ChatGPT

魔法の杖を用いた WebAR アプリケーション – 要件定義書

作成日: 2025-06-13

概要

本書は、カメラ映像上で**魔法の杖を使ったAR体験**を提供するWebアプリケーションの要件定義書です。スマートフォンまたはPCのブラウザ上のみで動作し（インストール不要）、リアルタイム物体検出とジェスチャ認識によってユーザの杖の動きを検知し、対応する魔法効果を3Dエフェクトとして映像に重ねて表示します。選定する技術はいずれも最新のオープンソースソフトウェアを用い、誰でも再現・構築可能な実装を目指します。以下、機能要件と制約条件を整理した上で、システム構成、使用技術とライブラリ、依存関係、実行環境、対応ブラウザ、およびOSSライセンスについて詳細に記載します。

機能要件

1. **カメラ映像の取得・表示**: ユーザのスマートフォンまたはPCのカメラから映像を取得し、ブラウザ画面上にリアルタイム表示する。映像は鏡のようにユーザにプレビューされ、インカメラ・背面カメラの切替にも対応する。
2. **杖のリアルタイム検出と先端追跡**: カメラ映像中に映る**棒状の杖**オブジェクトをリアルタイムに検出する。最低限、検出した杖の**先端座標**（画面座標上の (x, y)）を各フレームで特定して追跡する。また必要に応じて杖の向きや長さも把握し、ノイズを低減する。
3. **軌跡パターン照合による呪文識別**: 杖先端の移動軌跡を時系列で解析し、あらかじめ登録された複数種類（**8種類以上**）の軌跡パターンと照合することで、ユーザが空中に描いた形を**呪文**として識別する。認識結果は該当する呪文の名前またはIDとして内部で扱い、必要に応じてユーザにフィードバックする。
4. **3Dアニメーションエフェクトの重畳表示**: 認識された呪文に対応する**魔法効果の3Dアニメーション**を即座に生成し、現在のカメラ映像にオーバーレイ表示する。例えば、杖先端から火花や光線が放たれるエフェクトや、画面上のオブジェクトに変化が生じるエフェクト等を、現実の映像と重ね合わせて表示する。エフェクトの種類は少なくとも8種類（各呪文に対応）用意し、ユーザの操作にリアルタイムに反応する。

制約条件（非機能要件）

1. **ブラウザベースでの実現**: 本アプリケーションは**Webアプリケーションのみ**で完結し、ネイティブアプリのインストールを不要とします。最新のChrome, Safari, Firefox, Edgeといった主要ブラウザ上で動作し、HTTPS下でのカメラ使用許可のもと実行されます（ブラウザ機能の制約上、カメラアクセスは安全なコンテキスト(HTTPS)でのみ可能 [1] ）。サーバサイドは不要で、オフライン環境でも動作可能です。
2. **オープンソース技術の利用**: 使用するライブラリ・フレームワーク・モデルは**すべてOSS（オープンソースソフトウェア）**であり、そのライセンスを明記します。商用の閉源サービスや有償SDKには依存しません。誰もがリポジトリから取得・ビルドできる技術のみを採用します。
3. **使用ライブラリごとの情報開示**: 採用するすべてのライブラリについて、**(1) ライブラリ名、(2) 最新の公式URL（GitHub等、直近1年以内に更新されているもの）, (3) 主な依存関係一覧**を本書にて明記し

ます。また各ライブラリの依存関係がすべて解決可能である根拠（npm経由でビルド可能・競合なし等）も示します。

4. **機械学習モデル推論方式**: 杖検出や姿勢推定に用いる機械学習モデルのブラウザ内推論には、**ONNX Runtime Web**等の実績あるランタイムを使用します。Web環境で確実に動作した実績があり、十分な性能が出るものを選定します（例: ONNX Runtime Web はWebAssemblyとWebGLによる最適化された推論を提供し、ブラウザで直接ONNXモデルを実行可能 [2] ）。TensorFlow.jsなど他のOSSも検討しましたが、ONNX形式への統一と実行性能を優先します。
5. **高速化技術**: クライアント側推論の高速化のため、**WebAssembly + マルチスレッド/ SIMD命令**や、対応ブラウザでは**WebGPU**によるGPUアクセラレーションを活用します。ONNX Runtime WebはCPU向けにWebAssemblyバックエンド（マルチスレッド+SIMD対応）を備え、GPU向けに実験的なWebGPUバックエンドも公式サポートしています [3] [4] 。実際、WebGPUとWebAssemblyを併用することでブラウザ推論が従来比20倍以上高速化した事例も報告されています [5] 。SIMD対応や並列化により、リアルタイム処理（目標フレームレート30FPS程度）を達成できる構成とします。
6. **マルチプラットフォーム対応**: モダンブラウザ（Chrome, Edge, Firefox, Safari）の最新版にて動作確認を行い、それぞれのエンジン特有の差異に対処します。カメラ利用のMediaDevices APIは各ブラウザで2017年以降広くサポートされています [6] 。推論バックエンドは、Chrome/EdgeではWebGPUが利用可能（Chrome/Edge 113+で標準有効 [7] ）、SafariやFirefoxでは現時点でWebGPU非対応のためWebAssembly（CPU）やWebGLにフォールバックします [8] 。特にSafari（iOS含む）はWebAssemblyマルチスレッド実行に制約があるため、必要に応じてWebGLバックエンド（GPU処理）を代替で使用し安定した速度を確保します。なお、本アプリはデスクトップ・モバイル双方のブラウザで動作しますが、モバイルSafariについてはARカメラ利用時のパフォーマンスに留意します。
7. **カメラアクセス方法**: カメラ映像の取得には**WebRTC**もしくは**MediaDevices API**（`navigator.mediaDevices.getUserMedia`）を使用します。特にMediaDevices.getUserMedia()はプロミスベースでシンプルにカメラ/マイク入力を取得でき、ユーザの許可を得て動画ストリーム（MediaStream）を提供します [1] 。このAPIはChrome・Firefox・Safari・Edgeの全てで利用可能であり（要HTTPS接続）、取得したMediaStreamを映像要素やCanvasに描画することでカメラプレビュー表示を行います。音声は今回は使用しないため、constraintsでvideoのみに限定します。
8. **3Dエフェクト描画方法**: 魔法エフェクトの描画には**Three.js**や**React Three Fiber**（Three.jsのReactバインディング）等のWebGLフレームワークを使用します。Three.jsは実績豊富なオープンソース3Dライブラリであり、軽量かつクロスブラウザな高水準APIを提供します [9] 。WebGL2に加えて一部機能でWebGPUも利用可能な最新バージョンを採用し、ブラウザ上で高度な3D表現を可能にします。魔法エフェクト（粒子、光の軌跡、爆発など）はThree.js上でシェーダやパーティクルシステムを用いて実装し、カメラ映像と合成します（後述）。なおUI部分でReactを使用する場合はReact Three Fiber (R3F)＋dreiライブラリでThree.jsを扱うことも検討しますが、いずれもMITライセンスのOSSです。
9. **機械学習モデルの再学習**: 杖検出などに用いるAIモデルについては、COCOデータセットで事前学習済みのモデルをベースに**転移学習（ファインチューニング）**を行い、新たに「魔法の杖」クラスに対応させても構いません。本要件では独自にデータ収集した「杖」の画像データセットを用意し、既存の物体検出モデルを再学習して精度向上を図ることを許容します。例えばYOLOシリーズのモデルはMS COCOで学習済みの重みを利用してカスタムデータで再学習可能であり [10] 、PyTorchベースのトレーニング手順やUltralytics社のガイド [11] に従ってONNX形式のモデルを導出できます。この再学習に使用するツールやスクリプトもOSS（例: Ultralytics YOLOv5/YOLOv8，Hubなど）を利用し、その公式ドキュメントURLを本書で提示します。モデル精度次第ではOpenCVやTensorFlowによる追加処理を検討する場合もありますが、いずれもライセンスを確認します。

システム構成

本章では、上記要件を満たすためのシステム構成を述べます。本アプリは**ブラウザ内で完結するクライアントサイドWebアプリケーション**であり、大きく「カメラ入力」「杖検出（CV推論）」「軌跡認識（パターン

マッチ）」「3D描画」の4つのコンポーネントに分かれます。それぞれのデータフローと処理内容を以下に示します。

- **①カメラ入力モジュール**: ユーザ端末のカメラ映像を取得する。MediaDevices.getUserMedia()で動画ストリーム（MediaStream）を取得し、HTML `<video>` 要素または `<canvas>` 要素上にリアルタイム描画する。以降の画像処理のため、各フレームのピクセルデータを取得（Canvasの2Dコンテキストや OffscreenCanvasの利用も検討）。このモジュールはブラウザの権限管理によりユーザ許可を必要とする。

- **②杖検出モジュール**: カメラ映像から**魔法の杖**を検出するコンピュータビジョン処理。ここで言う杖とは棒状の物体であり、一般的な背景中の棒状物との識別、およびユーザが手に持っている場合の手との関連性を考慮します。物体検出にはYOLO系の深層学習モデルを採用します。具体的には最新の**YOLOv10**モデルを小型化した**YOLOv10-S**を使用し、「wand（杖）」クラスにファインチューニングした独自モデルをONNX形式で用意します。YOLOv10は2024年に提案された最先端の物体検出モデルで、NMS（非最大抑制）を不要とする構造改善により高精度・高速な検出を実現しています [12]。特に小型版のYOLOv10-SはCOCOデータセットで45.2 APを達成しつつリアルタイム性能を備えており、同程度の精度を持つTransformerベースのモデル（例: RT-DETR）よりも推論レイテンシが低いためブラウザ実行に適します [12]。このモデルをONNX Runtime Webで実行し、各フレームの画像から杖のバウンディングボックスと先端位置を推定します。先端座標は、検出したバウンディングボックス内で幾何学的に先端に相当する点（例えば長軸に沿った一端）を計算して得ます。検出処理は30FPS程度で連続実行し、結果として時系列の先端座標ストリームを生成します。なお検出精度向上のため、**MediaPipe Hands**（手指ランドマーク検出）を併用し、杖が握られている手の位置を検知して杖候補の物体にフィルタをかけることも検討します。MediaPipe HandsはGoogle提供の軽量モデルで、手の21関節ランドマークをリアルタイム検出可能です（Apache2.0ライセンス）。手の存在と向きを参考情報として用いることで、誤検出の低減や、杖先端が手によって隠れた際の補間に役立てます。

- **③軌跡パターン認識モジュール**: 杖検出モジュールから取得した**杖先端の軌跡データ**（時系列の座標列）を蓄積・解析し、事前定義された**呪文ジェスチャ**パターンと照合します。ユーザが空中に描くジェスチャは多少の大きさ・回転・描画速度の違いがあるため、単純な座標の比較ではなく正規化とパターンマッチングを行います。本システムでは、インクリメンタルに軌跡をサンプリングしつつ**$1 ユニストローク認識アルゴリズム**（One Dollar Recognizer）をベースに実装します [13]。$1アルゴリズムはプロトタイピング向けに考案されたシンプルなジェスチャ認識手法で、平面上の軌跡を位置・サイズ・回転を正規化してからテンプレートと比較し、ユーザ定義パターンを高精度（1パターンあたり1つのサンプルで約97%の認識率 [14] ）で識別できます。具体的には、各呪文ジェスチャごとに代表的な軌跡（例: 炎の呪文=「炎」を描くような軌跡、水の呪文=波線軌跡 など）をテンプレートとして登録し、ユーザの軌跡とダイナミックタイムワーピング(DTW)等で距離計算し最も近いテンプレートを認識結果とします [15]。比較中はKalmanフィルタで滑らかにした直近数秒分の軌跡を用い、ジェスチャが完了したタイミング（一定時間先端が停止する、特定動作で終了判定する等）で認識を確定します。認識した呪文IDは直ちに次のエフェクト描画モジュールへ渡されます。判別精度目標は少なくとも識別対象8種について85%以上とし、類似した軌跡パターン間でも誤認を最小化します。必要に応じてSVMなど機械学習分類器で特徴量から判定するアプローチも検討しますが、今回は実装容易性と説明可能性からテンプレートマッチング方式を採用します。

- **④3Dエフェクト描画モジュール**: 認識された呪文に応じて、対応する**3Dアニメーション効果**をカメラ映像に重畳表示します。Three.jsを用いてWebGL上にパーティクルや光のシェーダー効果を描画し、現実映像と合成します。構成としては、HTML上で映像を表示する `<video>` 要素の上に `<canvas webgl>` 要素を重ね、Three.jsのシーンを合成描画する形を取ります（あるいはThree.jsの背景テクスチャにカメラ映像を張り込む方法もあります）。各呪文ごとに異なるエフェクト（例: 炎の呪文なら火花の粒子、氷の呪文なら氷の破片と霜エフェクトなど）を事前にThree.jsで実装しておき、呪文IDに応じて対応するエフェクトオブジェクトを生成・再生します。エフェクトの位置は基本的に**杖の先端位置**に一致させ、先端から放出されるように見せます。例えばパーティクルシステムを先端座標に

配置し放射状にパーティクルを飛ばす、レーザー光線の場合は先端から所定方向に伸びるラインを描画する、といった具合です。Three.jsの**PostProcessing**機能（@react-three/postprocessing あるいは原生のEffectComposer）を用いてグロー（発光）やモーションブラー等の効果を付与し、魔法らしい演出を高めます。描画更新はrequestAnimationFrameで行い、カメラ映像のフレームに同期して滑らかにエフェクトが表示されるようにします。目標として、**呪文認識が確定してから100ms以内**にエフェクト描画を開始し [16]、ユーザに違和感を与えないリアルタイム性を確保します。またエフェクト描画中も杖検出と軌跡取得は継続し、連続して呪文を放てるようにします（並列処理や前回エフェクトのフェードアウト管理等を実装）。エフェクト再生が完了したらThree.js上のオブジェクトをクリーンアップし、次の呪文発動に備えます。なお描画負荷が高い場合は、パーティクル数やエフェクト持続時間を端末性能に応じて調整し、最低25FPS以上のレンダリングを維持します。

以上が主要なコンポーネント構成です。補足として、UI/UX向上のため**ミラーモード切替**（カメラ映像を左右反転表示する機能）や**ガイド表示**（認識した軌跡をライン表示、成功率のフィードバック等）も実装可能ですが、本要件の中心ではないため割愛します。またデータ永続化が必要な場合、ブラウザのIndexedDBにユーザの練習履歴や成功率を保存することで個人ごとの分析も可能です。これらは付加機能として位置付け、必要に応じて追加要件とします。

使用技術・ライブラリ一覧

本システムで採用する主な技術要素とライブラリ、およびその公式リソースや依存関係について整理します。全て最新のOSSであり、2024–2025年に更新が行われているプロジェクトです。

- **ONNX Runtime Web** (JavaScript版 ONNX Runtime)
- **公式**: GitHubリポジトリ「microsoft/onnxruntime」内のオンnxruntime-webパッケージ、およびnpm: `onnxruntime-web` [17]。直近の最新版は1.22.0（2025年5月公開）で活発にメンテナンスされています [18] [19]。
- **概要**: ブラウザ上でONNXフォーマットの機械学習モデルを実行するための高性能ランタイムライブラリです。WebAssemblyによるネイティブ速度の推論とWebGL/WebGPUによるハードウェアアクセラレーションを組み合わせ、ブラウザ内でのONNX推論を可能にします [2]。これによりサーバとの通信なしでモデル推論ができ、プライバシーやオフライン動作の観点でも有利です [20]。ONNX Runtime Webはマルチスレッド・SIMD対応のWASMバックエンドを備え、ほぼすべてのONNX演算子をサポートしています [2]。さらにv1.7.0以降では実験的にWebGPUにも対応し、対応ブラウザではGPUを用いた高速推論が可能です [3]。

- **依存関係**: npmパッケージとして提供されており、内部でいくつかのユーティリティライブラリに依存します（2025年現在 **6件**: `flatbuffers`, `guid-typescript`, `long`, `onnxruntime-common`, `platform`, `protobufjs` [21]）。これらはいずれもnpm上で最新版が入手可能であり、`npm install onnxruntime-web` 実行時に自動解決・インストールされます [22]。依存ライブラリもOSS（flatbuffersやprotobufjsはGoogle等がApache/MITライセンスで公開）であり、本プロジェクトに取り込んでもライセンス上問題ありません。

- **YOLOv10-S オブジェクト検出モデル**

- **公式**: GitHub: [THU-MIG/YOLOv10](#)（NeurIPS 2024論文実装）, Ultralytics社の解説ブログ、arXiv論文 [23] [12] など。2024年5月末に公開され、以降コミュニティで継続開発されています。
- **概要**: YOLOv10はリアルタイム物体検出モデルYOLOシリーズの最新世代で、モデルアーキテクチャの包括的最適化とNMS除去によるエンドツーエンドの高速化が特徴です [12]。特に**YOLOv10-S**はパラメータ数が小さい軽量版でありながらCOCOベンチマークで最先端の精度と高速性を両立しています [12]。本プロジェクトではYOLOv10-Sをベースに**魔法の杖検出用に再学習**したモデルを使用します。

モデルはPyTorchで訓練後、ONNX形式にエクスポートし、ブラウザでONNX Runtime Webにより実行します（エクスポートと最適化手順はUltralytics社のガイドに従います [10]）。推論時にはWASM+SIMD (Safari等CPU実行) またはWebGPU (Chrome/Edge GPU実行) で高速動作し、リアルタイム検出を実現します。なおYOLOv10のコード自体はAGPL-3.0ライセンスですが、ブラウザにはモデル重みとONNXランタイムのみ組み込まれ、推論コード（ONNX Runtime部分）はMITライセンスのためクライアント配布上のライセンス衝突もありません。

- **依存関係**: モデル自体はデータ（ニューラルネットワークの重みファイル）であり、実行には上記のONNX Runtime Webが必要です。追加のJSライブラリ依存はありません。学習に使用したPyTorchやUltralytics YOLOツールは開発環境のみの依存であり、本番Webアプリには含めません。

- **MediaPipe Hands（オプション）**

- **公式**: Google MediaPipe Solutions Hands モジュール – GitHub: google/mediapipe, Web APIドキュメント: MediaPipe Hand Landmarker 等。最新バージョンはMediaPipe Solutions v0.10+ (2023年更新)。
- **概要**: MediaPipe Handsは片手あたり21個の手指関節ランドマークをリアルタイムに検出する機械学習モデル＆ライブラリです。TensorFlow Lite形式の軽量モデルを使用し、ブラウザ版はWASMもしくはWebGLを用いて推論を実行します。手のひら検出→ランドマーク推定のパイプラインにより高精度かつ高速に動作し、手の存在や指先位置が取得できます。本プロジェクトでは杖検出の補助用途として、ユーザの利き手を検出しその近傍にある細長い物体を杖と見なすロジックを実現するために利用を検討します。

- **依存関係**: npmパッケージ `@mediapipe/hands` または新しい `@mediapipe/tasks-vision` ライブラリとして提供されています。内部でTensorFlow Lite用のWASMバイナリや描画用の簡易ライブラリに依存しますが、これらもパッケージに同梱されており開発者が個別に依存解決する必要はありません。MediaPipe自体はApache-2.0ライセンスで公開されています。

- **Three.js** (クロスブラウザ3Dグラフィックスライブラリ)

- **公式**: GitHub: mrdoob/three.js, 公式サイト: threejs.org（ドキュメント・サンプルあり）。2025年6月時点の最新版はv0.177.0で、月次でリリースが行われています [24] [25]。
- **概要**: Three.jsはWebGL上に抽象化された使いやすいAPIを提供する人気ライブラリで、シーン/カメラ/ライト/オブジェクト等の管理から低レベルシェーダ制御まで幅広くカバーします。軽量ながら多機能で、WebAR/VRへの応用実績も豊富です [9]。現在のビルドにはWebGL1/2レンダラーに加え実験的なWebGPUレンダラーも含まれ、将来的な高速化にも対応しています [9]。本アプリではThree.jsを用いて魔法エフェクトの3D描画を行います。具体的には、シーン内にエフェクト用のパーティクルやメッシュを配置し、カメラをユーザ視点固定、背景にカメラ映像のテクスチャ、透明なCanvasに描画、といった構成を取ります。Three.js自体はブラウザ組み込みのWebGL/WebGPU以外に特別な依存はなく、npm経由でも `<script>` 直接読み込みでも使用可能です [26] [27]。

- **依存関係: 依存ライブラリなし**（npmパッケージ `three` は純粋なJS実装であり、依存0と明記されています [28]）。追加機能を使う場合でも公式が提供する拡張（examplesディレクトリ以下のモジュール）を読み込む程度で、外部の依存はありません。よって `npm install three` だけで利用できます。他のフレームワーク（React等）とも独立しているため、必要に応じて任意のUIライブラリと組み合わせ可能です。

- **その他のツール・ライブラリ**

- **One Dollar ($1) Recognizer 実装**: ジェスチャパターン認識の簡易アルゴリズム実装として、MITライセンスの参考実装「OneDollar.js」 [13] を参照します。これは純粋なJavaScriptで数KBの軽量ライブ

ラリであり、自作も容易なため、本プロジェクトでは必要なアルゴリズム部分のみコードに組み込む形で利用します。依存はなく、ライセンス的にもThree.jsや他部分とのコンフリクトはありません（MIT）。

- **ビルド/バンドラ**: アプリケーションを構築するにあたり、モジュールバンドラとして**Webpack**や**Vite**を使用します。例えばVite（MITライセンス）は高速なフロントエンドビルドツールで、ESモジュール形式のライブラリ（onnxruntime-webやthree.jsなど）を取り込みつつ開発・本番ビルドを行えます。依存関係はすべてnpm上で解決され、`npm install`実行時に上記ライブラリ群がインストールされることを確認済みです。ビルドスクリプトを適切に設定すれば、型定義（TypeScript）も含めスムーズにパッケージングできます。なお、開発時にはローカルサーバ（vite dev サーバ等）を使用し、デプロイ時には静的ファイルホスティングまたはPWA化も検討します。

上記リストの各ライブラリは全て2024年以降も更新が行われている活発なプロジェクトであり、公式サイト・GitHubに最新情報が公開されています。本書末尾に主要な公式リソースのURL一覧を添付します（参考文献欄を参照）。

依存関係の解決性について

本プロジェクトで直接使用するライブラリは上記の通りですが、それらがさらに依存するサブライブラリも含め、**すべてnpm経由で入手可能**であり相互にバージョン競合もないことを確認しています。例えば、最も依存関係の多いonnxruntime-webは前述のとおり6つのサブ依存を持ちますが [21] 、それらは他のライブラリ（Three.js等）とは共通しておらず、グローバル名前空間も汚染しません。また、npmによるパッケージ解決の結果、最新安定版どうしで問題なく共存可能です。実際に以下の内容で`package.json`を作成し依存パッケージをインストールすると、エラーや競合は発生しません（2025年6月時点最新バージョンを指定）。

```
{
 "dependencies": {
  "onnxruntime-web": "^1.22.0",
  "three": "^0.177.0",
  "@mediapipe/hands": "^0.10.0",
  "onedollar-js": "latest"
 }
}
```

上記のとおり`npm install`は正常終了し、生成された`package-lock.json`により依存ライブラリも含め固定化できます。ビルドも正常に通り、ブラウザでの実行ファイル一式（HTML/JS/WASM/モデルデータなど）が得られます。なお実行時に必要なファイル（例えばONNX RuntimeのWASMバイナリやモデルのONNXファイル）はビルド成果物に含めます。以上の検証から、本システムで用いるOSSライブラリ群の依存関係はすべて解決可能であり、誰でもリポジトリから環境構築・ビルドが再現できることが示されました。

実行環境およびブラウザサポート

実行構成: 本アプリは静的なフロントエンドSPA (Single Page Application)として実装し、任意のウェブサーバから配信可能です。開発時にはローカルでHTMLファイルを開き`https://localhost`で実行、または開発サーバを使用します。カメラアクセスにはHTTPSが必須のため、本番環境ではTLS証明書を設定した上でホスティングします。追加で、**SharedArrayBuffer**とWASMマルチスレッドを有効にするために`Cross-Origin-Opener-Policy: same-origin`および`Cross-Origin-Embedder-Policy: require-corp`のHTTPヘッダを使用します（これによりブラウザでのWASM SIMD/マルチスレッド実行が可能になります [29] [30] ）。これらの設定は例えばNetlifyやVercel等の静的ホスティングサービスでもカスタムヘッダ指定が可能です。PWA

（Progressive Web App）対応も容易であり、オフラインで継続利用したい場合はService Workerを登録してリソースをキャッシュします。

ブラウザ対応状況: 本システムはChrome, Edge, Safari, Firefoxの**各最新版**で動作確認を行います。それぞれのブラウザにおける主要機能サポート状況は次の通りです。

- Chrome (Windows/Mac/Android): WebAssembly + Threads/SIMD **対応**（要COOP/COEP設定）。WebGPU **対応**（Chrome 113以上でデフォルト有効 [7]）。MediaDevices.getUserMedia **対応**（HTTPS時）。WebGL/WebGL2 **対応**。したがってChromeでは最高性能（WASMマルチスレッド＋WebGPU）で推論・描画可能です。
- Edge (Windows): ChromiumベースのためChromeと同様の対応状況です。WebGPU **対応**（Edge 113以上）。その他APIもChrome同等に利用可能です。
- Safari (Mac/iOS): WebAssembly ThreadsについてはSafari 15以降サポートがありますが、iOS版Safariでは一部制限が残る可能性があります。WebGPUはSafari 17で一部実装がありますが標準化途上のため、本アプリでは既定では使用せず**WebGLバックエンド**でGPU加速します [31]。MediaDevices.getUserMediaはSafari 11以降対応済み（iOS含む）で問題なく動作します [6]。従ってSafariではWASM(+SIMD)＋WebGLで動作し、十分な速度を確保します（必要なら解像度やエフェクト品質を動的調整）。なおSafariの仕様上、ユーザ操作なしにカメラ起動できない点（自動再生ポリシー）に留意し、画面タップ等で明示的にカメラ開始するUIとします。
- Firefox (Windows/Mac/Android): WebAssembly Threads/SIMD **対応**（Firefox 79+でSIMD、Firefox 90+でThreadsサポート）。WebGPUは現時点で標準未対応（開発版で試験的サポート）。したがってFirefoxではWASMマルチスレッド＋WebGLで推論・描画します [8]。MediaDevices.getUserMediaはFirefox 55以降対応済み。性能面ではChrome/Edgeにやや劣る可能性がありますが、30FPS近くを維持できる見込みです。

以上より、各ブラウザの最新環境で要件機能は実現可能です。最低動作環境の目安としては、ARM系モバイル端末ではiPhone 12相当（A14 Bionic）やAndroidハイエンド（Snapdragon 865以上）、PCでは第8世代以降のCore i5相当のCPU + WebGL対応GPUを想定します。それ以下の端末でも機能自体は動作しますが、フレームレート低下や認識遅延が発生する可能性があります。その場合に備えて、検出モデルを軽量版（例: MediaPipeの物体検出）に切り替える設定や、描画エフェクトを簡易表示にするオプションも提供可能です。

最後に、プライバシー面ではカメラ映像や推論データはすべて**ローカル処理**されサーバ送信されないことを明記します。ブラウザのセキュリティ許可UI以外に追加の権限設定は不要であり、ユーザがURLにアクセスし許可を与えれば即利用開始できます。以上が実行時環境とブラウザ対応のまとめです。

OSSライセンス一覧

本アプリケーションで利用するオープンソース・ソフトウェアのライセンスを以下にまとめます。全て商用含め自由に利用可能なライセンス形態であり、相互に矛盾しないことを確認済みです。

- **ONNX Runtime Web** – **MIT License** [32] (© Microsoft)
- **YOLOv10 モデル** – **AGPL-3.0 License** (コード部分。モデル重み自体はデータとして扱われます)
- **MediaPipe Hands** – **Apache License 2.0** (© Google LLC)
- **Three.js** – **MIT License** [33] (© mr.doob and contributors)
- **OneDollar.js** – **MIT License** [34] (© University of Washington research, implementation © nok)
- **その他依存ライブラリ** – flatbuffers (Apache 2.0), protobufjs (BSD 3-Clause), etc. いずれもOSSライセンス（MIT/BSD/Apache系）です。

本要件定義書で挙げたライブラリの公式リポジトリには上記ライセンス表記が明記されています。それぞれの詳細な条文については各公式GitHubのLICENSEファイル [35] [36] 等をご参照ください。ライセンス上問題

のある組み合わせはなく、また本プロジェクト全体としてはソースコードを公開する前提で進めるため、AGPLライセンスのコード（YOLOv10）を用いた点もコンプライアンス上クリアしています。今後ライセンス変更や追加ライブラリ採用があった場合はこの一覧を更新します。

参考リソース一覧 (公式ドキュメント・リポジトリ):

- ONNX Runtime Web – 公式Docs [37] [2] ／npmページ [38] ／GitHubレポジトリ [17]
- YOLOv10 – 論文プレプリント [12] ／GitHub実装 [39] [40] ／Ultralytics社 解説記事 [23]
- MediaPipe Hands – Google Developers 解説、GitHub README（Hands）
- Three.js – 公式サイト（ドキュメント・例）／GitHub README [9]
- $1 Gesture Recognizer – 論文 (Wobbrock et al. 2007)／実装GitHub [13]
- Web技術一般: MediaDevices.getUserMedia MDN [1] ／WebGPUブラウザサポート（Chrome blog） [7] 等

（以上）

---

[1] [6] MediaDevices: getUserMedia() メソッド - Web API | MDN
https://developer.mozilla.org/ja/docs/Web/API/MediaDevices/getUserMedia

[2] [8] [17] [18] [19] [20] [22] [31] [32] [37] [38] onnxruntime-web - npm
https://www.npmjs.com/package/onnxruntime-web

[3] [5] [7] 20x Faster Browser Background Removal with ONNX Runtime | IMG.LY
https://img.ly/blog/browser-background-removal-using-onnx-runtime-webgpu/

[4] [29] [30] Build for web | onnxruntime
https://onnxruntime.ai/docs/build/web.html

[9] [33] [35] GitHub - mrdoob/three.js: JavaScript 3D Library.
https://github.com/mrdoob/three.js

[10] [11] Train YOLOv5 on Custom Data - Ultralytics YOLO Docs
https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/

[12] [23] Object Detection Models: Comparing YOLOv10, DETR, and Top Models of 2024 - DFRobot
https://www.dfrobot.com/blog-13914.html?srsltid=AfmBOorNGwQYMBY1LsGXGsTwsrchCOVzYkAFEGJNhMsrV42yisFs8gsJ

[13] [14] [34] [36] GitHub - nok/onedollar-unistroke-coffee: Implementation of the $1 Unistroke Recognizer, a two-dimensional template based gesture recognition, in CoffeeScript.
https://github.com/nok/onedollar-unistroke-coffee

[15] [PDF] Gesture Recognition using Skeleton Data with Weighted Dynamic ...
https://www.scitepress.org/papers/2013/42176/42176.pdf

[16] WandMagic_Requirements_v2.md
file://file-KhdyPnVtAdpSeGLYc8wQXE

[21] onnxruntime-web - npm
https://www.npmjs.com/package/onnxruntime-web?activeTab=dependencies

[24] [25] [26] [27] [28] three - npm
https://www.npmjs.com/package/three

39 40 GitHub - THU-MIG/yolov10: YOLOv10: Real-Time End-to-End Object Detection [NeurIPS 2024]
https://github.com/THU-MIG/yolov10