Bản Thiết Kế Kiến Trúc Toàn Diện Cho Hệ Thống Ghi Hình Thể Thao Thông Minh: Xử Lý Bộ Đệm Di Động, Điều Phối IoT Và Thị Giác Máy Tính Tự Động

1. Tổng Quan Điều Hành và Phạm Vi Chiến Lược

Sự hội tụ của sức mạnh tính toán di động, trí tuệ nhân tạo tại biên (Edge AI) và kết nối vạn vật (IoT) đang tạo ra một sự chuyển dịch mô hình trong lĩnh vực phân tích và phát sóng thể thao nghiệp dư. Báo cáo này cung cấp một bản phân tích kỹ thuật chuyên sâu và lộ trình triển khai chi tiết cho một nền tảng thể thao kép, giải quyết hai thách thức cốt lõi mà đội ngũ phát triển đang đối mặt: tính năng tự ghi hình với bộ đệm vòng (ring buffer) trên thiết bị di động và hệ thống camera tự động tại sân bãi được kích hoạt qua thanh toán QR.

Trong bối cảnh thị trường công nghệ thể thao (SportsTech) đang bùng nổ, nhu cầu về việc ghi lại những khoảnh khắc "highlight" mà không cần sự can thiệp thủ công liên tục là rất lớn. Đối với tính năng tự quay (Self-Recording), thách thức kỹ thuật không chỉ nằm ở việc ghi hình mà là quản lý bộ nhớ hiệu quả để lưu trữ hồi tố (retrospective saving) video chất lượng cao dựa trên kích hoạt giọng nói trong môi trường ồn ào. Đối với tính năng camera tại sân (Venue Capture), bài toán chuyển từ xử lý trên thiết bị sang kiến trúc hệ thống phân tán, đòi hỏi sự phối hợp chặt chẽ giữa cổng thanh toán, đám mây (Cloud) và thiết bị vật lý (IoT Device) với độ trễ thấp và bảo mật cao.

Bản báo cáo này tổng hợp dữ liệu từ hơn 500 tài liệu kỹ thuật, bao gồm các chuẩn nén video H.264/H.265, các thư viện nhận dạng giọng nói offline (Vosk, TensorFlow Lite), giao thức truyền tin IoT (MQTT), và các mô hình thị giác máy tính nhẹ (MobileNet, TrackNetV2). Mục tiêu là cung cấp một tài liệu tham chiếu kỹ thuật toàn diện, từ cấp độ mã hóa (code-level) đến kiến trúc hệ thống (system architecture), giúp giải quyết triệt để các khó khăn trong luồng xử lý và giao diện người dùng mà dự án đang gặp phải.

---

2. Kiến Trúc Tính Năng Tự Quay (Self-Recording): Xử Lý Bộ Đệm và Tương Tác Người Dùng

Tính năng tự quay đòi hỏi ứng dụng phải hoạt động như một "cỗ máy thời gian", liên tục ghi nhận dữ liệu nhưng chỉ lưu lại khi có sự kiện quan trọng. Vấn đề cốt lõi ở đây là

quản lý bộ nhớ RAM và Disk I/O để tránh hiện tượng tràn bộ nhớ hoặc nóng máy, đồng thời xử lý tín hiệu âm thanh song song để nhận diện từ khóa.

2.1. Cơ Chế Bộ Đệm Vòng (Circular Buffer) Cho Video Độ Phân Giải Cao

Khái niệm "ghi lại 30-60 giây trước đó" về mặt kỹ thuật không thể thực hiện bằng cách ghi liên tục vào bộ nhớ lưu trữ (Flash/SSD) vì sẽ gây hao mòn thiết bị và độ trễ I/O lớn. Giải pháp bắt buộc là sử dụng cấu trúc dữ liệu **Bộ Đệm Vòng (Circular Buffer)** trong bộ nhớ RAM.

2.1.1. Nguyên Lý Hoạt Động Của Bộ Đệm Video

Bộ đệm vòng là một vùng nhớ có kích thước cố định, hoạt động theo nguyên tắc FIFO (First-In, First-Out) đặc biệt: khi bộ đệm đầy, dữ liệu mới sẽ ghi đè lên dữ liệu cũ nhất. Trong bối cảnh video, thách thức lớn nhất không phải là cấu trúc dữ liệu mà là bản chất của luồng video nén. Video hiện đại (H.264/AVC hoặc H.265/HEVC) được cấu thành từ các Nhóm Hình Ảnh (Group of Pictures - GOP). Một GOP bắt đầu bằng một khung hình I-Frame (Intra-frame - chứa đầy đủ thông tin hình ảnh) và theo sau là các P-Frames (Predictive) và B-Frames (Bi-directional) chỉ chứa thông tin thay đổi.

Vấn Đề Cắt Video Tại Điểm Bất Kỳ: Nếu người dùng ra lệnh "Lưu highlight" tại thời điểm T, và hệ thống cắt video từ $T-30s$, điểm cắt đó có thể rơi vào một P-Frame. P-Frame này không thể giải mã nếu thiếu I-Frame đứng trước nó. Do đó, bộ đệm vòng phải được thiết kế để lưu trữ một khoảng thời gian *lớn hơn* thời gian highlight mong muốn, tối thiểu là `Target_Duration + GOP_Duration`. Ví dụ, để lưu 30 giây, và GOP của camera là 2 giây, bộ đệm cần chứa ít nhất 32-35 giây dữ liệu để đảm bảo luôn tìm thấy một I-Frame hợp lệ ở điểm bắt đầu.

2.1.2. Triển Khai Kỹ Thuật Trên Android (Kotlin/Java)

Trên Android, việc sử dụng `MediaRecorder` là không khả thi cho tính năng này vì API này được thiết kế để ghi trực tiếp xuống file. Kiến trúc đề xuất sử dụng **MediaCodec API** kết hợp với một lớp quản lý bộ đệm tùy chỉnh.

Kiến Trúc Luồng Dữ Liệu (Data Flow Architecture):

1. **Camera Input:** Sử dụng `Camera2 API` hoặc `CameraX` để đẩy dữ liệu hình ảnh thô (RAW) vào một `Surface`.
2. **Encoder (MediaCodec):** `MediaCodec` được cấu hình ở chế độ Encoder (mã hóa), nhận dữ liệu từ `Surface` và xuất ra các đơn vị NAL (Network Abstraction Layer) đã nén dưới dạng `ByteBuffer`.

3. **In-Memory Circular Buffer:** Thay vì đẩy `ByteBuffer` này vào `MediaMuxer` (để ghi ra file ngay), ứng dụng sẽ sao chép dữ liệu này vào một mảng vòng tròn trong RAM. Lớp `CircularEncoderBuffer` (tham khảo từ thư viện Grafika của Google) là mô hình chuẩn cho việc này.
    - Bộ đệm này không chỉ lưu dữ liệu video (byte array) mà phải lưu kèm siêu dữ liệu (Metadata): `BufferInfo` bao gồm timestamp (PTS) và cờ đánh dấu frame (Keyframe flag).
4. **Sự Kiện Trigger:** Khi người dùng nói "Highlight":
    - Hệ thống xác định timestamp hiện tại T
    - *now*
    -
    - .
    - Tìm kiếm ngược trong bộ đệm để xác định I-Frame gần nhất tại vị trí T
    - *start*
    -
    - $\approx T$
    - *now*
    -
    - $-30s$.
    - Khởi tạo `MediaMuxer` với đường dẫn file mới.
    - "Xả" (Drain) dữ liệu từ vị trí T
    - *start*
    -
    - đến T
    - *now*
    -
    - từ RAM vào `MediaMuxer`.
    - Đóng file và tiếp tục vòng lặp ghi hình mà không làm gián đoạn luồng encoder.

Tối Ưu Hóa Bộ Nhớ: Với video 1080p bitrate 5Mbps, 60 giây video sẽ chiếm khoảng: $5\ \text{Mbps} \times 60s / 8 = 37.5\ \text{MB}$. Đây là mức dung lượng RAM chấp nhận được trên các thiết bị Android hiện đại. Tuy nhiên, cần lưu ý sử dụng `ByteBuffer.allocateDirect()` để tránh overhead của Garbage Collection (GC) trong Java.

2.1.3. Triển Khai Kỹ Thuật Trên iOS (Swift/AVFoundation)

Hệ sinh thái iOS cung cấp các công cụ cấp cao hơn nhưng cũng chặt chẽ hơn về quản lý tài nguyên thông qua `AVFoundation`.

Cơ Chế CMSampleBuffer: Khác với Android xử lý byte thô, iOS đóng gói dữ liệu video trong các đối tượng `CMSampleBuffer`.

1. **AVCaptureSession:** Quản lý luồng tín hiệu từ camera.
2. **AVCaptureVideoDataOutput:** Delegate này trả về `CMSampleBuffer` cho mỗi khung hình (frame).
3. **Ring Buffer Logic:**
    - Tạo một mảng (Array) hoặc Danh sách liên kết (Linked List) chứa các `CMSampleBuffer`.
    - Trong hàm delegate `captureOutput(_:didOutput:from:)`, thêm buffer mới vào cuối mảng.
    - Kiểm tra timestamp của buffer đầu mảng. Nếu T
    - *new*
    -
    - $-T$
    - *old*
    -
    - $>30s$, giải phóng buffer cũ và xóa khỏi mảng.
    - **Cảnh báo bộ nhớ:** `CMSampleBuffer` giữ tham chiếu đến `CVPixelBuffer` (vùng nhớ chứa ảnh). Nếu không giải phóng đúng cách, ứng dụng sẽ bị crash do hết bộ nhớ (OOM) chỉ sau vài giây. Cần đảm bảo cơ chế Reference Counting của Swift hoạt động chính xác hoặc quản lý thủ công vùng nhớ này.

Quy Trình Xuất File (Export Process): Khi kích hoạt highlight:

1. Khởi tạo `AVAssetWriter` với URL file đích.
2. Sử dụng `AVAssetWriterInput` để nhận dữ liệu.
3. Duyệt qua mảng `CMSampleBuffer` hiện có và nạp lần lượt vào writer. Vì dữ liệu đã nằm sẵn trong RAM dưới dạng nén (nếu lấy từ output nén) hoặc raw (cần encode lại), quá trình này diễn ra rất nhanh.
4. Lưu ý quan trọng: `AVAssetWriter` yêu cầu timestamp phải bắt đầu từ 0 hoặc phải set `SessionStartTime` chính xác. Khi cắt highlight, cần thực hiện phép toán "Offset Timestamp" để frame đầu tiên của video mới có timestamp bắt đầu hợp lệ.

2.2. Nhận Diện Từ Khóa Offline (Offline Keyword Spotting - KWS)

Yêu cầu nhận diện từ khóa "Highlight" trong môi trường sân bãi ồn ào mà không có kết nối internet ổn định buộc phải sử dụng các mô hình AI chạy trực tiếp trên thiết bị (On-Device AI). Các giải pháp dựa trên Cloud API (như Google Speech-to-Text) sẽ có độ trễ quá cao và không tin cậy.

2.2.1. Lựa Chọn Công Nghệ: Vosk vs. TensorFlow Lite

Có hai hướng tiếp cận chính cho bài toán này: sử dụng thư viện nhận dạng giọng nói tổng quát (ASR) hoặc sử dụng mô hình chuyên biệt cho từ khóa (Wake Word Engine).

Phương Án 1: Vosk (Kaldi-based) Vosk là một bộ công cụ nhận dạng giọng nói offline mã nguồn mở, hỗ trợ nhiều ngôn ngữ và chạy tốt trên Android/iOS.

- **Ưu điểm:** Hỗ trợ từ vựng rộng, không cần huấn luyện lại nếu từ khóa là từ phổ thông. Có thể nhận diện cả câu lệnh phức tạp như "Lưu pha bóng vừa rồi".
- **Nhược điểm:** Kích thước mô hình khá lớn (~50MB), tiêu tốn CPU nhiều hơn so với các engine chuyên biệt chỉ bắt 1 từ khoá. Trong bối cảnh vừa quay video (tốn GPU/CPU) vừa nghe, Vosk có thể gây nóng máy.

Phương Án 2: TensorFlow Lite (Khuyên dùng) Huấn luyện một mô hình Deep Learning nhỏ (Micro-controller level) chuyên biệt để phát hiện đúng từ khóa "Highlight".

- **Kiến trúc:** Sử dụng mạng nơ-ron tích chập chiều sâu (Depthwise Separable CNN - DS-CNN) hoặc MobileNetV3 thu nhỏ. Mô hình này nhận đầu vào là phổ âm thanh (spectrogram) của 1 giây âm thanh gần nhất.
- **Ưu điểm:** Kích thước cực nhỏ (< 1MB), độ trễ cực thấp (< 200ms), tiêu thụ pin không đáng kể. Rất phù hợp để chạy nền song song với quay video.
- **Quy trình:**
    1. Thu thập dataset: Ghi âm từ "Highlight" từ nhiều người khác nhau, trộn với nhiễu nền sân bóng (tiếng còi, tiếng giày rít, tiếng gió).
    2. Huấn luyện mô hình trên TensorFlow và convert sang `.tflite`.
    3. Trên Mobile App: Chạy một luồng (Thread) riêng đọc Audio Buffer, cắt thành các đoạn 1s (cửa sổ trượt), đưa vào mô hình TFLite để lấy xác suất. Nếu $P(Highlight) > 0.85$, kích hoạt sự kiện lưu video.

2.2.2. Xử Lý Xung Đột Tài Nguyên Audio (Audio Concurrency)

Một vấn đề kỹ thuật lớn trên Android là quyền truy cập Microphone. Thông thường, `MediaRecorder` sẽ chiếm quyền độc quyền Microphone. Nếu bạn khởi tạo thêm một `AudioRecord` cho AI nhận diện giọng nói, ứng dụng sẽ bị crash hoặc một trong hai tiến trình sẽ không nhận được dữ liệu.

Giải Pháp: Kiến Trúc Audio Dispatcher Không thể để hai thành phần cùng truy cập phần cứng. Cần xây dựng một lớp trung gian:

1. Khởi tạo **một** đối tượng `AudioRecord` (Android) hoặc `AudioUnit` (iOS) duy nhất đọc dữ liệu PCM thô (Raw PCM) từ Microphone.
2. Dữ liệu này được đẩy vào một bộ phân phối (Dispatcher).
3. Dispatcher sẽ sao chép luồng byte này ra hai bộ đệm riêng biệt:
    - **Buffer A:** Đưa vào bộ mã hóa âm thanh (AAC Encoder) để lồng vào video.
    - **Buffer B:** Đưa vào bộ tiền xử lý (Downsampling về 16kHz) để phục vụ cho mô hình AI nhận diện từ khóa.

2.3. Thiết Kế Giao Diện UI/UX Cho Tương Tác Rảnh Tay (Hands-Free)

Người dùng khi đang thi đấu thể thao không thể nhìn chăm chú vào màn hình hay thao tác chính xác trên các nút nhỏ. Giao diện phải tuân thủ nguyên tắc **"Glanceability"** (Khả năng nắm bắt thông tin trong nháy mắt) và phản hồi đa giác quan.

2.3.1. Phản Hồi Thị Giác và Thính Giác (Feedback Loops)

- **Flash Visual Cue:** Khi hệ thống nhận diện được từ khóa "Highlight", toàn bộ màn hình hoặc viền màn hình nên lóe sáng (ví dụ: màu xanh lá cây hoặc cam) trong 500ms. Điều này giúp vận động viên nhận biết lệnh đã được thực thi ngay cả khi đứng xa thiết bị 5-10 mét.
- **Auditory Confirmation:** Phát ra một âm thanh đặc trưng (như tiếng màn trập máy ảnh hoặc tiếng "ping" lớn) để xác nhận. Âm thanh này cần khác biệt với tiếng còi trọng tài hoặc tiếng la hét trên sân.
- **Trạng Thái Lắng Nghe:** Hiển thị một biểu đồ sóng âm (waveform) nhỏ ở góc màn hình đang chuyển động theo thời gian thực. Điều này trấn an người dùng rằng micro đang hoạt động và đang "nghe" họ, tránh lo lắng về việc lệnh không được nhận.

2.3.2. Giao Diện Chỉnh Sửa Timeline Trên Mobile

Sau khi trận đấu kết thúc, người dùng sẽ có một danh sách các clip highlights rời rạc. Việc ghép nối cần trực quan:

- **Timeline Tương Tác:** Sử dụng mô hình timeline nằm ngang (như Instagram Reels hoặc TikTok Editor). Các clip highlight được biểu diễn bằng các khối chữ nhật có thumbnail.
- **Thao Tác Cử Chỉ (Gestures):**
    - **Long-press & Drag:** Để thay đổi thứ tự các pha bóng.
    - **Pinch-to-zoom:** Phóng to timeline để cắt (trim) chính xác từng giây đầu/cuối của clip.
    - **Magnetic Snapping:** Khi kéo thả các clip gần nhau, chúng phải tự động "hút" vào nhau để không tạo ra khoảng đen (black gap) giữa các video.
- **Auto-Transition:** Hệ thống nên đề xuất tự động thêm các hiệu ứng chuyển cảnh (cross-dissolve) giữa các clip để video mượt mà hơn mà người dùng không cần chỉnh thủ công từng chút.

---

3. Kiến Trúc Tính Năng Quét QR Tại Sân: IoT và Tự Động Hóa

Tính năng này chuyển dịch vai trò ghi hình từ người dùng sang cơ sở vật chất của sân bãi. Mô hình này yêu cầu một hệ thống IoT phức tạp để quản lý phiên (session), thanh toán và kích hoạt thiết bị từ xa.

3.1. Luồng Triển Khai Người Dùng (Deployment Flow)

Vấn đề "khó khăn trong luồng triển khai" thường nằm ở việc đồng bộ trạng thái giữa ba bên: App người dùng, Server (Backend) và Camera tại sân.

Luồng Quy Trình Chuẩn Hóa:

1. **Khởi tạo:** Người dùng quét mã QR dán cố định tại sân (mã này chứa `court_id` và `venue_id`).
2. **Deep Link:** Mã QR kích hoạt Deep Link mở ứng dụng (hoặc Instant App nếu chưa cài đặt) đi thẳng vào màn hình thanh toán cho sân đó.
3. **Thanh toán:** Người dùng thanh toán qua cổng (MoMo/ZaloPay/Stripe).
4. **Kích hoạt:** Sau khi thanh toán thành công, Server gửi lệnh xuống Camera tại sân. Camera bật lên, đèn trạng thái chuyển sang màu đỏ (Recording).

5. **Kết thúc & Trả hàng:** Khi hết giờ hoặc trận đấu xong, Camera tự tắt, video được xử lý tại biên (Edge) hoặc Cloud và gửi thông báo (Push Notification) cho người dùng tải về.

3.2. Giải Pháp Công Nghệ Cho IoT Trigger

3.2.1. Webhook và Bảo Mật Thanh Toán

Webhook là cơ chế để cổng thanh toán thông báo cho Server của bạn rằng "Tiền đã về". Tuy nhiên, đây là điểm yếu bảo mật nếu không xử lý kỹ.

- **Vấn đề:** Hacker có thể giả lập một request webhook để kích hoạt camera miễn phí.
- **Giải pháp HMAC (Hash-based Message Authentication Code):**
    - Cổng thanh toán sẽ gửi kèm một header chữ ký (Signature), được tạo ra bằng cách băm (hashing) nội dung gói tin (payload) với một khóa bí mật (secret key).
    - Server của bạn cần tính toán lại chuỗi hash này từ payload nhận được và so sánh với header. Nếu khớp, request là hợp lệ.
    - **Idempotency (Tính nhất quán):** Cần xử lý trường hợp Webhook được gửi trùng lặp (retry). Server phải kiểm tra `transaction_id`, nếu đã xử lý rồi thì trả về `200 OK` ngay mà không kích hoạt lại camera.

3.2.2. Giao Thức Điều Khiển Camera: MQTT vs. WebSockets

Để Server gửi lệnh "Bật Camera" xuống thiết bị tại sân, HTTP truyền thống không phù hợp vì Server không thể chủ động gọi xuống Client (Camera) nếu Client nằm sau NAT/Firewall của mạng Wifi sân bóng.

Lựa Chọn Tối Ưu: MQTT (Message Queuing Telemetry Transport) MQTT là giao thức chuẩn cho IoT, hoạt động theo mô hình Publish/Subscribe.

- **Tại sao chọn MQTT?**
    - **Nhẹ & Tiết kiệm băng thông:** Header chỉ 2 byte, cực kỳ quan trọng nếu sân bóng dùng 4G/Wifi chập chờn.
    - **Keep-Alive & LWT:** MQTT có cơ chế giữ kết nối và "Di chúc" (Last Will and Testament). Nếu Camera mất kết nối (mất điện/mạng), Broker sẽ báo ngay cho Server biết để cập nhật trạng thái trên App người dùng ("Camera Offline").

- **QoS (Quality of Service):** Sử dụng QoS level 1 hoặc 2 để đảm bảo lệnh "Bật Camera" chắc chắn đến được thiết bị, ngay cả khi mạng bị gián đoạn tạm thời.

Kiến Trúc Topic:

- Camera Subscribe topic: `venues/{venue_id}/courts/{court_id}/cmd`
- Server Publish lệnh: `{"command": "START_RECORDING", "session_id": "xyz", "duration": 3600}`
- Camera Publish trạng thái: `venues/{venue_id}/courts/{court_id}/status` (gửi `RECORDING`, `IDLE`, `ERROR`).

3.3. Tự Động Bắt Highlight Bằng AI Tại Biên (Edge AI)

Để camera "tự bắt highlight", việc gửi toàn bộ video lên Cloud để xử lý là quá tốn kém về băng thông và chi phí server. Giải pháp là **Edge Computing** (Xử lý tại biên). Camera sẽ được kết nối với một máy tính nhỏ (như NVIDIA Jetson Nano hoặc Raspberry Pi có AI Accelerator).

Pipeline Xử Lý AI Đề Xuất:

1. **Input:** Luồng video từ Camera.
2. **Object Detection (Phát hiện đối tượng):** Sử dụng mô hình **YOLOv8-Nano** hoặc **NanoDet** để phát hiện người chơi và bóng. Các mô hình này cực nhẹ, chạy tốt trên thiết bị biên.
3. **Ball Tracking (Theo dõi bóng):** Đối với các môn bóng nhỏ (tennis, cầu lông), YOLO thường không đủ chính xác. Cần sử dụng **TrackNetV2** - một mạng chuyên dụng để theo dõi quỹ đạo bóng tốc độ cao.
4. **Action Recognition (Nhận diện hành động):** Để biết đâu là "Highlight" (ví dụ: cú đập bóng ghi điểm), cần phân tích chuỗi hành động. Sử dụng mô hình **MoViNet (Mobile Video Networks)** hoặc **TSM (Temporal Shift Module)**. Đây là các mô hình được thiết kế tối ưu cho mobile/edge, sử dụng kỹ thuật "stream buffering" để phân tích video theo thời gian thực mà không tốn quá nhiều bộ nhớ.
5. **Heuristic Logic (Logic suy luận):** Kết hợp AI với logic âm thanh. Highlight thường đi kèm với âm thanh lớn (tiếng đập bóng, tiếng vỗ tay). Nếu AI phát hiện hành động "Smash" VÀ âm thanh tăng đột ngột -> Xác suất Highlight là 99%.

Kết quả: Thiết bị biên chỉ cắt ra các đoạn clip ngắn (10-15 giây) và gửi các clip này lên Cloud, giảm tải 95% băng thông so với gửi toàn bộ video trận đấu.

---

4. Kết Luận và Lộ Trình Triển Khai

Để giải quyết hai vấn đề cốt lõi của sản phẩm, đội ngũ phát triển cần tách biệt rõ ràng kiến trúc giữa Client-Side (App) và Edge-Side (Sân bãi).

1. **Với tính năng Tự Quay:** Tập trung nguồn lực vào việc tối ưu hóa **Bộ Đệm Vòng MediaCodec** và tích hợp mô hình **TensorFlow Lite** cho nhận diện từ khóa offline. Đây là nền tảng để tạo ra trải nghiệm mượt mà, không phụ thuộc vào internet.
2. **Với tính năng Quét QR:** Xây dựng hệ thống Backend dựa trên **MQTT** để quản lý trạng thái thiết bị theo thời gian thực. Đầu tư vào phần cứng Edge AI (như Jetson Nano) cài đặt sẵn **TrackNetV2/MoViNet** để xử lý video tại chỗ, giải quyết bài toán chi phí vận hành đường truyền.

Bằng cách áp dụng các mô hình kiến trúc này, sản phẩm không chỉ giải quyết được các khó khăn hiện tại mà còn tạo ra lợi thế cạnh tranh bền vững về công nghệ trong thị trường SportsTech.

Phần I: Kiến Trúc Kỹ Thuật Tính Năng Tự Quay (Self-Recording Architecture)

Trong kỷ nguyên của nội dung số do người dùng tạo ra (UGC), tính năng tự quay phim thông minh không chỉ đơn thuần là việc ghi lại hình ảnh mà là một bài toán phức tạp về quản lý tài nguyên hệ thống, tối ưu hóa trải nghiệm người dùng (UX) và xử lý tín hiệu thời gian thực. Phần này sẽ đi sâu vào các giải pháp kỹ thuật cụ thể để hiện thực hóa khả năng "quay ngược thời gian" và "điều khiển bằng giọng nói" trên thiết bị di động.

1.1. Cơ Chế Bộ Đệm Vòng (Circular Buffer): Trái Tim Của Tính Năng Highlight

Yêu cầu "tự cắt video 30s-60s trước đó" đặt ra một thách thức lớn về kiến trúc lưu trữ. Nếu ứng dụng ghi toàn bộ video vào ổ cứng (Flash Storage) rồi mới cắt, nó sẽ nhanh

chóng làm đầy bộ nhớ thiết bị và gây nóng máy do ghi đĩa liên tục (Disk I/O). Giải pháp tối ưu là giữ video trong bộ nhớ tạm (RAM) và chỉ ghi xuống đĩa khi cần thiết.

1.1.1. Bản Chất Kỹ Thuật Của Bộ Đệm Video

Khác với bộ đệm văn bản hay âm thanh, bộ đệm video chứa các khung hình (frames) có mối quan hệ phụ thuộc lẫn nhau. Trong chuẩn nén phổ biến H.264 (AVC) hoặc H.265 (HEVC), luồng video bao gồm:

- **I-Frame (Intra-frame / Keyframe):** Khung hình độc lập, chứa toàn bộ thông tin hình ảnh. Đây là điểm duy nhất có thể bắt đầu giải mã một đoạn video.
- **P-Frame (Predictive frame):** Chỉ chứa thông tin thay đổi so với khung hình trước đó.
- **B-Frame (Bi-directional frame):** Chứa thông tin thay đổi so với cả khung hình trước và sau.

Vấn đề: Nếu người dùng yêu cầu "lấy 30 giây cuối", nhưng thời điểm 30 giây trước đó rơi vào một P-Frame, trình phát video sẽ không thể hiển thị hình ảnh cho đến khi gặp I-Frame tiếp theo. Điều này tạo ra video bị lỗi (artifact) hoặc màn hình đen ở đầu clip.

Giải pháp Kiến trúc: Hệ thống bộ đệm vòng phải được thiết kế dựa trên cấu trúc GOP (Group of Pictures). Kích thước bộ đệm không được tính bằng giây cứng nhắc (ví dụ: đúng 30s) mà phải tính bằng:

$$\text{Buffer\_Size} \geq \text{Target\_Duration} + \text{GOP\_Duration}$$

Nếu khoảng cách giữa hai I-Frame (GOP length) là 2 giây, bộ đệm cần lưu ít nhất 32 giây để đảm bảo luôn bao chứa một I-Frame hợp lệ nằm *trước* mốc thời gian cắt mong muốn.

1.1.2. Triển Khai Trên Android (Deep Dive)

Trên Android, việc triển khai bộ đệm vòng hiệu năng cao đòi hỏi phải vượt qua giới hạn của `MediaRecorder` và tiếp cận tầng thấp hơn là `MediaCodec`.

Mô hình Kiến Trúc Đề Xuất:

1. **Tầng Thu Nhận (Input Layer):** Sử dụng `SurfaceTexture` để nhận dữ liệu hình ảnh từ Camera. Texture này được chia sẻ (shared context) giữa hai luồng: một để hiển thị preview lên màn hình (`SurfaceView`) và một để đưa vào bộ mã hóa (`MediaCodec`).

2. **Tầng Mã Hóa & Đệm (Encoding & Buffering Layer):**
    - `MediaCodec` (Encoder) xuất ra các `ByteBuffer` chứa dữ liệu nén (H.264 NAL units).
    - Thay vì gửi các `ByteBuffer` này trực tiếp tới `MediaMuxer` (như cách quay phim thông thường), ta đưa chúng vào một lớp quản lý bộ nhớ tùy chỉnh: `CircularEncoderBuffer`.
    - `CircularEncoderBuffer` là một mảng vòng tròn (Ring Array) trong Java/Kotlin. Nó lưu trữ hai thành phần:
        - Dữ liệu video thực tế (được nối liền trong một mảng byte lớn để giảm phân mảnh bộ nhớ).
        - Siêu dữ liệu (`MediaCodec.BufferInfo`): chứa timestamp, kích thước frame, và quan trọng nhất là cờ hiệu `BUFFER_FLAG_KEY_FRAME`.
3. **Tầng Xuất File (Output Layer):**
    - Khi nhận tín hiệu "Lưu Highlight":
        - Hệ thống xác định timestamp hiện tại T
        - now
        -
        - .
        - Thuật toán tìm kiếm ngược (Backtracking) trong mảng siêu dữ liệu để tìm index của I-Frame có timestamp T
        - $start$
        -
        - $\leq (T$
        - now
        -
        - $-30s)$.
        - Khởi tạo `MediaMuxer` mới với đường dẫn file (ví dụ: `/sdcard/Highlights/clip_01.mp4`).
        - Chạy vòng lặp đọc dữ liệu từ `CircularEncoderBuffer` bắt đầu từ T
        - $start$
        -
        - và ghi vào `MediaMuxer`.
        - **Lưu ý quan trọng:** Cần điều chỉnh lại timestamp (PTS - Presentation Time Stamp) của frame đầu tiên về 0 để video có thể phát bình thường trên mọi trình phát.

Bảng: So Sánh Các Phương Pháp Ghi Hình Trên Android

Phương Pháp	Ưu Điểm	Nhược Điểm	Đánh Giá Cho Dự Án
MediaRecorder	Dễ triển khai, code ngắn gọn.	Ghi trực tiếp ra file, không thể truy cập frame để đệm.	**Không phù hợp.**
MediaCodec + File	Kiểm soát frame tốt.	Ghi ra file tạm liên tục rồi cắt (file I/O cao), hại bộ nhớ.	**Kém hiệu quả.**
MediaCodec + RAM Ring Buffer	Hiệu năng cao nhất, không hại bộ nhớ Flash, cắt chính xác.	Phức tạp, cần quản lý bộ nhớ thủ công (Memory Leak risk).	**Khuyên dùng (Best Practice).**

1.1.3. Triển Khai Trên iOS (Deep Dive)

Hệ sinh thái Apple cung cấp framework `AVFoundation` rất mạnh mẽ nhưng cũng cực kỳ nghiêm ngặt về quản lý bộ nhớ. Việc rò rỉ bộ nhớ (Memory Leak) trong xử lý video sẽ khiến ứng dụng bị hệ điều hành "kill" ngay lập tức.

Chiến Lược AVAssetWriter & CMSampleBuffer:

1. **Capture Pipeline:**
    - Sử dụng `AVCaptureSession` để thiết lập luồng quay.
    - Output đi qua `AVCaptureVideoDataOutput`. Tại đây, delegate `captureOutput` sẽ trả về các đối tượng `CMSampleBuffer`.
2. **Quản Lý Bộ Nhớ Vòng:**
    - Tạo một mảng `var buffer: =`.
    - Mỗi khi nhận một `sampleBuffer` mới:
        - Giữ lại buffer đó: `buffer.append(sampleBuffer)`.
        - Kiểm tra thời lượng: Lấy timestamp của frame mới nhất trừ đi timestamp của frame đầu tiên trong mảng. Nếu > 30s, loại bỏ frame đầu tiên: `buffer.removeFirst()`.
    - **Thách thức cốt tử:** `CMSampleBuffer` tham chiếu đến `CVPixelBuffer` (vùng nhớ chứa ảnh thực tế). Swift ARC (Automatic Reference Counting) thường xử lý tốt, nhưng nếu bạn vô tình tạo reference cycle, bộ nhớ sẽ

đầy rất nhanh. Cần đảm bảo mảng buffer không bị giữ bởi các closure (block) khác.

3. **Cơ Chế Ghi (Writing Mechanism):**
    - Khi người dùng bấm lưu, không thể dùng `AVCaptureMovieFileOutput` vì nó ghi luồng trực tiếp.
    - Phải dùng `AVAssetWriter`.
    - Khởi tạo `AVAssetWriter` trỏ tới file output.
    - Duyệt qua mảng `buffer` trong RAM và gọi lệnh `writerInput.append(sampleBuffer)`.
    - **Tối ưu hóa:** Vì các `sampleBuffer` đã nằm trong RAM, tốc độ ghi xuống đĩa sẽ nhanh hơn nhiều so với thời gian thực (Real-time), cho phép lưu video 30s chỉ trong vài giây xử lý.

1.2. Nhận Diện Từ Khóa Offline (KWS): Thách Thức Môi Trường Thể Thao

Việc người dùng hô "Highlight" trên sân bóng đặt ra hai bài toán: **Độ trễ** (phải bắt ngay lập tức) và **Nhiễu** (tiếng gió, tiếng la hét). Giải pháp Cloud (như Siri) bị loại bỏ vì phụ thuộc internet và độ trễ cao.

1.2.1. Lựa Chọn Stack Công Nghệ AI On-Device

So Sánh Các Thư Viện:

Thư Viện	Kiến Trúc	Kích Thước	Độ Chính Xác (Noise)	Đề Xuất
Vosk (Kaldi)	FST Graph	~50MB	Trung bình. Tốt cho câu dài, nhưng nặng nề cho 1 từ khóa.	Dùng nếu cần nhận diện câu lệnh phức tạp.
Pocketsphinx	HMM-GMM	Nhỏ	Kém. Công nghệ cũ, dễ sai trong môi trường ồn.	Không nên dùng.

TensorFlow Lite (Micro)	DS-CNN / MobileNet	< 1MB	Cao. Có thể huấn luyện riêng cho từ khóa cụ thể kèm nhiễu nền.	**Khuyên dùng.**

Chiến Lược Huấn Luyện TFLite: Để đạt độ chính xác cao nhất cho từ khóa "Highlight", không nên dùng mô hình có sẵn (pre-trained) mà nên huấn luyện lại (Transfer Learning):

1. **Dataset:** Thu âm từ "Highlight" từ 20-30 người khác nhau (nam/nữ, giọng địa phương).
2. **Augmentation (Làm giàu dữ liệu):** Trộn các file ghi âm này với "âm thanh nền sân bóng" (tiếng giày, tiếng gió, tiếng thở dốc) ở các mức âm lượng khác nhau (Signal-to-Noise Ratio).
3. **Model:** Sử dụng kiến trúc **DS-CNN (Depthwise Separable Convolutional Neural Network)**. Đây là kiến trúc chuẩn cho KWS trên thiết bị di động, cân bằng hoàn hảo giữa độ chính xác và mức tiêu thụ pin.

1.2.2. Kiến Trúc Đa Luồng Audio (Audio Concurrency Architecture)

Một lỗi phổ biến khi phát triển tính năng này là xung đột tài nguyên Microphone. Hệ điều hành Android (và đôi khi iOS) không cho phép hai ứng dụng hoặc hai luồng cùng "chiếm" Microphone theo các cấu hình khác nhau (ví dụ: Video Recorder cần 48kHz Stereo, trong khi AI KWS cần 16kHz Mono).

Giải Pháp: Audio Dispatcher Pattern Hệ thống cần được thiết kế theo mô hình "Một nguồn, nhiều đích" (One Source, Multiple Sinks):

1. **Audio Capture Thread:** Một luồng ưu tiên cao duy nhất chịu trách nhiệm đọc dữ liệu PCM thô từ phần cứng (`AudioRecord`).
2. **The Splitter (Bộ chia):** Dữ liệu PCM thô được sao chép (copy) vào hai bộ đệm vòng tròn riêng biệt:
    - *Ring Buffer A:* Phục vụ cho `MediaCodec` (Video Encoder). Tại đây dữ liệu được giữ nguyên (hoặc convert sang định dạng encoder yêu cầu) để lồng vào video.
    - *Ring Buffer B:* Phục vụ cho `TFLite Interpreter`. Tại đây dữ liệu có thể cần được Downsampling (giảm mẫu) từ 44.1kHz xuống 16kHz để phù hợp với mô hình AI.

3. **Thread Safety:** Việc truy cập vào các bộ đệm này phải được đồng bộ hóa (synchronized) để tránh Race Condition (điều kiện đua), đảm bảo dữ liệu âm thanh trong video không bị giật cục.

1.3. UI/UX: Thiết Kế Cho Tương Tác Từ Xa (Distant Interaction)

Người dùng đặt điện thoại trên tripod và ra sân thi đấu (cách xa 5-10m). Giao diện người dùng (UI) và Trải nghiệm người dùng (UX) phải được thiết kế để họ có thể tương tác mà không cần nhìn rõ màn hình.

1.3.1. Phản Hồi Đa Giác Quan (Multimodal Feedback)

Khi người dùng hét "Highlight", họ cần biết chắc chắn máy đã nghe thấy.

- **Phản Hồi Thị Giác (Visual Flash):** Thiết kế màn hình sao cho khi lệnh được kích hoạt, một lớp phủ (overlay) màu tương phản cao (như xanh Neon hoặc Cam) sẽ nháy lên toàn màn hình trong 0.5 - 1 giây. Điều này cho phép người dùng nhận biết tín hiệu từ xa dưới ánh sáng mặt trời.
- **Phản Hồi Thính Giác (Auditory Cue):** Phát ra một âm thanh "Chime" tần số cao (để cắt qua tiếng ồn trầm của môi trường). Âm thanh này cần khác biệt hoàn toàn với tiếng còi trọng tài hay tiếng thông báo điện thoại thông thường.

1.3.2. Giao Diện Chỉnh Sửa Timeline (Mobile Editing)

Sau trận đấu, người dùng cần ghép các highlight lại. Giao diện chỉnh sửa trên mobile cho video thể thao cần sự chính xác nhưng đơn giản.

- **Mô Hình Timeline:** Sử dụng timeline nằm ngang với khả năng zoom (pinch-to-zoom). Các clip highlight được biểu diễn bằng các khối (blocks).
- **Snap-to-Grid:** Khi người dùng kéo thả các clip, chúng phải tự động "hít" (snap) vào nhau. Tránh việc người dùng phải căn chỉnh thủ công từng frame để xóa khoảng đen.
- **Coach Marks:** Vì tính năng tự quay bằng giọng nói là mới lạ, lần đầu mở app cần có các lớp hướng dẫn (Coach Marks) phủ lên màn hình, giải thích ngắn gọn: "Đặt máy xuống, lùi lại và nói Highlight".

Phần II: Kiến Trúc Tính Năng Quét QR & IoT Tại Sân (Venue Recording Architecture)

Tính năng này chuyển đổi mô hình từ "User-Device" sang "User-Cloud-IoT". Đây là một hệ thống phân tán phức tạp, nơi độ tin cậy của kết nối và tính bảo mật của giao dịch là yếu tố sống còn.

2.1. Luồng Triển Khai & Hành Trình Người Dùng (Deployment Flow)

Để giải quyết "khó khăn trong luồng triển khai", chúng ta cần một quy trình khép kín, giảm thiểu ma sát (friction) cho người dùng nhưng chặt chẽ về mặt kỹ thuật cho hệ thống.

Sơ Đồ Luồng Tương Tác (Interaction Sequence):

1. **The Trigger (QR Scan):** Mã QR tại sân không chỉ chứa URL. Nó chứa payload định danh: `myapp://venue/payment?vid=SanCauLong_A&cid=San_1`. Sử dụng Deep Link (iOS Universal Links / Android App Links) để mở trực tiếp App vào màn hình thanh toán của đúng sân đó. Nếu chưa cài App, chuyển hướng về Store.
2. **The Transaction (Thanh toán):** Người dùng chọn gói (ví dụ: "1 Giờ - 50k"). Thực hiện thanh toán qua ví điện tử.
3. **The Actuation (Kích hoạt):**
    - Ngay khi App nhận tín hiệu thanh toán thành công (Client-side success), App chuyển sang giao diện "Đang kết nối Camera..." (Optimistic UI).
    - Server nhận Webhook từ cổng thanh toán -> Gửi lệnh MQTT xuống Camera -> Camera bật -> Camera phản hồi trạng thái "RECORDING" -> App cập nhật giao diện "Đang Ghi Hình".
4. **The Delivery (Trả hàng):** Khi hết giờ, Camera tự tắt, video highlight được xử lý tại biên (Edge) và upload lên Cloud. Server gửi Push Notification cho người dùng.

2.2. Giải Pháp Bảo Mật Cho Giao Dịch IoT

Việc kích hoạt một thiết bị vật lý (tốn điện, băng thông, khấu hao) thông qua Internet đòi hỏi bảo mật nghiêm ngặt để tránh việc "dùng chùa".

2.2.1. Webhook Security: Chống Giả Mạo & Tấn Công Phát Lại

Cổng thanh toán (Payment Gateway) sẽ gọi vào API của bạn (Webhook) để báo tiền đã về. Hacker có thể bắt gói tin này và gửi lại (Replay Attack) để kích hoạt camera lần nữa.

Cơ Chế Xác Thực HMAC:

1. **Chữ Ký Số (Signature):** Cổng thanh toán sẽ băm (hash) toàn bộ nội dung (payload) cùng với timestamp và một khóa bí mật (Secret Key) mà chỉ bạn và cổng thanh toán biết. Kết quả băm được gửi trong Header (ví dụ: `X-Signature`).
2. **Kiểm Tra Tại Server:**
    - Lấy Payload và Timestamp từ request.
    - Tự tính toán hash: `HMAC_SHA256(Payload + Timestamp, Secret_Key)`.
    - So sánh hash tự tính với `X-Signature`. Nếu khớp tuyệt đối -> Request hợp lệ.
3. **Chống Replay Attack:** Kiểm tra Timestamp trong header. Nếu thời gian gửi cách thời gian hiện tại quá 5 phút -> Từ chối request (dù chữ ký đúng). Điều này ngăn hacker lưu lại gói tin hợp lệ và gửi lại sau đó.

2.2.2. Quản Lý Phiên (Session Management) & Idempotency

Hệ thống cần xử lý logic: "Nếu người dùng quét QR khi sân đang có người khác quay thì sao?".

- **Cơ Chế Khóa (Locking Mechanism):** Mỗi Camera trên Database có trạng thái: `IDLE` (Rảnh) hoặc `BUSY` (Bận).
- **Xử Lý Xung Đột:** Khi có request thanh toán, Server kiểm tra trạng thái. Nếu `BUSY`, trả về thông báo cho App: "Sân đang được ghi hình bởi người khác".
- **Idempotency (Tính duy nhất):** Webhook thanh toán có thể bị gửi 2 lần do lỗi mạng. Server phải lưu `Transaction_ID`. Nếu nhận được ID đã xử lý, Server trả về thành công nhưng *không* gửi lệnh kích hoạt Camera lần nữa.

2.3. Giao Thức Điều Khiển Camera: Tại Sao MQTT Thắng WebSockets?

Môi trường sân bãi thường có kết nối Internet không ổn định (Wifi chập chờn, 4G yếu).

So Sánh Kỹ Thuật:

- **WebSockets:** Cần duy trì kết nối TCP liên tục. Nếu rớt mạng, phải thiết lập lại handshake HTTP tốn kém tài nguyên. Không có cơ chế lưu tin nhắn khi mất mạng (QoS).
- **MQTT (Message Queuing Telemetry Transport):**
    - **Lightweight:** Header chỉ 2 byte, tối ưu cho mạng yếu.
    - **QoS (Quality of Service):** MQTT hỗ trợ QoS 1 (At least once) và QoS 2 (Exactly once). Khi Server gửi lệnh "Bật Camera" với QoS 1, nếu Camera đang mất mạng, Broker sẽ lưu lệnh đó lại và gửi ngay khi Camera có mạng trở lại.
    - **Last Will and Testament (LWT):** Tính năng "Di chúc". Nếu Camera đột ngột mất điện, Broker sẽ tự động gửi tin nhắn "Camera Offline" đến Server. Server lập tức cập nhật App người dùng để họ không chờ đợi vô vọng.

Kết Luận: Sử dụng **MQTT over TLS (MQTTS)** là giải pháp bắt buộc cho tính ổn định của hệ thống Camera IoT tại sân.

2.4. Trí Tuệ Nhân Tạo Tại Biên (Edge AI): Tự Động Bắt Highlight

Gửi toàn bộ video 90 phút (dung lượng vài GB) lên Cloud để xử lý là "tự sát" về chi phí băng thông và lưu trữ. Giải pháp là xử lý ngay tại camera (Edge Computing). Camera cần kết nối với một thiết bị tính toán nhỏ (ví dụ: Jetson Nano).

2.4.1. Pipeline Xử Lý Video (Computer Vision Pipeline)

Hệ thống AI cần trả lời hai câu hỏi: "Bóng đang ở đâu?" và "Hành động gì vừa xảy ra?".

1. **Tracking (Theo Dõi Bóng):**
    - Sử dụng mô hình **TrackNetV2**. Đây là mô hình chuyên biệt cho tennis/cầu lông, sử dụng input là chuỗi 3 khung hình liên tiếp để dự đoán vị trí bóng (kể cả khi bóng bị mờ do chuyển động nhanh - motion blur). TrackNetV2 xuất ra Heatmap vị trí bóng chính xác hơn nhiều so với YOLO thông thường đối với vật thể nhỏ.
2. **Action Recognition (Nhận Diện Hành Động):**
    - Sử dụng mô hình **MoViNet (Mobile Video Networks)**. Đây là dòng mô hình của Google thiết kế riêng cho thiết bị di động/biên.
    - **Cơ chế Stream Buffering:** MoViNet có thể xử lý video theo kiểu streaming (từng frame một) mà không cần chờ cả video, giúp giảm độ trễ bộ nhớ đệm, phù hợp hoàn hảo cho phần cứng yếu tại biên.

3. **Hợp Nhất Đa Phương Thức (Multimodal Fusion):**
    - AI thị giác có thể nhầm lẫn. Kết hợp với **Phân Tích Âm Thanh (Audio Analysis)**.
    - Logic: Nếu (TrackNet phát hiện bóng đi nhanh) VÀ (MoViNet phát hiện dáng người đập bóng) VÀ (Audio Sensor phát hiện âm lượng > 80dB - tiếng đập vợt) => **Highlight 100%**.

2.4.2. Tối Ưu Hóa "Gửi Lại Video"

Thay vì gửi video gốc, thiết bị Edge sẽ:

1. Cắt clip highlight (ví dụ: từ -5s đến +5s quanh sự kiện).
2. Transcode clip này về định dạng MP4 (H.264) nhẹ.
3. Upload chỉ các clip này lên Cloud Storage (S3/MinIO).
4. Gửi link video về App người dùng qua MQTT hoặc Push Notification. Chiến lược này giảm băng thông tiêu thụ từ hàng GB xuống còn vài chục MB mỗi trận, giải quyết bài toán chi phí vận hành.

Phần III: Kết Luận và Kiến Nghị Triển Khai

Để chuyển hóa ý tưởng thành sản phẩm thực tế, đội ngũ cần tuân thủ nghiêm ngặt các nguyên tắc kỹ thuật đã phân tích:

1. **Về Mobile App (Self-Recording):**
    - Tuyệt đối không dùng `MediaRecorder` cho tính năng highlight. Hãy đầu tư thời gian xây dựng lớp **Circular Buffer** trên nền `MediaCodec` (Android) và `AVAssetWriter` (iOS). Đây là nền móng cho sự ổn định.
    - Sử dụng **TensorFlow Lite** với mô hình DS-CNN huấn luyện riêng cho từ khóa "Highlight" để đảm bảo nhận diện tốt trong môi trường ồn và không làm nóng máy.
    - Thiết kế UI/UX với tư duy "Distant Interaction": Phản hồi bằng âm thanh và ánh sáng màn hình là bắt buộc.
2. **Về Hệ Thống Sân (Venue Recording):**
    - Triển khai **MQTT** thay vì WebSockets để đảm bảo tính ổn định kết nối IoT.
    - Bảo mật Webhook thanh toán bằng **HMAC Signature** để tránh gian lận thương mại.
    - Áp dụng mô hình **Edge AI (TrackNetV2 + MoViNet)** để xử lý video tại nguồn, chỉ gửi dữ liệu highlight lên Cloud để tối ưu chi phí.

Bản thiết kế này không chỉ giải quyết các khó khăn hiện tại về luồng xử lý và giao diện, mà còn đặt nền móng cho một hệ thống có khả năng mở rộng (scalable) lên hàng nghìn sân bãi và hàng triệu người dùng trong tương lai.

en.wikipedia.org
Circular buffer - Wikipedia
Opens in a new window

medium.com
A Practical Guide to Implementing a Generic Ring Buffer in Go - Medium
Opens in a new window

algocademy.com
When to Consider Using a Circular Buffer: A Comprehensive Guide - AlgoCademy
Opens in a new window

stackoverflow.com
ios endless video recording - cocoa touch - Stack Overflow
Opens in a new window

developer.tuya.com
Audio and Video Ring Buffer-TuyaOS-Tuya Developer
Opens in a new window

github.com
grafika/app/src/main/java/com/android/grafika/CircularEncoderBuffer.java at master - GitHub
Opens in a new window

github.com
ArtisteHsu/SampleMediaCodec: Android video decoder example by using MediaCodec class - GitHub
Opens in a new window

github.com
Using a ring/circular buffer for recording · Issue #952 · pedroSG94/RootEncoder - GitHub

Opens in a new window

stackoverflow.com

Circular buffer implementation in Android - java - Stack Overflow

Opens in a new window

gist.github.com

Recording video with AVAssetWriter - GitHub Gist

Opens in a new window

gist.github.com

Process a CMSampleBuffer to modify the audio using an AVAssetReader and AVAssetWriter in Swift - GitHub Gist

Opens in a new window

stackoverflow.com

How would I put together a video using the AVAssetWriter in swift? - Stack Overflow

Opens in a new window

stackoverflow.com

Is it possible to use a circular buffer for video frames on iOS? - Stack Overflow

Opens in a new window

youtube.com

Recording Video with AVAssetWriter - CueCam Devlog 8 - YouTube

Opens in a new window

research.google.com

small-footprint keyword spotting using deep neural networks - Google Research

Opens in a new window

arxiv.org

Advances in Small-Footprint Keyword Spotting: A Comprehensive Review of Efficient Models and Algorithms - arXiv

Opens in a new window

alphacephei.com
VOSK Offline Speech Recognition API - Alpha Cephei
Opens in a new window

preprints.org
Comparative Analysis of Vosk Toolkit and Other Speech Recognition Frameworks for Custom Language Model Implementation - Preprints.org
Opens in a new window

firebase.google.com
Add on-device Text Classification to your app with TensorFlow Lite and Firebase - Android Codelab
Opens in a new window

wiki.seeedstudio.com
Keyword Spotting with TensorFlow Lite | Seeed Studio Wiki
Opens in a new window

stackoverflow.com
Android Video Circular Buffer with Sound - audio - Stack Overflow
Opens in a new window

sportfitnessapps.com
How to Design User-Friendly Sports App Interfaces - 2V Modules
Opens in a new window

unifiedinfotech.net
Voice User Interface (VUI) for Mobile Apps - A Complete Guide - Unified Infotech
Opens in a new window

aufaitux.com
Voice User Interface Design Best Practices | Top 10 VUI Tips - Aufait UX
Opens in a new window

medium.com

8 Best Practices When Designing Mobile App Onboarding Flow | by Queble - Medium

Opens in a new window

img.ly

Designing A Timeline For Mobile Video Editing - IMG.LY

Opens in a new window

play.google.com

VN - AI Video Editor - Apps on Google Play

Opens in a new window

researchgate.net

A Review Paper on Development of an IoT-Based QR Code Access Control and Payment System using Arduino and ESP8266 - ResearchGate

Opens in a new window

repository.arizona.edu

PAYMENT TRANSACTION SYSTEM USING QR CODES By MUNEEB MATEEN AHMED - The University of Arizona

Opens in a new window

hookdeck.com

Complete Guide to Webhook Security - Hookdeck

Opens in a new window

hookdeck.com

How to Implement SHA256 Webhook Signature Verification - Hookdeck

Opens in a new window

blog.dreamfactory.com

Webhook Triggers for Event-Driven APIs - DreamFactory Blog

Opens in a new window

hivemq.com

Understanding the Differences between MQTT and WebSockets for IoT - HiveMQ

Opens in a new window

ably.com

MQTT vs WebSocket - Which protocol to use when in 2024 - Ably

Opens in a new window

aboutiiot.com

IoT Protocols: MQTT vs. HTTP vs. WebSocket - My Store

Opens in a new window

hivemq.com

MQTT Security Fundamentals: How to secure MQTT in IoT - HiveMQ

Opens in a new window

docs.aws.amazon.com

MQTT design best practices - Designing MQTT Topics for AWS IoT Core

Opens in a new window

datafloq.com

Opens in a new window

bcdvideo.com

Edge vs. Cloud Processing for Video Analytics | BCD

Opens in a new window

abcloudz.com

Real-Time Object Detection and Classification in Sports Using AI and Computer Vision

Opens in a new window

github.com

NanoDet-Plus Super fast and lightweight anchor-free object detection model. Only 980 KB(int8) / 1.8MB (fp16) and run 97FPS on cellphone - GitHub

Opens in a new window

github.com

Unofficial PyTorch implementation of TrackNet - GitHub

Opens in a new window

gitlab.nol.cs.nycu.edu.tw

open-source / TrackNetV2 - GitLab

Opens in a new window

analyticsvidhya.com

Exploring MoViNets: Efficient Mobile Video Recognition - Analytics Vidhya

Opens in a new window

github.com

WavesUR/embedded_TSM: cs231n project - GitHub

Opens in a new window

pmc.ncbi.nlm.nih.gov

Automatic summarization of soccer highlights using audio-visual descriptors - PMC - NIH

Opens in a new window

source.android.com

SurfaceTexture | Android Open Source Project

Opens in a new window

qcall.ai

Speech To Text Open Source: 21 Best Projects 2025

Opens in a new window

invicti.com

Webhook Security Best Practices and Checklist - Invicti

Opens in a new window

inventivehq.com

Webhook Signature Verification: Complete Security Guide - Inventive HQ

Opens in a new window

tensorflow.org

MoViNet for streaming action recognition | TensorFlow Hub

Opens in a new window