Interview Task — Real-time WebRTC VLM Multi-Object Detection (Phone → Browser → Inference → Overlay)

One-line goal: Build a reproducible demo that performs **real-time multi-object detection** on live video streamed from a phone via **WebRTC**, returns detection bounding boxes + labels to the browser, overlays them in near real-time, and deliver a **1-minute Loom video** showing the live demo, metrics, and one-sentence tradeoffs.

Deliverables (exact)

- Git repo (frontend + optional server) with Dockerfile(s) and docker-compose.yml for local run. Include start.shconvenience script.

- README.md with one-command start instructions and mode-switch (server-mode / wasm-mode). Clear phone-join instructions (QR or short URL).

- metrics.json produced by a short bench run (30s) listing median & P95 end-to-end latency, processed FPS, and uplink/downlink kbps.

- A **1-minute Loom** video (hosted link) that: (a) shows phone → browser live overlay, (b) shows metrics output briefly, and (c) one-line improvement you'd do next.

- Short report (README appendix or report.md, 1 page) explaining design choices, low-resource mode, and backpressure policy.

Non-functional constraints & fairness

- **Low-resource path required.** Candidate must provide a mode runnable on modest laptops (no GPU). Typical approaches: WASM on-device inference (onnxruntime-web or tfjs-wasm), quantized small models, downscale input to 320×240, and adaptive sampling (10–15 FPS).

Real-time from a phone. Phone must use only a browser (Chrome on Android, Safari on iOS) to connect — no custom native app requirement.

One command to start. Candidate should supply `docker-compose up` or `./start.sh` to launch demo locally.

Minimal acceptance criteria (pass/fail)

Phone can connect via QR/URL and stream live camera to the demo. Browser shows live overlays of bounding boxes aligned to frames.

`metrics.json` with median & P95 latency and FPS exists.

README explains how to run both low-resource and server modes.

Loom video demonstrates the live phone stream and metrics clearly within 1 minute.

UX / API contract (frame alignment)

Use this JSON message per frame over DataChannel / WebSocket for detection results (server → client):

```
{
  "frame_id": "string_or_int",
  "capture_ts": 1690000000000,
  "recv_ts": 1690000000100,
  "inference_ts": 1690000000120,
  "detections": [
    { "label": "person", "score": 0.93, "xmin": 0.12, "ymin": 0.08, "xmax": 0.34, "ymax": 0.67 }
  ]
}
```

Coordinates normalized [0..1] to simplify overlay across resolutions.

Browser uses capture_ts and frame_id to align overlays with the correct frame and compute E2E latency.

Measurement & bench instructions

E2E latency (per frame): overlay_display_ts - capture_ts → report median & P95 over a 30s run.

Server latency: inference_ts - recv_ts.

Network latency: recv_ts - capture_ts.

Processed FPS: count of frames with detections displayed / seconds.

Bandwidth: estimate via browser network inspector or tools like ifstat/nethogs during run.

Provide a simple bench script ./bench/run_bench.sh --duration 30 --mode server that outputs metrics.json.

Low-resource guidance (what candidate *must* provide)

WASM on-device mode using onnxruntime-web or tfjs-wasm with a small quantized model (example: MobileNet-SSD or YOLOv5n quantized).

Downscale: default input size 320×240 and target processing 10–15 FPS.

Frame thinning: process only latest frames; maintain a fixed-length queue and drop old frames when overloaded.

- **Simple mode switch**: MODE=wasm vs MODE=server in start.sh.

Candidates must document CPU usage on a modest laptop (e.g., Intel i5, 8GB RAM) for both modes.

Suggested technology & third-party components (install on dev machine OR phone)

For phone (user-facing, minimal required)

- **Chrome (Android)** — recommended: stable Chrome app.

- **Safari (iOS)** — iOS Safari supports WebRTC but feature parity varies; recommend latest iOS.

- No app installs required — phone uses browser to open a QR/URL and stream.

Optional phone tools (only if candidate documents and uses them):

- **ngrok** or **localtunnel** — for exposing localhost to the phone if Wi-Fi NAT blocks direct connect (candidate must include free-tier instructions).

- **Termux (Android)** — optional for advanced phone-side testing (not required).

For dev laptop / server (recommended installs)

- **Docker & Docker Compose** — recommended for reproducible local environment.

- **Node.js (>=16)** — for frontend dev server and lightweight WebRTC gateway if used.

- **Python 3.9+** — if using aiortc or server-side Python inference.

ONNX Runtime: onnxruntime (CPU) for server-mode; onnxruntime-web for browser WASM.

tfjs (optional) — @tensorflow/tfjs or tfjs-backend-wasm for JS inference.

aiortc (Python) or **pion** (Go) or **mediasoup** (Node) — pick any for a gateway that can receive WebRTC tracks.

ngrok (optional) — for quick phone connectivity.

Model & assets

ONNX Model Zoo — MobileNet-SSD, YOLO variants, or quantized models.

TensorFlow Lite models — if using TF.js or tflite-web.

Tools for measurement & debugging

Chrome DevTools (webrtc-internals) — inspect RTP stats.

getStats() WebRTC API — for per-RTCPeerConnection metrics.

ifstat / iftop / nethogs — bandwidth during run.

ps/top/htop — CPU & memory.

tc (linux) — simulate packet loss/latency for robustness tests (optional).

Step-by-step candidate run instructions (to include in README)

git clone <repo>

./start.sh (defaults to MODE=wasm if no GPU) or docker-compose up --build

Open http://localhost:3000 on your laptop; scan displayed QR with your phone.

Allow camera on phone; you should see phone video mirrored on the laptop with overlays.

Run ./bench/run_bench.sh --duration 30 --mode wasm to collect metrics; inspect metrics.json.

If phone cannot reach laptop directly: run ./start.sh --ngrok to start ngrok and copy the public URL to the phone.

Troubleshooting tips (include these in README)

If phone won't connect: ensure phone and laptop are on same network OR use ngrok/localtunnel.

If overlays are misaligned: confirm timestamps (capture_ts) are being echoed and units match (ms).

If CPU is high: reduce resolution to 320×240 or offload to WASM mode.

Use Chrome webrtc-internals to inspect packet send/receive times and jitter.

Quick evaluation rubric (one-liner)

Functionality (30%): phone stream + overlays + metrics exist.

Latency (25%): median & p95 E2E latency are sensible for chosen mode.

Robustness (15%): queue/drop/backpressure strategy & low-resource mode.

Docs & reproducibility (15%): clear README + docker-compose + 1-min Loom.

Design reasoning (15%): tradeoffs and improvement plan.