sử dụng header 0x55 0xAA và byte stuffing để truyền dữ liệu nhị phân) [15](#).
7. Jetpack Compose + TFLite Sample – Ứng dụng mẫu kết hợp Compose và TensorFlow Lite để nhận diện vật thể realtime trên Android (SSD MobileNet v1) [12](#).
8. Android Developers – Hướng dẫn sử dụng CameraX phân tích hình ảnh và SpeechRecognizer song song trên thiết bị (tham khảo cấu hình luồng camera, tránh xung đột mic khi không thu âm video) [16](#).

(Các nguồn trên hỗ trợ cho việc lựa chọn công nghệ và phương pháp triển khai trong thiết kế. Thiết kế chi tiết có thể tinh chỉnh dựa trên thử nghiệm thực tế và phản hồi người dùng.)

[1](#) [5](#) [11](#) How to implement an AI-native mobile app: Facial recognition | Online Courses, Learning Paths, and Certifications - Pluralsight

<https://www.pluralsight.com/resources/blog/ai-and-data/ai-native-mobile-apps-guide>

[2](#) How to optimise latency for voice agents - Nikhil R
<https://rnikhil.com/2025/05/18/how-to-reduce-latency-voice-agents>

[3](#) Blog - Pure Speech Technology
<https://www.purespeechtechnology.com/blog/>

- ④ Training and serving a realtime mobile object detector in 30 minutes with Cloud TPUs — The TensorFlow Blog

<https://blog.tensorflow.org/2018/07/training-and-serving-realtime-mobile-object-detector-cloud-tpus.html>

- ⑥ ⑦ ⑩ ⑯ GitHub - shubham0204/OnDevice-Face-Recognition-Android: On-device customizable face recognition in Android with FaceNet and an embedded vector database

<https://github.com/shubham0204/OnDevice-Face-Recognition-Android>

- ⑧ ⑯ Communication Protocol - DFRobot WIKI

https://wiki.dfrobot.com/DFRobot_Communication_Protocol

- ⑨ How to deal with start sequence in payload of communication packet?

<https://electronics.stackexchange.com/questions/152857/how-to-deal-with-start-sequence-in-payload-of-communication-packet>

- ⑫ GitHub - martinlprb23/Object-Detection: Real-time object detection using TensorFlow lite, Jetpack Compose and the ssd_mobilenet_v1 model

<https://github.com/martinlprb23/Object-Detection>

- ⑬ How to track face with MLKit FaceDetector for images - Stack Overflow

<https://stackoverflow.com/questions/63596528/how-to-track-face-with-mlkit-facedetector-for-images>

- ⑯ Using SpeechRecognizer while recording video with enabled audio

https://www.reddit.com/r/androiddev/comments/s8nn2d/using_speechrecognizer_while_recording_video_with/