# Deepfake Detection Chrome Extension: Full Roadmap

## 1) Define Scope and Success Criteria

- **Target Sites**: Start with all sites via tab capture; optimize later for YouTube or TikTok if needed.
- **Signals Used**: Begin with vision-only (face-artifact detection). Add lip-sync and rPPG later.
- **Success Metrics**: Aim for ≥80% AUC, real-time alerts at 1–2 inferences/sec, banner updates every 3s.

## 2) Gather Evaluation Clips

- Collect 20–50 short clips (real/fake, different resolutions, formats).
- Store labels in a simple JSON file for evaluation.

## 3) Choose Initial Model (Vision-Only)

- Use a small CNN/ViT (MobileNetV3-Small, EfficientNet-Lite, ViT-Tiny).
- Export to **ONNX**; quantize (fp16/int8) for WebGPU and WASM.
- Input: 160–224 px crops. Output: 0–1 fake probability.

## 4) Runtime Stack

- **ONNX Runtime Web** for inference (WebGPU/WASM backends).
- **Face Detection** via MediaPipe or lightweight WASM model.
- Use **MediaStreamTrackProcessor** for frame access.

## 5) Extension Architecture (MV3)

- **Service Worker**: Manages offscreen document.
- **Offscreen Document**: Runs model inference and handles streams.
- **Content Script**: Displays UI overlay and handles user input.
- **Action Button**: Starts tab capture.
- **IndexedDB**: Caches models and kernels.

## 6) Capture → Process → Display

- Capture video/audio with `chrome.tabCapture`.
- Process 1–3 frames/sec for faces; crop, align, normalize.
- Run inference; smooth results with EMA.
- Overlay risk score via content script UI.

## 7) Add Audio-Visual Lip-Sync (Phase 2)

- Extract audio (WebAudio API) and video (mouth crops).
- Use a lightweight A/V sync model (SyncNet-style) via ONNX.
- Fuse vision and audio results for improved reliability.

## 8) Add Physiology (rPPG) Signal (Phase 3)

- Track skin patches; analyze color changes for blood flow signals.
- Band-pass filter 0.7–4 Hz; compute plausibility score.
- Add as a weak feature in score fusion.

## 9) Score Fusion and Calibration

- Combine vision, audio, rPPG scores (logistic regression).
- Define color-coded thresholds:
- Low: 0–0.4 (Green)
- Medium: 0.4–0.7 (Amber)
- High: 0.7–1.0 (Red)
- Add hysteresis to avoid flickering UI.

## 10) Performance Optimization

- Prioritize WebGPU; fallback to WASM with SIMD.
- Keep model <25 MB after quantization.
- Sparse sampling; batch multiple faces.
- Use Web Workers for off-main-thread inference.

## 11) Privacy, Security, and UX

- All inference on-device.
- Show disclaimer: "This is a risk signal, not proof."
- Allow per-site toggle and global disable switch.
- Store no frames unless explicitly allowed.

## 12) Developer Setup

- Local test grid page with labeled videos.
- Log FPS, latency, and backend used.
- Build in live threshold tuning panel.

## 13) Packaging for Chrome Web Store

- Minimize permissions; justify `tabCapture`.
- Include privacy policy (no data leaves device).
- Add accessibility features (keyboard toggles).

## 14) Post-MVP Roadmap

- Add provenance (C2PA) signature checks.
- Implement creator identity whitelist.
- Support model update via IndexedDB.
- Handle edge cases (filters, slideshows, gameplay).

## 15) Implementation Checklist

1. Initialize repo with extension boilerplate.
2. Write manifest.json with permissions.
3. Create service worker + offscreen doc.
4. Implement tab capture.
5. Process video frames with MediaStreamTrackProcessor.
6. Integrate ONNX Runtime Web.
7. Build face detector pipeline.
8. Smooth scores with EMA.
9. Add content overlay UI.
10. Test and tune latency.
11. Add lip-sync and rPPG later.
12. Add privacy policy and publish.