AI V-Tuber システム：技術設計と開発に関する調査報告書 (改訂版 v2)

エグゼクティブサマリー

（最終段階で記述）

1. AI V-Tuber システムアーキテクチャ設計

- 1.1. はじめに
  本報告書は、提示された構想に基づき、AI V-Tuber システムの設計と開発を進めるための技術的な調査と検討事項を整理することを目的とする。AI V-Tuber システムは、vtuber-behavior-engine、stage-director（MCPサーバー機能を含む）、vtube-stage（Webアプリケーション/VRM表示）、および OBS Studio を連携させ、自律的な対話と表現を行う V-Tuber を実現するものである。本報告書では、システム全体のアーキテクチャ、各コンポーネントの技術選定、連携方法、対話生成、感情表現、開発環境、そして開発における課題と考慮事項について詳細に検討する。
- 1.2. システムコンポーネント概要
  本システムは、以下の主要コンポーネントから構成される。
  - **vtuber-behavior-engine:** Python ベースで実装され、マルチエージェントシステムとして V-Tuber の対話生成、感情判断、行動決定（画面要素の指示など）を担当する頭脳部分。
  - **stage-director:** Python ベースで実装され、vtuber-behavior-engine と vtube-stage 間の通信を仲介するミドルウェア。AI からの指示を vtube-stage が解釈可能な具体的なコマンドに変換し、WebSocket を介して送信する。
  - **vtube-stage:** TypeScript/React/Vite/Three.js で構築され、VRM モデルの描画、アニメーション、表情制御、および画像や Web ページなどの画面要素の表示を担当するフロントエンド部分。stage-director から WebSocket 経由でコマンドを受信し、画面を更新する。
  - **OBS Studio:** ライブストリーミングや録画を行うためのソフトウェア。vtube-stage の画面をキャプチャし、最終的な映像として配信・記録する。
- 1.3. コンポーネント連携とデータフロー分析
  システム全体の連携は、以下のデータフローで実現されることを想定する。
  1. vtuber-behavior-engine が、キャラクターの対話内容、感情、および表示すべき画面要素（画像、Webページ、背景など）を生成する。
  2. vtuber-behavior-engine は、生成された指示（対話テキスト、感情ラベル、表示コマンドなど）を構造化されたデータとして stage-director に送信する。ADK を利用する場合、これは MCP ツールセット（MCPToolset）を介したツール呼び出しとして実装される可能性がある [1]。
  3. stage-director は、vtuber-behavior-engine からの抽象的な指示を受け取り、

vtube-stage を制御するための具体的なコマンド（例: setCharacterPose, setCharacterExpression, displayImage）に変換する。
4. stage-director は、変換したコマンドを WebSocket プロトコルを使用して vtube-stage に送信する。
5. vtube-stage は、WebSocket を介してコマンドを受信する。
6. 受信したコマンドに基づき、vtube-stage は Three.js を使用して VRM モデルのポーズ、表情、アニメーションを更新し、指定された画像や Web ページを画面上に表示または更新する。
7. OBS Studio は、vtube-stage の画面を指定された解像度で「ブラウザソース」としてキャプチャし、配信または録画を行う。

このアーキテクチャにおいて、stage-director は AI の論理（vtuber-behavior-engine）と視覚的表現（vtube-stage）の間で標準化されたインターフェースを提供する重要な役割を担う[3]。MCP は、LLM アプリケーションと外部ツールやデータソース間の接続を標準化することを目的としており、「AI アプリケーション用の USB-C ポート」に例えられる[4]。本システムでは、vtube-stage を stage-director から制御可能な「ツール」と見なすことができる。この構成により、AI の「頭脳」と「舞台」が明確に分離される。stage-director は、AI からの高度な意図を具体的な視覚的アクションに変換する「舞台監督」として機能する。このモジュール性は、システムの保守性を向上させ、各コンポーネントの独立した開発とアップグレードを可能にする。例えば、vtube-stage の描画ロジックが変更された場合、修正が必要なのは stage-director のコマンド実装のみであり、vtuber-behavior-engine のコアロジックには影響しない。同様に、vtuber-behavior-engine も、定義されたインターフェースを介して通信する限り、vtube-stage に影響を与えることなく交換またはアップグレードが可能である。このような関心の分離は、本システムのような複雑なソフトウェア開発において大きな利点となる。

- 1.4. インターフェース定義
  各コンポーネント間の連携を確実にするため、インターフェースを明確に定義する必要がある。
  - **vtuber-behavior-engine <-> stage-director:**
    - **データ形式:** JSON 形式が適している。エージェントの意図（対話内容、感情、実行したいコマンドの種類）、および関連パラメータ（キャラクターID、表情名、画像URLなど）を含む構造を定義する。
    - **通信プロトコル:** 両コンポーネントが Python で実装され、同一環境で動作する場合は、直接的なライブラリ呼び出しやローカルプロセス間通信（IPC）が考えられる。ADK を使用する場合、ADK エージェントが MCPToolset を介して stage-director のツールを呼び出す形式となる[1]。分散環境の場合は gRPC や REST API も選択肢となるが、本構成ではその必要性は低い。
  - **stage-director <-> vtube-stage:**
    - **プロトコル:** WebSocket を使用する。
    - **メッセージ形式:** MCP のベースプロトコルである JSON-RPC 2.0[3] も利用可能だが、本システム固有の制御コマンド体系を考えると、カスタム JSON メッセージ形

式の方が実装が容易である可能性がある。例えば、{"command": "setCharacterExpression", "payload": {"characterId": "char1", "expressionName": "happy", "weight": 0.9}} のような形式を定義する。vtube-stage からの応答（Acknowledgement）が必要な場合は、{"status": "success", "command": "..."} や {"status": "error", "message": "...", "command": "..."} といった形式を定義する。WebSocket は双方向通信をサポートする [8]。
        - カスタム JSON メッセージ形式を採用する利点は、実装の簡潔さにある。vtube-stage は厳密な意味での MCP クライアントではなく、制御対象のエンドポイントであるため、完全な JSON-RPC スタックを双方に実装することは、特定の視覚制御コマンドを送受信するという目的においては過剰な複雑さをもたらす可能性がある。シンプルでタスク固有の JSON メッセージは、実装とデバッグを容易にする。
    - **vtube-stage <-> OBS Studio:**
        - 基本連携: 主に OBS の「ブラウザソース」機能による一方向の画面キャプチャとなる。
        - 双方向制御（オプション）: 必要に応じて、obs-websocket ライブラリ [11] を使用し、vtube-stage または stage-director から OBS Studio を直接制御する（例: AI の判断に基づき OBS のシーンを切り替える）ことも検討する。

2. vtuber-behavior-engine (Python) 調査

- 2.1. マルチエージェントフレームワーク比較
  vtuber-behavior-engine の中核となるマルチエージェントシステムの構築には、適切な Python フレームワークの選定が重要となる。以下に主要な候補フレームワークを比較検討する。
    - **Google ADK (Agent Development Kit):**
        - 概要: Google が提供するオープンソースの Python ツールキット。コードファーストのアプローチを採用し、柔軟性と制御性を提供する [2]。特に Gemini モデルや Google Cloud (Vertex AI) との緊密な統合を重視して設計されている [2]。階層的なマルチエージェントシステムの構築に強みを持ち、複数の専門化されたエージェントを組み合わせてモジュール化されたスケーラブルなアプリケーションを設計できる [2]。SequentialAgent, ParallelAgent, LoopAgent といったワークフローエージェントによる予測可能なパイプライン定義や、LlmAgent による LLM 駆動の動的なルーティング（エージェント転送）が可能 [2]。組み込みの評価機能、デプロイ支援（Cloud Run, Vertex AI Agent Engine）、開発用 Web UI を提供する [2]。ネイティブな双方向ストリーミング（音声・動画）機能も特徴 [2]。
        - **MCP** 連携: MCPToolset クラスを通じて、外部の MCP サーバー（本システムでは stage-director）が提供するツールを ADK エージェントから利用することが可能 [1]。ADK エージェントは MCP クライアントとして機能する [1]。また、ADK ツールを

MCP サーバーとして公開する実装も可能である [2]。
    - ツールエコシステム: 検索やコード実行などの組み込みツール、カスタム Python 関数、OpenAPI 仕様に基づくツールに加え、LangChain, LlamaIndex, CrewAI といったサードパーティライブラリや他の ADK エージェント自体をツールとして統合できる [2]。
    - 考慮事項: 非同期処理や状態管理の複雑さが指摘されることがある [16]。ドキュメントに一部不備や分かりにくさが存在する可能性も指摘されている [16]。Google Cloud 環境に最適化されているが、他の環境でも利用可能 [13]。
- **LangGraph:**
    - 概要: LangChain Inc. によって開発された、状態を持つマルチアクタアプリケーション構築のためのライブラリ（LangChain フレームワークから独立して使用可能）[20]。ワークフローをグラフ（DAG）として表現し、エージェント間のインタラクションフローと状態遷移に対する詳細な制御を提供することに重点を置いている [20]。複雑な複数ステップのタスクにおいて、分岐処理、エラーハンドリング、状態の可視化に優れる [21]。LangChain の豊富なインテグレーションを継承している [21]。
    - 考慮事項: シンプルなユースケースには過剰な複雑さをもたらす可能性がある [23]。CrewAI と比較して学習曲線が急である可能性がある [22]。
- **CrewAI:**
    - 概要: ロールプレイング型の自律 AI エージェントのオーケストレーションに特化したフレームワーク [20]。明確な役割、ツール、目標を持つ専門エージェント間の協調作業を促進するように設計されている [20]。"Crew", "Agent", "Task", "Process" といった高レベルな抽象化を提供し、開発者体験と迅速なプロトタイピングを重視している [20]。組み込みのメモリ管理、エラーハンドリング、テストツール、CLI を提供する [21]。
    - 考慮事項: フレームワークとしての意見が強く（Highly opinionated）、LangGraph や ADK と比較して深いカスタマイズの柔軟性が低い可能性がある [23]。主に役割ベースのタスク分解に適している [20]。
- フレームワーク比較表:

特徴	**Google ADK**	**LangGraph**	**CrewAI**
コア思想	コードファースト、柔軟性、モジュール性、Google Cloud 統合 [2]	グラフベースの状態管理、複雑なワークフロー制御 [20]	ロールベース協調、開発者体験、迅速なプロトタイピング [20]
マルチエージェント構造	階層構造（親子関係）、サブエージェント [2]	グラフノードとしてのエージェント、状態遷移 [21]	クルー（Crew）、役割（Role）、タスク（Task） [20]

オーケストレーション	ワークフローエージェント (Sequential, Parallel, Loop)、LLM 駆動ルーティング (transfer_to_agent) [2]	グラフのエッジと条件分岐による明示的なフロー制御 [20]	プロセス（逐次、階層）、タスク委任 [20]
状態管理	共有セッション状態 (session.state)、メモリ管理機能 [2]	中央集権的な永続レイヤー、グラフ状態 [20]	組み込みメモリ管理（短期・長期） [21]
ツール統合 **(MCP**含む**)**	MCPToolset による MCP 連携 [1]、組み込み/カスタム/OpenAPI/サードパーティツール [2]	LangChain 統合、カスタムツール	カスタムツール、LangChain ツール統合 [24]
使いやすさ**/**学習曲線	中～高（非同期/状態管理が複雑な可能性 [16]）	中～高（グラフ概念、複雑な制御 [22]）	低～中（高レベル抽象化、迅速な開始 [22]）
エコシステム**/**統合	Google Cloud (Vertex AI, Gemini) 最適化 [13]、LiteLLM [16]、他フレームワーク連携 [13]	LangChain エコシステム [21]	LangChain 統合 [24]
デプロイオプション	コンテナ (Cloud Run, Docker)、Vertex AI Agent Engine [2]	コンテナ化、手動デプロイ	コンテナ化、手動デプロイ
ストリーミングサポート	ネイティブな双方向音声・動画ストリーミング [2]	なし（カスタム実装が必要）	なし（カスタム実装が必要）
コミュニティ/ドキュメント	成長中、Google サポート [2]、ドキュメントに一部課題？ [16]	LangChain コミュニティの一部 [20]	活発なコミュニティ、比較的新しい [21]

* **推奨:** 本 AI V-Tuber システムにおいては、**Google ADK** の採用を推奨する。主な理由は以下の通りである。

*   **MCP 連携:** `MCPToolset` [1, 2] を介した stage-director との直接的な連携機能は、AI が vtube-stage（VRM や画面要素）を制御する上で不可欠であり、ADK はこれをネイティブにサポートしている。
*   **マルチエージェント機能:** 階層構造、ワークフローエージェント、LLM 駆動ルーティングなど、本システムで想定される「舞台制御」「キャラクター」といった役割分担と連携を実現するための機能が豊富である [2, 15]。
*   **ストリーミング:** 将来的に音声対話などを導入する場合、ネイティブなストリーミング機能 [2, 13] は大きなアドバンテージとなる。
*   **柔軟性:** 他のフレームワーク（LangGraph, CrewAI）をツールとして利用できる可能性 [13] もあり、将来的な拡張や部分的な利用にも対応できる。
*   **デプロイ:** Vertex AI Agent Engine [2, 13] との統合により、Google Cloud 環境でのスケーラブルなマネージドデプロイへの道筋が明確である点は、長期的な運用を見据えた場合に大きな利点となる。他のフレームワークでは、より多くの手動でのインフラ設定が必要になる可能性がある。

*   フレームワークの選択は、単なる機能比較だけでなく、目指すアーキテクチャパターンとの適合性も考慮すべきである。ADK は階層制御と Google Cloud デプロイに適している。LangGraph は状態遷移が複雑なプロセスに向いている。CrewAI は役割分担が明確なチームシミュレーションに優れている。V-Tuber システムは、階層（舞台/キャラクター）、状態（対話コンテキスト）、役割（個性）の要素を併せ持つため、これらのパターンに柔軟に対応できる ADK が有利と考えられる。ADK が提供する多様なオーケストレーションエージェント（Sequential, Parallel, Loop）[2, 15] は、システム内の様々な連携ニーズへの適応力をさらに高める。

- 2.2. エージェント役割定義
  選択したフレームワーク（ADK を想定）に基づき、各エージェントの役割と責任範囲を詳細化する。
  - 舞台制御エージェント **(Stage Control Agent):**
    - 役割**:** システム全体の対話フロー管理、話題提供・転換、キャラクター間の発話ターン制御、キャラクター以外の画面要素（背景、画像、Web ページ表示など）の指示（stage-director 経由）、必要に応じた OBS アクションのトリガーを担当するオーケストレーター。
    - 責任範囲**:** 全体の進行管理、大局的なコンテキスト維持、外部要素（画面、OBS）の制御。
  - キャラクターエージェント **(Character Agent):**
    - 役割**:** 担当キャラクターのペルソナに基づいた対話内容の生成、自身の感情状態の判断、担当 VRM の表現（表情、ポーズ、アニメーション）の指示（stage-director 経由）を担当する。複数キャラクターが存在する場合は、キャラクターごとにインスタンス化される。

        - 責任範囲: キャラクター固有の対話生成、感情表現、VRM 制御。
- 2.3. エージェント協調メカニズム設計
  ADK を利用する場合、以下のメカニズムを組み合わせてエージェント間の協調を実現する。
    - 階層構造: 舞台制御エージェントを親エージェントとし、キャラクターエージェントをサブエージェントとして構成する [2]。
    - 情報共有: 共有セッション状態 (session.state) を利用して、現在の話題、対話履歴、各キャラクターの状態などのコンテキスト情報をエージェント間で共有する [2]。例えば、舞台制御エージェントが現在の話題を state['current_topic'] に書き込み、キャラクターエージェントがそれを読み取って対話を生成する。キャラクターエージェントは自身の生成した対話や感情状態を state に書き戻す。
    - オーケストレーション（ターン制御など）:
        - **LLM** 駆動委譲: 舞台制御エージェント（LlmAgent）が、対話の流れや指示に基づき、次に発話すべきキャラクターエージェントを判断し、transfer_to_agent 機能 [2] を使用して実行を委譲する。柔軟性が高いが、LLM の判断に依存するため制御が難しい側面もある。
        - ワークフローエージェント: 舞台制御エージェントが SequentialAgent や LoopAgent [2] を使用して、キャラクター間の発話ターンを厳密に制御する。予測可能だが、柔軟性に欠ける可能性がある。
        - 推奨アプローチ: 初期段階では、舞台制御エージェントが SequentialAgent を使用して基本的なターン制御を行い、話題転換や介入などの複雑な判断は舞台制御エージェント内の LLM ロジックで行うハイブリッドアプローチが考えられる。これにより、制御のしやすさと対話の自然さのバランスを取る。
    - 具体例: 舞台制御エージェントが話題を設定 (state['topic'] = '今日の天気') → ワークフローに従いキャラクターAにターンを渡す → キャラクターAが state['topic'] と自身の個性に基づき発話内容を生成し、感情を判断 (state['charA_dialogue'], state['charA_emotion']) → キャラクターAが stage-director のツールを呼び出し表情等を設定 → 舞台制御エージェントが次のターン（キャラクターB）に進める。
- 2.4. stage-director 連携戦略
  vtuber-behavior-engine 内のエージェント（主に舞台制御エージェントとキャラクターエージェント）は、stage-director を介して vtube-stage を制御する必要がある。
    - **ADK MCPToolset** の活用: ADK を使用する場合、MCPToolset [1] を利用する。AI エージェントは、stage-director が公開する関数（例: setCharacterPose）を標準的な ADK ツールとして扱うことができる [1]。
    - ツール呼び出しの責任分担: どのエージェントがどの stage-director ツールを呼び出すかを明確にする。例：キャラクターエージェントは setCharacterExpression, setCharacterPose, triggerCharacterAnimation を呼び出し、舞台制御エージェントは displayImage, changeStageBackground, playAudio などを呼び出す。
    - 接続パラメータ: MCPToolset.from_server [1] に渡す接続パラメータを定義する。ロー

カルで stage-director を実行する場合は StdioServerParameters、リモートサーバーの場合は SseServerParams を使用する [1]。
    - 接続クリーンアップ: MCPToolset.from_server が返す exit_stack を使用して、エージェント終了時に stage-director との接続を確実にクリーンアップすることが重要である [1]。async with 文脈マネージャまたは await exit_stack.aclose() を使用する。

3. stage-director (Python) 設計考察

- 3.1. 要求される API ツールの定義
  クエリ項目 (3a) および V-Tuber の想定されるアクションに基づき、stage-director が vtube-stage 制御のために公開する必要がある具体的なツール（APIエンドポイント）を定義する。以下はその例である。
    - setCharacterPose(characterId: string, poseName: string): 指定されたキャラクターのポーズを設定する。
    - setCharacterExpression(characterId: string, expressionName: string, weight: float): 指定されたキャラクターの表情（ブレンドシェイプ）とその強度を設定する。
    - triggerCharacterAnimation(characterId: string, animationName: string): 指定されたキャラクターのアニメーション（例: 手を振る）を再生する。
    - setVrmPosition(characterId: string, x: float, y: float, z: float): キャラクターのワールド座標を設定する。
    - setVrmRotation(characterId: string, x: float, y: float, z: float): キャラクターの向きを設定する。
    - displayImage(imageId: string, imageUrl: string, position: object, size: object): 指定された URL の画像を、指定された位置とサイズで表示する。
    - hideImage(imageId: string): 指定された ID の画像を非表示にする。
    - displayWebpage(webpageId: string, url: string, position: object, size: object): 指定された URL の Web ページを、指定された位置とサイズで表示する（iframe など）。
    - hideWebpage(webpageId: string): 指定された ID の Web ページを非表示にする。
    - changeStageBackground(backgroundUrl: string): 舞台の背景画像を変更する。
    - playAudio(audioUrl: string): 指定された URL の音声ファイルを再生する（効果音など）。
    - showTextBubble(characterId: string, text: string, duration: float): キャラクターの近くに吹き出しでテキストを表示する（デバッグ用や補助情報として）。
    - clearStageElements(): 画像や Web ページなどの追加要素をすべてクリアする。

  これらのツールは、MCP の「ツール」機能 [3] の概念に基づき、stage-director が vtube-stage に対して実行できる操作として定義される。既存の MCP サーバー例 [26] は主にデータ連携や外部ツール実行に関するものが多いが、機能を関数として公開するという基本原則は同じである。

- 3.2. WebSocket 通信プロトコル設計
  stage-director から vtube-stage へ送信されるコマンドのメッセージ構造を定義する。前述 (1.4) の通り、完全な JSON-RPC よりもシンプルなカスタム JSON メッセージ形式を推奨する。
  - **コマンドメッセージ例:**
    ```JSON
    {
      "command": "setCharacterExpression",
      "payload": {
        "characterId": "char1",
        "expressionName": "happy",
        "weight": 0.9
      }
    }
    ```
  - **応答/確認メッセージ例（必要な場合）:**
    ```JSON
    {"status": "success", "command": "setCharacterExpression"}
    ```
    または
    ```JSON
    {"status": "error", "command": "setCharacterExpression", "message": "Invalid characterId"}
    ```

- 3.3. Python WebSocket サーバーライブラリ評価
  stage-director の実装に使用する Python の WebSocket サーバーライブラリを選定する。
  - **FastAPI:**
    - **利点:** モダンで高性能な ASGI フレームワーク [8]。Starlette ベースの WebSocket サポートを組み込みで提供 [8]。Pydantic による自動データバリデーションと OpenAPI による自動 API ドキュメント生成機能 [8]。依存性注入（Dependency Injection）による構造化の容易さ [8]。活発なコミュニティと豊富なドキュメント [27]。非同期処理（async/await）をネイティブサポート [8]。
    - **欠点:** WebSocket 機能のみが必要な場合、基本的なライブラリと比較して若干複雑になる可能性がある [27]。
  - **websockets ライブラリ:**
    - **利点:** 純粋な WebSocket プロトコルの実装であり、軽量 [30 (示唆)]。基本的なユースケースではシンプルに利用できる。
    - **欠点:** HTTP サーバー機能は含まないため、通常 asyncio や他のフレームワークと組み合わせて使用する必要がある。FastAPI のようなデータバリデーション、API ドキュメント、依存性注入などの組み込み機能は提供されない。

- 推奨: **FastAPI** の採用を強く推奨する。その統合された性質により、WebSocket 接続の処理に加えて、ヘルスチェック用の簡単な HTTP エンドポイントの公開なども容易になり、stage-director の開発を簡素化できる [29]。Pydantic によるデータバリデーションは、vtuber-behavior-engine からのデータやコマンドパラメータの処理に役立つ。また、その高いパフォーマンス [8] は、リアルタイム性が要求される本システムに適している。
- stage-director は単なる WebSocket リレーではなく、AI の意図を解釈し、具体的な Web アプリコマンドに変換するロジックを持つ。FastAPI の提供する構造、バリデーション機能、非同期サポートは、この変換ロジックをクリーンかつ堅牢に実装する上で、websockets のような基本的なライブラリを使用するよりも有利である。

4. vtube-stage (TypeScript/React/Vite/Three.js) 実装戦略

- **4.1. @pixiv/three-vrm** による **VRM** 読み込みとアニメーション
  - ライブラリ: @pixiv/three-vrm は Three.js 環境で VRM モデルを扱うための標準的なライブラリである [31]。
  - 読み込み: Three.js の GLTFLoader に VRMLoaderPlugin を登録して .vrm ファイルを読み込む [32]。ロード完了後、gltf.userData.vrm プロパティを通じて VRM インスタンスにアクセスする [32]。
  - 主要コンポーネント: ロードされた vrm オブジェクトは、以下の主要なマネージャーコンポーネントを含む。
    - vrm.expressionManager: ブレンドシェイプ（表情）を制御する [34 (例から示唆)]。setValue(name, weight) メソッドで表情の重みを設定する [34 (使用例), 36]。
    - vrm.humanoid: ヒューマノイドボーンへのアクセスと制御を提供し、ポーズを設定する [35 (例から示唆)]。getNormalizedBoneNode(boneName) でボーンノードを取得し、position, rotation, quaternion を操作する [38]。
    - vrm.lookAt: 視線の向きを制御する [34 (例から示唆)]。target プロパティに視線目標となる THREE.Object3D または THREE.Vector3 を設定し、update(deltaTime) を毎フレーム呼び出す [34]。
    - vrm.scene: モデルデータを含む Three.js の Group オブジェクト。これをメインの Three.js シーンに追加する [32]。
  - アニメーション: ポーズ（Humanoid ボーン）と表情（Expression）に対してキーフレームアニメーションを適用できる [35]。Three.js の標準的なアニメーションシステム (AnimationMixer) を使用し、ボーンのトランスフォームや表情のウェイトをターゲットとしてアニメーションさせることが可能と考えられる。humanoidAnimation の例では Mixamo アニメーションの読み込みと再生が示されている [35]。
  - **WebGPU** サポート: v3 以降で対応。MToonMaterialLoaderPlugin の materialType オプションに MToonNodeMaterial を指定する必要がある [31]。Three.js r167 以降が必要。

    - 参照例: 具体的な実装については、公式の Example ページ [35] の basic.html, expressions.html, bones.html, lookat.html, animations.html などを参照することが推奨される。
- **4.2. WebSocket** クライアント実装
    - 要件: stage-director に接続し、コマンドを受信し、必要に応じてステータスや確認応答を送信する。
    - ネイティブ **WebSocket API (JavaScript):**
        - 利点: ブラウザ組み込みであり、外部ライブラリ不要。オーバーヘッドが少ない [30]。接続、送信、受信、切断のためのシンプルな API を提供する。
        - 欠点: 再接続ロジック、接続断時のメッセージキューイング/ハンドリング、複雑な状態管理などを手動で実装する必要がある [30]。
    - **Socket.IO** クライアント:
        - 利点: 高レベル API を提供。自動再接続、名前空間/ルーム（本件では不要な可能性が高い）、イベント駆動モデルなどをサポート [10]。一部の複雑さを抽象化する [10]。
        - 欠点: バックエンドに Socket.IO サーバーが必要（FastAPI はネイティブでは使用しないが、統合は可能）。ライブラリ依存関係とオーバーヘッドが増加する [30]。プロトコルが標準 WebSocket ではない [30]。
    - 推奨: React アプリケーション内でネイティブ **WebSocket API** を使用することを推奨する。stage-director との接続は制御された環境内での直接的なものであり、Socket.IO が解決する主な課題（フォールバック、制御不能なネットワーク環境での互換性など）の関連性は低い。接続状態の管理や再接続は、React の状態管理（カスタムフックなど）の範囲内で実装可能である。これにより、不要な依存関係を避け、通信レイヤーを軽量に保つことができる。
    - stage-director と vtube-stage 間の直接的かつ制御された接続という性質は、Socket.IO の主な利点（フォールバック、広範な互換性）の重要性を低下させる。ネイティブ WebSocket は、この特定の内部通信チャネルに対して、より少ないオーバーヘッドで十分な機能を提供する。
- **4.3.** 動的コンテンツ表示方法
    - stage-director からの WebSocket コマンドに基づいて動的に更新されるプロパティ（src など）を持つ React コンポーネントを実装する。画像表示には <img> タグ、Web ページ表示には <iframe> タグなどが考えられる。
    - これらのコンポーネントの表示/非表示、位置、サイズは、React の状態管理を通じて CSS やインラインスタイルを動的に変更することで制御する。
- **4.4.** キャラクター制御メカニズム設計
    - 状態管理: React の状態管理機能（useState, useReducer、または Zustand, Redux などのライブラリ）を使用して、キャラクターの現在の目標状態（ポーズ、表情の重み、視線目標位置など）を保持する。
    - **WebSocket** リスナー: アプリケーションの中心的な部分（例: カスタムフックや

Context API）で stage-director からの WebSocket メッセージをリッスンする。
    - コマンド処理: コマンド（例: setCharacterExpression）を受信したら、ペイロードを解析し、関連する React 状態を更新する。
    - **Three.js** 更新ループ: Three.js の requestAnimationFrame ループ内（react-three-fiber を使用する場合は useFrame フック内）で以下を実行する。
        1. 現在の目標キャラクター状態を React 状態から読み取る。
        2. @pixiv/three-vrm の API を使用して vrm オブジェクトを更新する。
            - 表情: vrm.expressionManager.setValue(expressionName, weight) [34 (使用例)] を呼び出す。React 状態で定義されたアクティブな表情とその重みをループ処理して適用する。
            - ポーズ: vrm.humanoid.getNormalizedBoneNode(boneName) [38] でボーンを取得し、React 状態に基づき position, rotation, quaternion を設定する。
            - 視線: vrm.lookAt.target [34] に視線目標を設定し、ループ内で vrm.lookAt.update(deltaTime) を呼び出す。
            - アニメーション: キーフレームアニメーションがトリガーされた場合、AnimationMixer を更新する [35 (例から示唆)]。
    - 非同期で到着する WebSocket コマンドと、同期的に実行される Three.js のレンダーループの間を橋渡しすることが、ここでの中心的な課題となる。React 状態はこの仲介役として機能し、レンダー ループが継続的に読み取って VRM モデルに適用する目標状態を保持する。このパターンにより、非同期のコマンド受信と同期的なレンダリングプロセスが分離され、スムーズで一貫性のある更新が保証される。

5. 対話・コンテンツ生成技術

- **5.1.** 自然なマルチエージェント会話生成
    - 課題: 複数の AI エージェント間で、個々のペルソナを維持しつつ、一貫性があり、魅力的で、単調でない対話を生成すること [44]。
    - プロンプト設計: エージェントの振る舞いを制御する上で極めて重要。プロンプトには以下の要素を含めるべきである [44]。
        - エージェントペルソナ/プロファイル: 性格、話し方、知識、役割などを定義する。専門家ペルソナは推論能力を向上させることがある [45]。
        - 環境記述: 会話やシーンの状況的背景を提供する。
        - メモリ: 関連する過去の対話や知識（短期記憶・長期記憶）[44]。メモリは対話の多様性に大きく影響する [44]。
        - 対話履歴: 直近の会話のターンを含める。
        - タスク/目標: 現在の会話ターンにおける目的を明確にする。
    - コンテキスト管理: エージェント間で共有されるコンテキスト（対話履歴、世界の状態など）を維持する。LLM のコンテキスト長制限を超えないように管理し、必要に応じて要約技術を用いる [45]。ADK の Session と state がこの管理に利用できる [2]。

    - 発話ターン制御: 次に誰が話すかを管理するロジック（通常は舞台制御エージェント内）を実装する。ルールベース、シーケンシャル、または LLM 駆動型が可能 [45]。
    - コミュニケーション戦略/パラダイム: エージェントがどのように情報を交換するか（例: 1対1、同時発話、黒板モデル）、情報の可視性ルールなどを検討する [45]。V-Tuber の場合、最初は履歴が完全に見える逐次的なターンベース（1対1）のアプローチが最も適切と考えられる。
    - 多様性制御: 温度（temperature）/Top-p サンプリング [44] やプロンプトプルーニング（APP など [44]）のような技術を用いて、出力の多様性を調整できる。一貫性と多様性のバランスを取ることが重要 [44]。
    - フレームワーク: ADK, LangGraph, CrewAI などのマルチエージェントフレームワークは、これらのインタラクションを管理するための構造を提供する [2]。
    - マルチエージェント会話の質は、効果的なプロンプトエンジニアリングとコンテキスト管理に大きく依存する。単に複数の LLM を組み合わせるだけでは良好なインタラクションは保証されず、構造化された役割、明確な目標、共有された履歴、そして明確に定義されたコミュニケーションプロトコルが不可欠である [45]。
- **5.2.** コンテンツ表示タイミングとトリガーロジック
    - 対話の流れに基づいて、画像や Web ページをいつ表示するか、またはアニメーションをいつトリガーするかを決定するロジック（通常は舞台制御エージェント内）を設計する。
    - 方法:
        - キーワード検出: 生成された対話内に特定のキーワードが出現した場合にアクションをトリガーする（シンプルだが、誤検知や見逃しが発生しやすい）。
        - 明示的なエージェントコマンド: キャラクターまたは舞台エージェントが、対話と共に関連コンテンツを表示するためのコマンドを明示的に出力する（例: 「これに関する画像はこちらです: ``」）。LLM が指示形式に正確に従う必要がある。
        - 意味解析: 対話の 意味 やトピックを分析して、適切なコンテンツを決定する（より複雑で、別の LLM 呼び出しや埋め込み分析が必要になる可能性がある）。
    - 推奨: LLM が指定された形式に確実に出力できることを前提として、より直接的な制御が可能な明示的なエージェントコマンドから始めることを推奨する。
- **5.3. LLM** 感情分析と **VRM BlendShape** マッピング
    - ステップ **1:** テキストからの感情分析:
        - LLM（EmoLLMs [50] が利用可能であればそれ、または汎用 LLM へのプロンプト指示）または専用の NLP ライブラリ（NLTK [51], TextBlob [52], spaCy [53], VADER [52], Hugging Face Transformers ライブラリと事前学習済みモデル [54] など）を使用して、生成された対話テキストから表現されている感情を分類する。
        - ターゲットとする感情は、基本的なカテゴリ（喜・怒・哀・驚・無表情など）[37] や、感情価（Valence）/覚醒度（Arousal）の次元にマッピングすることも考えられる。
        - 単なるカテゴリ分類だけでなく、感情の強度（intensity/strength）の分析も考慮する [50]。

        - Python ライブラリがこれらのタスクのためのツールを提供する [51]。
    - ステップ **2:** 感情から **VRM BlendShape** へのマッピング**:**
        - VRM 仕様では、プリセット表情（happy, angry, sad, relaxed, surprised, aa, ih, ou, ee, oh, blink など）が定義されている [36]。これらのプリセットは、基盤となるブレンドシェイプの重みの組み合わせである [36]。
        - 検出された感情（および潜在的な強度）から、VRM の expressionManager.setValue(name, weight) 呼び出しへのマッピング戦略が必要となる。
        - 単純マッピング**:** 検出された感情ラベルと VRM プリセットを直接 1:1 で対応させる（例: "happy" を検出 → VRM の "happy" プリセットを 1.0 に、他を 0.0 に設定）[37 (使用例)]。
        - 強度ベースマッピング**:** 検出された強度に基づいてプリセットの重みを調整する（例: 強い喜び → happy: 1.0, 弱い喜び → happy: 0.5）。
        - 組み合わせマッピング**:** より複雑な感情は、複数のプリセットを組み合わせることで表現する（例: 興奮 = happy + surprised？）。
        - カスタム **BlendShape:** モデルに独自のカスタムブレンドシェイプ [36] が定義されている場合、よりニュアンスのある表現のためにこれらをターゲットにすることができる。
        - 研究アプローチ**:** テキストや画像から直接ブレンドシェイプの重みを生成する研究（LLM/CLIP/VAE を使用）[64] や、顔のランドマーク/特徴量（MediaPipe などから取得）をブレンドシェイプにマッピングする研究 [57] も存在する。テキストから直接ブレンドシェイプを生成するのが理想的だが、実用的なアプローチとしては、テキスト→感情ラベル→プリセット重み、という段階的なマッピングが考えられる。
    - 提案マッピングテーブル**:**

検出感情ラベル	ターゲット **VRM** 表情プリセット	デフォルト重み **/** 強度ロジック
Happy	happy	1.0（または強度に応じて 0.3〜1.0 でスケール）
Sad	sad	1.0（または強度に応じて 0.3〜1.0 でスケール）
Angry	angry	1.0（または強度に応じて 0.3〜1.0 でスケール）
Surprised	surprised	1.0（または強度に応じて 0.3〜1.0 でスケール）

Fear	sad + (オプション: surprised 少量)	sad: 0.7, surprised: 0.2（強度で調整）
Disgust	angry + (オプション: sad 少量)	angry: 0.6, sad: 0.3（強度で調整）
Neutral	neutral (または全プリセット 0)	1.0 (または全プリセット 0)
Relaxed	relaxed	1.0（または強度に応じて 0.3〜1.0 でスケール）
...	...	...

*　**テーブルの意図:** このテーブルは、システムが抽象的な感情を具体的な VRM 表情にどのように変換するかを明確かつ設定可能に定義する。これは感情マッピングコンポーネントの核となるロジックであり、キャラクターの表現力を容易に調整できるようにする。このマッピングロジックを明確に定義することで、V-Tuber の「感情的な演技」のルールを設定し、コードを変更することなく容易に変更・調整が可能になる。

*　離散的な感情ラベルを連続的なブレンドシェイプの重みにマッピングして自然な表情を作り出すことは、簡単な課題ではない。単純な 1:1 マッピングはロボットのように見える可能性がある。感情の強度を取り入れ、ヒューリスティックや小規模な二次モデルに基づいてプリセットを組み合わせることで、よりニュアンス豊かで信憑性のある結果が得られる可能性がある [57, 64, 65]。表現の質は、VRM モデルの表情（ブレンドシェイプ）がどれだけ良く定義されているかにも大きく依存する [63, 66, 67]。

6. OBS Studio 連携方法

- **6.1. vtube-stage** キャプチャ技術
  - ブラウザソース**:** 標準的かつ推奨される方法。OBS Studio は、vtube-stage の URL（ローカル実行の場合は http://localhost:ポート番号 など）をブラウザソースとして指定することで、そのレンダリング出力を直接キャプチャできる。
  - 設定**:** OBS 内のブラウザソースの解像度を、目的の配信/録画解像度に合わせる。vtube-stage がこの解像度で正しくレンダリングされるように設計されていることを確認する。
- **6.2. obs-websocket** による **OBS** 自動化

- 目的: AI システム（stage-director 経由、または vtuber-behavior-engine から直接）が OBS の機能をプログラム的に制御できるようにする [11]。
- プロトコル: WebSocket を使用し、リアルタイムの双方向通信を実現する [11]。OBS の設定（ツール -> WebSocket サーバー設定）で obs-websocket サーバーを有効にし、必要に応じてパスワードを設定する必要がある [11]。デフォルトポートは 4455 [11]。
- ユースケース: 会話のトピックやムードに基づいてシーンを自動切り替えする、AI の指示でソースの表示/非表示を切り替える（例: オーバーレイの表示/非表示）、録画/配信を開始/停止する [11]。
- クライアントライブラリ:
  - **Python:** obsws-python（同期的）[11] または simpleobsws（非同期）[11]。obsws-python は Python 3.10 以上が必要 [69]。接続、リクエスト送信（例: set_current_program_scene, set_scene_item_enabled）、イベント処理の例が示されている [11]。リクエスト名は API コールと snake_case で対応することが多い [69]。
  - **JavaScript:** obs-websocket-js は Node.js およびブラウザ環境で使用可能 [11]。接続とコマンド送信の例が示されている [68]。
- 特定コマンド（プロトコル仕様 [11] より）:
  - シーン切り替え: SetCurrentProgramScene（スタジオモード使用時は SetCurrentPreviewScene も）。sceneName パラメータが必要 [11]。（注意: [71] は古い v4 API SetCurrentScene を使用している可能性がある）
  - ソース表示/非表示: SetSceneItemEnabled。sceneName と、sceneItemId または sceneItemName のいずれか、および表示/非表示を指定する真偽値 sceneItemEnabled が必要 [11]。（注意: [72] はこれを抽象化する obswebsocket-py を使用、[70] は obsws-python をラップする obs-cli を使用）
- obs-websocket を統合することで、AI が配信の「ライブディレクター」となり、会話のコンテキストに基づいてシーンやソースを動的に変更できるようになる。これにより、静的なシーン設定よりもはるかに魅力的でインタラクティブな V-Tuber 体験が実現可能となる。AI の決定が最終的な配信出力に直接影響を与えることで、単なるアニメーションキャラクターを超えた、AI 駆動型の放送へとシステムが昇華する。

7. 開発環境とバージョン管理

- 7.1. モノレポ構成の推奨
  本システムのように、複数のコンポーネント（vtuber-behavior-engine, stage-director, vtube-stage）が密接に連携する場合、モノレポ構成を推奨する。これにより、依存関係の管理、コード共有、アトミックな変更、CI/CDパイプラインの構築が容易になる 12。
- 7.2. ディレクトリ構成案
  モノレポ構成を前提としたディレクトリ構造案は以下の通り。

```
/ai-vtuber-system/
├──.github/                 # GitHub Actionsワークフローなど [11, 32, 74]
```

```
│   └── workflows/
│       ├── ci.yml          # 継続的インテグレーション (テスト、リンティング)
│       └── cd.yml          # 継続的デプロイ (オプション)
├── packages/               # 各コンポーネントのソースコード
│   ├── vtuber-behavior-engine/ # AI制御システム (Python, ADK) [12, 14, 74]
│   │   ├── src/
│   │   ├── tests/
│   │   ├── agent.py        # エージェント定義のエントリーポイント (例)
│   │   ├── pyproject.toml
│   │   └── README.md
│   ├── stage-director/     # MCPサーバー機能を含む (Python, FastAPI) [76, 26, 6]
│   │   ├── src/
│   │   ├── tests/
│   │   ├── main.py
│   │   ├── pyproject.toml
│   │   └── README.md
│   └── vtube-stage/        # Webアプリケーション (TypeScript, React, Vite, Three.js)
[32]
│       ├── public/
│       ├── src/
│       ├── tests/
│       ├── index.html
│       ├── vite.config.ts
│       ├── tsconfig.json
│       ├── package.json
│       └── README.md
├── docs/                   # ドキュメント [76, 14, 11, 74]
│   ├── architecture.md
│   ├── setup.md
│   ├── api/
│   │   ├── stage_director_api.md # stage-director API仕様
│   │   └── websocket_api.md      # WebSocketコマンド仕様
│   └── contributing.md
├── scripts/                # ビルド、デプロイ等の補助スクリプト
│   ├── setup_dev.sh
│   └── deploy.sh
├──.gitignore
├── README.md               # プロジェクト全体の概要 [76, 12, 14, 11, 32, 74]
└── LICENSE                 # ライセンスファイル [12, 14, 11, 32, 70, 73, 77]
```

- **7.3.** ドキュメント
  - ルートREADME.mdにプロジェクト全体の概要を記載 [76]。
  - 各パッケージのREADME.mdにコンポーネント固有の詳細を記載。
  - docs/ディレクトリにアーキテクチャ、セットアップ手順、API仕様、コントリビューションガイドなどを配置 [76]。
- 7.4. ブランチ戦略
  main, develop, feature/xxx, release/vx.x.x, hotfix/xxx のような標準的なブランチ戦略を採用する。
- 7.5. CI/CD
  GitHub Actions 74 を利用し、テスト、リンティング、デプロイの自動化パイプラインを構築する 12。
- 7.6. ライセンス
  適切なオープンソースライセンス（例: Apache 2.0 12, MIT 32）を選択し、LICENSEファイルとREADME.mdに明記する。

8. 主要な課題と緩和戦略

- **8.1.** リアルタイム性能
  - **課題:** LLM の推論遅延、コンポーネント間のネットワーク遅延、vtube-stage のレンダリング/アニメーション処理遅延などが積み重なり、リアルタイム性が損なわれる可能性がある。
  - **LLM 遅延:** 高速なモデル（例: Gemini Flash [1]）の利用や、最適化された推論エンドポイントの活用を検討する。LLM 応答のストリーミング [2] を実装し、V-Tuber がより早く話し始めたり感情表現を開始できるようにする。
  - **ネットワーク遅延:** コンポーネント（vtuber-behavior-engine <-> stage-director <-> vtube-stage <-> OBS）間の遅延を最小限に抑える。コンポーネントの同一サーバー配置や低遅延ネットワークインフラが重要。WebSocket の効率性も寄与する [8]。
  - **アニメーション/レンダリング:** Three.js のレンダリングと VRM 更新の効率化。vtube-stage のパフォーマンス最適化（React の更新、Three.js シーンの複雑さ管理）。
  - **緩和策:** 高性能なライブラリ/フレームワーク（例: FastAPI [8]）の採用、コードの最適化、ストリーミング応答の活用、ネットワークホップ数の最小化。
- **8.2.** 会話の一貫性と単調さ
  - **課題:** 長時間にわたるマルチエージェント対話において、一貫したペルソナ、論理的な流れを維持し、反復的な対話を回避すること [44]。
  - **緩和策:** 高度なプロンプトエンジニアリング、効果的なコンテキスト/メモリ管理 [44]、多様性を制御する技術（温度サンプリング、プロンプトプルーニングなど）[44] の適用、話題転換のためのランダム性や外部トリガーの導入検討、慎重なエージェント役割設計 [20]。

- **8.3.** 同期
  - 課題**:** 対話のタイミングとリップシンクアニメーション、対話の感情と表情、会話イベントと OBS アクションなどを正確に同期させること。
  - 緩和策**:** イベント駆動型アーキテクチャの採用。イベントへのタイムスタンプ付与。stage-director を中央同期機構として利用し、正確なタイミングでコマンドを発行したり、確認応答を調整したりする可能性を検討。WebSocket 通信と vtube-stage 内の状態更新における慎重な設計。
  - この種のマルチコンポーネントリアルタイムシステムにおいて、同期はしばしば過小評価される重要な課題である。対話を聞くタイミング、口の動きを見るタイミング、表情の変化を観察するタイミング、そして OBS シーンの切り替えがわずかにずれるだけで、没入感が大きく損なわれる可能性がある。WebSocket 接続とレンダー ループ全体にわたる堅牢なイベント処理とタイミングメカニズムの設計が最も重要となる。
- **8.4.** エラーハンドリング
  - 課題**:** LLM API エラー、stage-director 接続断、WebSocket 切断 [29]、OBS 利用不可、無効なコマンドなど、発生しうる障害に適切に対処すること。
  - 緩和策**:** すべてのコンポーネントで堅牢なエラーハンドリング（try-except ブロックなど）を実装する。接続にはハートビートメカニズム [41] を使用する。フォールバック動作（例: デフォルトのポーズ/表情、エラーメッセージ表示）を定義する。FastAPI では WebSocketDisconnect および WebSocketException を活用する [29]。ログを監視する。
- **8.5. LLM API** コスト管理
  - 課題**:** 特に複数のエージェントが頻繁に LLM を呼び出す場合、API 利用コストが増大する可能性がある。
  - 緩和策**:** 可能な限りコスト効率の良いモデルを使用する。反復的なクエリに対してキャッシュを実装する。プロンプトを簡潔にするように最適化する。レート制限やコンテキスト要約の頻度削減を検討する。API 使用状況を注意深く監視する。
- **8.6. VRM** アニメーション制御の複雑性
  - 課題**:** 高レベルの概念（感情、アクション）を低レベルのブレンドシェイプの重みやボーンの変形にマッピングするには、慎重な設計と調整が必要であり、自然に見えるアニメーションを実現するのは難しい [57]。
  - 緩和策**:** 標準的な VRM 表情プリセットから始める [36]。明確なマッピングロジックを使用する（5.3 参照）。適用可能な既存のアニメーションライブラリや技術を活用する。視覚的なフィードバックに基づいた反復的な改善が不可欠。適切に定義されたブレンドシェイプを持つ良質なベースモデルを提供する [63]。

9. 提案開発ロードマップ

以下に、段階的な開発ロードマップ案を示す。このアプローチにより、複雑さを分離し、各段階でテストと検証を行いながら進めることができる。

- **9.1.** フェーズ **1:** コア **vtube-stage** と **VRM** レンダリング
  - タスク**:** React/Vite/Three.js プロジェクトのセットアップ、@pixiv/three-vrm [32] を用いた VRM 読み込みと表示の実装、表情・ポーズ・視線制御テスト用の基本的な手動 UI（スライダー/ボタン）[35] の作成、基本的な OBS ブラウザソースキャプチャの設定。
  - 目標**:** VRM のレンダリングと基本的な制御パイプラインを検証する。
- **9.2.** フェーズ **2: stage-director** と **WebSocket** 通信
  - タスク**:** FastAPI [8] を用いた stage-director の開発、基本的な WebSocket 接続ハンドリングの実装 [29]、初期の API コマンド（例: setCharacterExpression）の定義と実装 [2]、vtube-stage での WebSocket クライアント（ネイティブ API 推奨）実装とコマンド受信ログ、フェーズ 1 の手動 UI から stage-director 経由でコマンドを送信するよう接続。
  - 目標**:** バックエンドとフロントエンド間の信頼性の高いリアルタイム通信を確立する。
- **9.3.** フェーズ **3:** 単一キャラクターエージェント統合
  - タスク**:** AI マルチエージェントフレームワーク（例: ADK [2]）の選定とセットアップ、基本的な対話生成が可能な単一キャラクターエージェントの実装、エージェントと stage-director の統合（ADK なら MCPToolset [1]）、単純な対話分析や固定ロジックに基づく表情/ポーズコマンド送信、エージェント出力を stage-director と vtube-stage 経由で VRM に反映、基本的な感情分析とマッピングの実装 [37]。
  - 目標**:** AI 駆動による単一キャラクターの基本的な表情制御を実現する。
- **9.4.** フェーズ **4:** マルチエージェントシステムと対話
  - タスク**:** 舞台制御エージェントの実装、協調メカニズム（階層、共有状態、ターン制御）の実装 [2]、マルチエージェント対話生成能力（コンテキスト処理、ペルソナ維持）の開発 [44]、より自然な表情のための感情分析とマッピングの改良 [50]、非キャラクター要素（画像、Web ページ）の stage-director 経由制御の実装。
  - 目標**:** 複数のエージェントが協調して V-Tuber（複数可）と舞台を駆動するインタラクションを実現する。
- **9.5.** フェーズ **5: OBS** 連携
  - タスク**:** obs-websocket クライアント（例: obsws-python [69]）を stage-director または vtuber-behavior-engine に統合、会話コンテキストに基づいて AI エージェントが OBS のシーン切り替えやソース表示/非表示をトリガーするロジックの実装 [11]。
  - 目標**:** AI 駆動によるストリームプロダクションの制御を実現する。
- **9.6.** フェーズ **6:** 改良と高度な機能
  - タスク**:** リアルタイム性能、同期、エラーハンドリングの改善、会話 AI の強化（より複雑な話題、長期記憶）、より複雑なアニメーションとインタラクションの追加、ユーザー設定オプションの実装、テスト、評価 [2]、デプロイ [2]。
  - 目標**:** プロダクションレベルで利用可能な、堅牢で機能豊富な AI V-Tuber システムを完成させる。

この段階的なアプローチは、依存関係に基づいて論理的にフェーズを積み重ねる。基本となる

視覚表現（フェーズ1）、通信層（フェーズ2）、単一AI（フェーズ3）、複数AI協調（フェーズ4）、最終出力連携（フェーズ5）、そして品質向上（フェーズ6）という順序で進めることで、各段階でのリスクを低減し、より管理しやすく開発を進めることが可能となる。

10. 結論

本調査報告書では、提示された構想に基づき AI V-Tuber システムを設計・開発する上での技術的な検討事項、選択肢、および潜在的な課題について詳細に分析した。

システムアーキテクチャとしては、**vtuber-behavior-engine**、**stage-director**、**vtube-stage**、OBS Studio から構成されるモジュール化されたアプローチが推奨される。特に stage-director は、AI の意図と視覚的表現を分離し、標準化されたインターフェースを提供する上で重要な役割を果たす。

vtuber-behavior-engine には、Google ADK フレームワークが有望な選択肢として挙げられる。その理由は、MCP ツールセットとのネイティブな統合、階層的なマルチエージェントシステムの構築能力、Google Cloud エコシステムとの親和性、そして将来的な拡張性（ストリーミング機能など）にある。舞台制御エージェントとキャラクターエージェントという役割分担と、共有状態や LLM 駆動委譲を組み合わせた協調メカニズムが、効果的なシステム動作の鍵となる。

stage-director の実装には、高性能で多機能な FastAPI フレームワークが適している。vtube-stage との通信には WebSocket を用い、制御コマンドはシンプルな JSON 形式で定義することが推奨される。

vtube-stage 側では、Three.js と @pixiv/three-vrm ライブラリを用いて VRM モデルの読み込みと制御を行う。stage-director からのコマンド受信には、ネイティブ WebSocket API を利用し、React の状態管理を介して非同期コマンドと同期的な Three.js レンダリングループを連携させるアーキテクチャが効果的である。

対話生成においては、マルチエージェント間の自然で一貫性のある会話を実現するために、ペルソナ、コンテキスト、対話履歴を考慮した高度なプロンプト設計とコンテキスト管理が不可欠である。LLM の応答から感情を分析し、それを VRM の表情プリセットにマッピングする仕組みは、V-Tuber の表現力を高める上で重要となるが、自然さを実現するには強度や組み合わせの考慮が必要となる。

OBS Studio との連携では、vtube-stage 画面のブラウザソースキャプチャに加え、obs-websocket を利用した AI による動的なシーン・ソース制御が、よりインタラクティブな配信体験を実現する上で有効である。

開発における主要な課題としては、リアルタイム性能の確保、複数コンポーネント間の同期、

対話の一貫性維持、エラーハンドリング、そして LLM API コスト管理が挙げられる。これらの課題に対しては、本報告書で概説した緩和戦略を初期段階から考慮に入れる必要がある。

提案された段階的な開発ロードマップと、GitHubでのモノレポ構成によるバージョン管理は、リスクを管理しつつ、着実にシステムを構築するための指針となる。

総合的な実現可能性:
提示された構想と本調査結果に基づけば、AI V-Tuber システムの構築は技術的に実現可能である。ただし、特にマルチエージェント対話の自然さ、感情表現のニュアンス、リアルタイム性と同期の確保には、高度な技術力と継続的な試行錯誤が必要となる。
主な推奨事項:

1. フレームワーク選定: vtuber-behavior-engine には Google ADK を採用し、その MCP 連携機能とマルチエージェント機能を活用する。
2. インターフェース: stage-director を介した標準化されたインターフェース（WebSocket + カスタム JSON コマンド）を確立する。
3. バージョン管理: GitHubでモノレポ構成を採用し、提案されたディレクトリ構造とブランチ戦略を導入する。
4. 段階的開発: 提案されたロードマップに従い、コア機能から段階的に開発・検証を進める。
5. 同期設計: 対話、感情、アニメーション、OBS 操作間の同期メカニズムを早期に設計・検証する。
6. プロトタイピングと評価: 特に会話生成と感情表現については、早期にプロトタイプを作成し、定性的・定量的な評価を繰り返しながら改善する。

次のステップ:

- 本報告書に基づき、技術スタックとアーキテクチャの詳細設計を確定する。
- フェーズ 1（コア vtube-stage と VRM レンダリング）に着手し、基本的な視覚表現パイプラインを構築・検証する。
- 並行して、選択した AI フレームワーク（ADK）と LLM に関する詳細な技術検証とプロトタイピングを開始する。
- GitHubリポジトリをセットアップし、提案された構成とブランチ戦略を適用する。

11. 参考文献

- [2]https://google.github.io/adk-docs/
- [1]https://google.github.io/adk-docs/tools/mcp-tools/
- [74]https://github.com/google/adk-samples
- [12]https://github.com/google/adk-python
- [3]https://modelcontextprotocol.io/specification/2025-03-26
- [4]https://modelcontextprotocol.io/introduction

- [26]https://modelcontextprotocol.io/examples
- [78]https://modelcontextprotocol.io/clients
- [5]https://www.philschmid.de/mcp-introduction
- [76]https://github.com/modelcontextprotocol
- [17]https://cloud.google.com/blog/products/ai-machine-learning/build-and-manage-multi-system-agents-with-vertex-ai
- [13]https://developers.googleblog.com/en/agent-development-kit-easy-to-build-multi-agent-applications/
- [16]https://www.reddit.com/r/AI_Agents/comments/1jvsu4l/just_did_a_deep_dive_into_googles_agent/
- [6]https://www.datacamp.com/tutorial/mcp-model-context-protocol
- [79]https://google.github.io/adk-docs/get-started/tutorial/
- [7]https://www.digitalocean.com/community/tutorials/mcp-server-python
- [80]https://openai.github.io/openai-agents-python/mcp/
- [81]https://www.reddit.com/r/LocalLLaMA/comments/1jvsvzj/just_did_a_deep_dive_into_googles_agent/
- [82]https://forums.ankiweb.net/t/thinking-if-mcp-server-could-work-with-anki-connect-thx/57436
- 83
- 84
- [85]https://cloud.google.com/vertex-ai/generative-ai/docs/agent-development-kit/quickstart
- 86
- 87
- [14]https://github.com/google/adk-docs
- [88]https://www.youtube.com/watch?v=5ZWeCKY5WZE
- [89]https://www.youtube.com/watch?v=soC4n-nKWF8
- 90
- [18]https://github.com/google/adk-python/blob/main/pyproject.toml
- [20]https://developer.ibm.com/articles/awb-comparing-ai-agent-frameworks-crewai-langgraph-and-beeai
- [21]https://langfuse.com/blog/2025-03-19-ai-agent-comparison
- [24]https://blog.dataiku.com/open-source-frameworks-for-llm-powered-agents
- [22]https://www.datagrom.com/data-science-machine-learning-ai-blog/langgraph-vs-autogen-vs-crewai-comparison-agentic-ai-frameworks
- [23]https://www.relari.ai/blog/ai-agent-framework-comparison-langgraph-crewai-openai-swarm
- [31]https://pixiv.github.io/three-vrm/packages/three-vrm-materials-mtoon/docs/
- [38]https://github.com/pixiv/three-vrm/issues/1173

- [32]https://github.com/pixiv/three-vrm
- [91]https://gist.github.com/ahuglajbclajep/6ea07f6feb250aa776afa141a35e725b
- [35]https://pixiv.github.io/three-vrm/packages/three-vrm/examples/
- [8]https://www.getorchestra.io/guides/fast-api-websockets-a-comprehensive-guide
- [9]https://softwaremill.com/sse-vs-websockets-comparing-real-time-communication-protocols/
- [29]https://fastapi.tiangolo.com/advanced/websockets/
- [27]https://slaptijack.com/programming/python-quart-vs-fastapi.html
- [28]https://startup-house.com/blog/fastapi-vs-flask-comparison
- [39]https://ably.com/topic/socketio-vs-websocket
- [42]https://www.videosdk.live/developer-hub/websocket/socketio-vs-websocket
- [40]https://apidog.com/articles/socket-io-vs-websocket/
- [10]https://dev.to/imsushant12/websockets-socketio-and-real-time-communication-with-nodejs-4ea0
- [30]https://www.reddit.com/r/node/comments/18vx421/websocket_vs_ws_vs_socketio/
- [41]https://ably.com/blog/websocket-libraries-for-node
- [19]https://stackoverflow.com/questions/10112178/differences-between-socket-io-and-websockets
- [68]https://www.videosdk.live/developer-hub/websocket/obs-websocket
- [71]https://obsproject.com/forum/threads/python-script-to-connect-to-obs-websocket-server-help.173395/
- [72]https://stackoverflow.com/questions/78210213/python-obs-websocket-py-set-source-visibility
- [11]https://github.com/obsproject/obs-websocket
- [69]https://pypi.org/project/obsws-python/1.1.0/
- [70]https://github.com/pschmitt/obs-cli
- [44]https://arxiv.org/html/2412.21102v1 (Inaccessible)
- [49]https://arxiv.org/html/2501.06322v1
- [92]https://arxiv.org/html/2504.07303v1
- [47]https://arxiv.org/html/2411.14033v2
- [45]https://arxiv.org/html/2410.22932v1
- [48]https://arxiv.org/html/2502.14321v1
- [93]https://www.mdpi.com/2076-3417/15/5/2742
- [66]https://www.reddit.com/r/vtubertech/comments/1aorbuo/trying_to_get_vrm_blendshape_to_make_mouth_work/
- 94
- [50]https://arxiv.org/html/2401.08508v2 (Inaccessible)

- [63]https://wiki.virtualcast.jp/wiki/en/vrm/setting/blendshap
- 65
- [25]https://cloud.google.com/vertex-ai/generative-ai/docs/agent-engine/develop/adk
- [15]https://google.github.io/adk-docs/agents/
- [95]https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/
- [75]https://github.com/GoogleCloudPlatform/agent-starter-pack
- [56]https://vrm.dev/en/vrm/vrm_features/
- 96
- [62]https://vrm.dev/vrm1/expression/
- [67]https://vrm.dev/en/univrm/blendshape/blendshape_setup/
- [36]https://vrm.dev/en/univrm/blendshape/univrm_blendshape/
- [97]https://socket.io/docs/v4/tutorial/introduction
- [98]https://socket.io/docs/v4/tutorial/api-overview
- [99]https://socket.io/get-started/chat
- [43]https://socket.io/docs/v4/
- [100]https://socket.io/get-started/
- [77]https://github.com/eumario/ObsWs
- [101]https://github.com/aatikturk/obsws-python/issues
- [73]https://github.com/Elektordi/obs-websocket-py
- [46]https://arxiv.org/abs/2503.22458
- [102]https://arxiv.org/html/2503.22458v1
- [103]https://arxiv.org/html/2412.17481v2
- [53]https://www.netguru.com/blog/python-sentiment-analysis-libraries
- [58]https://airbyte.com/data-engineering-resources/text-analysis-in-python
- [51]https://realpython.com/python-nltk-sentiment-analysis/
- [52]https://www.analyticsvidhya.com/blog/2022/07/sentiment-analysis-using-python/
- [54]https://www.nlplanet.org/course-practical-nlp/02-practical-nlp-first-tasks/08-emotion-classification
- 104
- [57]https://extra-ordinary.tv/2024/01/07/getting-google-mediapipe-to-control-vrm-characters/
- [64]https://arxiv.org/html/2410.02049v1
- [105]https://www.youtube.com/watch?v=MIzJtdvgdb0
- [33]https://pixiv.github.io/three-vrm/packages/three-vrm/docs/
- [106]https://www.youtube.com/watch?v=C3s0UHpwlf8
- [34]https://github.com/pixiv/three-vrm/discussions/1303
- 107
- [108]https://codesandbox.io/examples/package/@pixiv/three-vrm

- [109]https://discourse.threejs.org/t/lerping-camera-and-lookat-together-with-dynamic-taret-react-three-fiber/28896
- 110
- [55]https://www.restack.io/p/ai-for-emotion-recognition-answer-python-code-facial-expression-analysis-cat-ai
- [59]https://pmc.ncbi.nlm.nih.gov/articles/PMC10751270/
- [60]https://m.youtube.com/watch?v=J87jV9OGodw
- [111]https://www.mdpi.com/1424-8220/22/10/3749
- [112]https://www.youtube.com/watch?v=Vq_01gFG2vk
- 61
- [113]https://www.researchgate.net/publication/323285196_Multimodal_Feature_Learning_for_Video_Captioning
- [114]https://ceng.calpoly.edu/surp-archives/
- 115
- 37
- [116]https://discourse.threejs.org/t/how-to-create-a-motion-vector-map/75319
- [117]https://extra-ordinary.tv/2024/01/11/tensorflow-js-in-the-browser-for-expression-prediction/
- 118
- [119]https://stackoverflow.com/questions/64956323/create-three-js-components-with-map-method
- [120]https://stackoverflow.com/questions/18393502/create-a-vector-map-and-make-user-move-through-with-three-js
- [121]https://www.reddit.com/r/VirtualYoutubers/comments/ojmn73/i_made_a_vtuber_web_app_with_face_and_full_body/
- [122]https://pixiv.github.io/three-vrm/docs/modules/three-vrm (Inaccessible)

(Note: Inaccessible sources [44] cannot be used for citation or direct information extraction. Information related to these topics will rely on other available snippets.)

12. 付録

付録**A:** 用語集

- **ADK (Agent Development Kit):** Googleが提供するオープンソースのPythonツールキット。AIエージェントおよびマルチエージェントシステムの構築、評価、デプロイを目的とし、コードファーストのアプローチとGoogleエコシステムとの統合を特徴とする [12]。
- **Agent:** 自律的に目標を達成するために行動する実行単位。ADKでは、LLMエージェント、ワークフローエージェント、カスタムエージェントなどの種類がある [52]。
- **BlendShape (**ブレンドシェイプ**):** 3Dモデルの形状（特に表情）を変化させるための仕組

み。複数のターゲット形状（例: 笑顔、怒り顔）を定義し、それらの重みを調整することで表情を合成する [36]。VRMでは表情制御に利用される [36]。
- **FastAPI:** Pythonのモダンで高性能なWebフレームワーク。非同期処理をサポートし、API開発やWebSocket通信に適している [8]。
- **GitHub Actions:** GitHub上でソフトウェアのビルド、テスト、デプロイなどのワークフローを自動化する機能 [74]。
- **LLM (Large Language Model):** 大規模言語モデル。大量のテキストデータで学習された、自然言語処理能力を持つAIモデル（例: Gemini）[2]。
- **MCP (Model Context Protocol):** アプリケーションがLLMに外部のコンテキスト（データソースやツール）を提供するための標準化されたオープンプロトコル [4]。
- **MCP Client:** MCPホストアプリケーション内に存在し、MCPサーバーとの接続を管理するコンポーネント [4]。
- **MCP Host:** ユーザーが操作し、MCPを介してデータやツールにアクセスするアプリケーション（例: IDE、チャットインターフェース）[4]。
- **MCP Server:** 特定のデータソースやツールへのアクセス機能を提供する軽量プログラム [4]。
- **MCPToolset (ADK):** ADKエージェントがMCPサーバーのツールを利用するためのクラス [13]。
- モノレポ **(Monorepo):** 複数の関連プロジェクトやコンポーネントを単一のバージョン管理リポジトリで管理する開発スタイル。
- マルチエージェントシステム **(MAS):** 複数のエージェントが協調または競合して、単一エージェントでは困難なタスクを解決するシステム [12]。
- **OBS Studio:** オープンソースのライブストリーミングおよび録画ソフトウェア。
- **obs-websocket:** OBS StudioをWebSocket経由でリモート制御するためのプラグインおよびプロトコル [11]。
- オーケストレーション **(Orchestration):** マルチエージェントシステムにおいて、エージェント間の実行フローや連携を制御・調整すること [13]。ADKではワークフローエージェントやLLM駆動ルーティングで実現される [2]。
- **React:** Facebook（現Meta）が開発した、ユーザーインターフェース構築のためのJavaScriptライブラリ。
- **stage-director:** 本システムにおけるコンポーネント名。vtuber-behavior-engineからの指示を受け、vtube-stageを制御するコマンドに変換し、WebSocketで送信する役割。MCPサーバー機能を含む。
- **Three.js:** Webブラウザ上で3Dグラフィックスを作成・表示するためのJavaScriptライブラリ [32]。
- **TypeScript:** Microsoftが開発した、JavaScriptに静的型付けなどの機能を追加したプログラミング言語。
- **Vite:** 高速なフロントエンド開発ビルドツール。
- **VRM:** 主にアバター利用を想定した、人型3Dモデルデータを扱うためのファイルフォー

マット。glTFベースで、表情、ボーン構造、視線制御などの情報を含むことができる [56]。
- **vtube-stage:** 本システムにおけるコンポーネント名。VRMモデルの描画、アニメーション、表情制御、コンテンツ表示など、視覚的な「舞台」を担当するWebアプリケーション。
- **vtuber-behavior-engine:** 本システムにおけるコンポーネント名。V-Tuberの対話生成、感情判断、行動決定などを行うAI制御システム。
- **WebSocket:** 単一のTCP接続上で双方向通信チャネルを提供する通信プロトコル。リアルタイム通信に適している [8]。

引用文献

1. MCP tools - Agent Development Kit - Google, 4月 12, 2025にアクセス、https://google.github.io/adk-docs/tools/mcp-tools/
2. Agent Development Kit - Google, 4月 12, 2025にアクセス、https://google.github.io/adk-docs/
3. Specification - Model Context Protocol, 4月 12, 2025にアクセス、https://modelcontextprotocol.io/specification/2025-03-26
4. Model Context Protocol: Introduction, 4月 12, 2025にアクセス、https://modelcontextprotocol.io/introduction
5. Model Context Protocol (MCP) an overview - Philschmid, 4月 12, 2025にアクセス、https://www.philschmid.de/mcp-introduction
6. Model Context Protocol (MCP): A Guide With Demo Project - DataCamp, 4月 12, 2025にアクセス、https://www.datacamp.com/tutorial/mcp-model-context-protocol
7. MCP Server in Python — Everything I Wish I'd Known on Day One | DigitalOcean, 4月 12, 2025にアクセス、https://www.digitalocean.com/community/tutorials/mcp-server-python
8. Fast API WebSockets: A Comprehensive Guide - Orchestra, 4月 12, 2025にアクセス、https://www.getorchestra.io/guides/fast-api-websockets-a-comprehensive-guide
9. SSE vs WebSockets: Comparing Real-Time Communication Protocols - SoftwareMill, 4月 12, 2025にアクセス、https://softwaremill.com/sse-vs-websockets-comparing-real-time-communication-protocols/
10. WebSockets, Socket.IO, and Real-Time Communication with Node.js, 4月 12, 2025にアクセス、https://dev.to/imsushant12/websockets-socketio-and-real-time-communication-with-nodejs-4ea0
11. obsproject/obs-websocket: Remote-control of OBS Studio ... - GitHub, 4月 12, 2025にアクセス、https://github.com/obsproject/obs-websocket
12. google/adk-python: An open-source, code-first Python ... - GitHub, 4月 12, 2025にアクセス、https://github.com/google/adk-python
13. Agent Development Kit: Making it easy to build multi-agent applications, 4月 12, 2025にアクセス、

https://developers.googleblog.com/en/agent-development-kit-easy-to-build-multi-agent-applications/
14. google/adk-docs: An open-source, code-first Python toolkit for building, evaluating, and deploying sophisticated AI agents with flexibility and control. - GitHub, 4月 12, 2025にアクセス、https://github.com/google/adk-docs
15. Agents - Agent Development Kit - Google, 4月 12, 2025にアクセス、https://google.github.io/adk-docs/agents/
16. Just did a deep dive into Google's Agent Development Kit (ADK). Here are some thoughts, nitpicks, and things I loved (unbiased) - Reddit, 4月 12, 2025にアクセス、https://www.reddit.com/r/AI_Agents/comments/1jvsu4l/just_did_a_deep_dive_into_googles_agent/
17. Build and manage multi-system agents with Vertex AI | Google Cloud Blog, 4月 12, 2025にアクセス、https://cloud.google.com/blog/products/ai-machine-learning/build-and-manage-multi-system-agents-with-vertex-ai
18. pyproject.toml - google/adk-python - GitHub, 4月 12, 2025にアクセス、https://github.com/google/adk-python/blob/main/pyproject.toml
19. Differences between socket.io and websockets - Stack Overflow, 4月 12, 2025にアクセス、https://stackoverflow.com/questions/10112178/differences-between-socket-io-and-websockets
20. Comparing AI agent frameworks: CrewAI, LangGraph, and BeeAI - IBM Developer, 4月 12, 2025にアクセス、https://developer.ibm.com/articles/awb-comparing-ai-agent-frameworks-crewai-langgraph-and-beeai
21. Comparing Open-Source AI Agent Frameworks - Langfuse Blog, 4月 12, 2025にアクセス、https://langfuse.com/blog/2025-03-19-ai-agent-comparison
22. Top 3 Trending Agentic AI Frameworks: LangGraph vs AutoGen vs Crew AI - Datagrom, 4月 12, 2025にアクセス、https://www.datagrom.com/data-science-machine-learning-ai-blog/langgraph-vs-autogen-vs-crewai-comparison-agentic-ai-frameworks
23. Choosing the Right AI Agent Framework: LangGraph vs CrewAI vs OpenAI Swarm, 4月 12, 2025にアクセス、https://www.relari.ai/blog/ai-agent-framework-comparison-langgraph-crewai-openai-swarm
24. A Tour of Popular Open Source Frameworks for LLM-Powered Agents - Dataiku blog, 4月 12, 2025にアクセス、https://blog.dataiku.com/open-source-frameworks-for-llm-powered-agents
25. Develop an Agent Development Kit agent | Generative AI on Vertex AI - Google Cloud, 4月 12, 2025にアクセス、https://cloud.google.com/vertex-ai/generative-ai/docs/agent-engine/develop/adk
26. Example Servers - Model Context Protocol, 4月 12, 2025にアクセス、https://modelcontextprotocol.io/examples
27. Comparing Python's Quart vs FastAPI: Which Async Framework Is Right for You? | slaptijack, 4月 12, 2025にアクセス、

https://slaptijack.com/programming/python-quart-vs-fastapi.html
28. FastAPI vs Flask: A Comprehensive Python Web Framework Comparison | Startup House, 4月 12, 2025にアクセス、https://startup-house.com/blog/fastapi-vs-flask-comparison
29. Install WebSockets - FastAPI, 4月 12, 2025にアクセス、https://fastapi.tiangolo.com/advanced/websockets/
30. WebSocket vs Ws vs Socket.io ? : r/node - Reddit, 4月 12, 2025にアクセス、https://www.reddit.com/r/node/comments/18vx421/websocket_vs_ws_vs_socketio/
31. pixiv/three-vrm-materials-mtoon - GitHub Pages, 4月 12, 2025にアクセス、https://pixiv.github.io/three-vrm/packages/three-vrm-materials-mtoon/docs/
32. pixiv/three-vrm: Use VRM on Three.js - GitHub, 4月 12, 2025にアクセス、https://github.com/pixiv/three-vrm
33. pixiv/three-vrm - GitHub Pages, 4月 12, 2025にアクセス、https://pixiv.github.io/three-vrm/packages/three-vrm/docs/
34. LookAt blendshapes · pixiv three-vrm · Discussion #1303 - GitHub, 4月 12, 2025にアクセス、https://github.com/pixiv/three-vrm/discussions/1303
35. three-vrm example, 4月 12, 2025にアクセス、https://pixiv.github.io/three-vrm/packages/three-vrm/examples/
36. BlendShape Setting - VRM, 4月 12, 2025にアクセス、https://vrm.dev/en/univrm/blendshape/univrm_blendshape/
37. Emotional Features of Short Conversations between a Generative AI-Controlled Virtual Communication Trainer and Four Children - Stephy Publishers, 4月 12, 2025にアクセス、https://www.stephypublishers.com/sojei/pdf/SOJEI.MS.ID.000504.pdf
38. Getting older VRM character to look at the camera · Issue #1173 · pixiv/three-vrm - GitHub, 4月 12, 2025にアクセス、https://github.com/pixiv/three-vrm/issues/1173
39. Socket.IO vs. WebSocket: Key differences and which one to use in 2024 - Ably, 4月 12, 2025にアクセス、https://ably.com/topic/socketio-vs-websocket
40. Socket. IO vs. WebSocket: Keys Differences - Apidog, 4月 12, 2025にアクセス、https://apidog.com/articles/socket-io-vs-websocket/
41. 8 best WebSocket libraries for Node - Ably, 4月 12, 2025にアクセス、https://ably.com/blog/websocket-libraries-for-node
42. Socket.IO vs WebSocket: Comprehensive Comparison and Implementation Guide, 4月 12, 2025にアクセス、https://www.videosdk.live/developer-hub/websocket/socketio-vs-websocket
43. Introduction | Socket.IO, 4月 12, 2025にアクセス、https://socket.io/docs/v4/
44. Exploring and Controlling Diversity in LLM-Agent Conversation - arXiv, 4月 12, 2025にアクセス、https://arxiv.org/html/2412.21102v1
45. Multi-Agent Large Language Models for Conversational Task-Solving - arXiv, 4月 12, 2025にアクセス、https://arxiv.org/html/2410.22932v1
46. [2503.22458] Evaluating LLM-based Agents for Multi-Turn Conversations: A Survey - arXiv, 4月 12, 2025にアクセス、https://arxiv.org/abs/2503.22458
47. LLM-based Multi-Agent Systems: Techniques and Business Perspectives - arXiv, 4月 12, 2025にアクセス、https://arxiv.org/html/2411.14033v2

48. Beyond Self-Talk: A Communication-Centric Survey of LLM-Based Multi-Agent Systems, 4月 12, 2025にアクセス、https://arxiv.org/html/2502.14321v1
49. Multi-Agent Collaboration Mechanisms: A Survey of LLMs - arXiv, 4月 12, 2025にアクセス、https://arxiv.org/html/2501.06322v1
50. EmoLLMs: A Series of Emotional Large Language Models and Annotation Tools for Comprehensive Affective Analysis - arXiv, 4月 12, 2025にアクセス、https://arxiv.org/html/2401.08508v2
51. Sentiment Analysis: First Steps With Python's NLTK Library - Real Python, 4月 12, 2025にアクセス、https://realpython.com/python-nltk-sentiment-analysis/
52. Sentiment Analysis Using Python - Analytics Vidhya, 4月 12, 2025にアクセス、https://www.analyticsvidhya.com/blog/2022/07/sentiment-analysis-using-python/
53. 6 Must-Know Python Sentiment Analysis Libraries - Netguru, 4月 12, 2025にアクセス、https://www.netguru.com/blog/python-sentiment-analysis-libraries
54. 2.8 Project: Detecting Emotions from Text — Practical NLP with Python - NLPlanet, 4月 12, 2025にアクセス、https://www.nlplanet.org/course-practical-nlp/02-practical-nlp-first-tasks/08-emotion-classification
55. Python Code For Facial Expression Analysis - Restack, 4月 12, 2025にアクセス、https://www.restack.io/p/ai-for-emotion-recognition-answer-python-code-facial-expression-analysis-cat-ai
56. Features and contents of VRM, 4月 12, 2025にアクセス、https://vrm.dev/en/vrm/vrm_features/
57. Getting Google MediaPipe to control VRM Characters - Extra Ordinary, the Series, 4月 12, 2025にアクセス、https://extra-ordinary.tv/2024/01/07/getting-google-mediapipe-to-control-vrm-characters/
58. Text Analysis in Python: Techniques and Libraries Explained - Airbyte, 4月 12, 2025にアクセス、https://airbyte.com/data-engineering-resources/text-analysis-in-python
59. Py-Feat: Python Facial Expression Analysis Toolbox - PMC, 4月 12, 2025にアクセス、https://pmc.ncbi.nlm.nih.gov/articles/PMC10751270/
60. Text Emotion Detection using NLP | Python | Streamlit Web Application - YouTube, 4月 12, 2025にアクセス、https://m.youtube.com/watch?v=J87jV9OGodw
61. Emotion Detection of Text Using Machine Learning and Python - YouTube, 4月 12, 2025にアクセス、https://m.youtube.com/watch?v=t1TkAcSDsI8&pp=ygUVI25vb2lIbW90aW9uZGV0ZWN0aW9u
62. VRMC_vrm: expression | VRM, 4月 12, 2025にアクセス、https://vrm.dev/vrm1/expression/
63. Configure blend shape on VRM [VirtualCast], 4月 12, 2025にアクセス、https://wiki.virtualcast.jp/wiki/en/vrm/setting/blendshap
64. Emo3D: Metric and Benchmarking Dataset for 3D Facial Expression Generation from Emotion Description - arXiv, 4月 12, 2025にアクセス、https://arxiv.org/html/2410.02049v1
65. Emotion-Preserving Blendshape Update With Real-Time Face Tracking - Feng Xu,

4月 12, 2025にアクセス、http://xufeng.site/publications/2020/wzb_Emotion-preserving%20Blendshape%20Update%20with%20Real-time%20Face%20Tracking.pdf
66. Trying to get VRM blendshape to make mouth work, every sliders simply do that. Why does it happen and how can I fix it? : r/vtubertech - Reddit, 4月 12, 2025にアクセス、https://www.reddit.com/r/vtubertech/comments/1aorbuo/trying_to_get_vrm_blendshape_to_make_mouth_work/
67. BlendShape Setup (v0.45) - VRM, 4月 12, 2025にアクセス、https://vrm.dev/en/univrm/blendshape/blendshape_setup/
68. How to Setup and Use OBS WebSocket? - Video SDK, 4月 12, 2025にアクセス、https://www.videosdk.live/developer-hub/websocket/obs-websocket
69. obsws-python - PyPI, 4月 12, 2025にアクセス、https://pypi.org/project/obsws-python/1.1.0/
70. pschmitt/obs-cli: CLI for controlling OBS Studio - GitHub, 4月 12, 2025にアクセス、https://github.com/pschmitt/obs-cli
71. Python Script to connect to OBS Websocket Server Help, 4月 12, 2025にアクセス、https://obsproject.com/forum/threads/python-script-to-connect-to-obs-websocket-server-help.173395/
72. Python OBS-WebSocket-Py Set Source Visibility - Stack Overflow, 4月 12, 2025にアクセス、https://stackoverflow.com/questions/78210213/python-obs-websocket-py-set-source-visibility
73. Elektordi/obs-websocket-py - GitHub, 4月 12, 2025にアクセス、https://github.com/Elektordi/obs-websocket-py
74. google/adk-samples: A collection of sample agents built ... - GitHub, 4月 12, 2025にアクセス、https://github.com/google/adk-samples
75. GoogleCloudPlatform/agent-starter-pack: A collection of production-ready Generative AI Agent templates built for Google Cloud. It accelerates development by providing a holistic, production-ready solution, addressing common challenges (Deployment & Operations, Evaluation, Customization, Observability) in building and deploying GenAI agents. - GitHub, 4月 12, 2025にアクセス、https://github.com/GoogleCloudPlatform/agent-starter-pack
76. Model Context Protocol · GitHub, 4月 12, 2025にアクセス、https://github.com/modelcontextprotocol
77. ObsWs is a API library for accessing OBS WebSocket API through Godot - GitHub, 4月 12, 2025にアクセス、https://github.com/eumario/ObsWs
78. Example Clients - Model Context Protocol, 4月 12, 2025にアクセス、https://modelcontextprotocol.io/clients
79. Build Your First Intelligent Agent Team: A Progressive Weather Bot with ADK - Google, 4月 12, 2025にアクセス、https://google.github.io/adk-docs/get-started/tutorial/
80. Model context protocol (MCP) - OpenAI Agents SDK, 4月 12, 2025にアクセス、https://openai.github.io/openai-agents-python/mcp/
81. Just did a deep dive into Google's Agent Development Kit (ADK). Here are some

thoughts, nitpicks, and things I loved (unbiased) - Reddit, 4月 12, 2025にアクセス、https://www.reddit.com/r/LocalLLaMA/comments/1jvsvzj/just_did_a_deep_dive_into_googles_agent/
82. Thinking if MCP server could work with anki connect? thx - Add-ons, 4月 12, 2025にアクセス、https://forums.ankiweb.net/t/thinking-if-mcp-server-could-work-with-anki-connect-thx/57436
83. Build Multi-Model Agent in 10 Minutes - Google's New Agent Kit Is INSANE! - YouTube, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=SjZG-QKrw5o
84. Google Agent2Agent Protocol (A2A) - Complements Anthropic's MCP - YouTube, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=JlISpEG8_VQ
85. Quickstart: Build an agent with the Agent Development Kit | Generative AI on Vertex AI, 4月 12, 2025にアクセス、https://cloud.google.com/vertex-ai/generative-ai/docs/agent-development-kit/quickstart
86. Google CAN'T STOP COOKING: FREE AI AGENTS SDK JUST DROPPED - YouTube, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=tldS6e5rD8A
87. Introducing Agent Development Kit - YouTube, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=zgrOwow_uTQ
88. The Model Context Protocol (MCP) Explained (and one cool code example.) - YouTube, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=5ZWeCKY5WZE
89. The MCP Integration EVERYONE is Sleeping On (MCP + Custom AI Agents) - YouTube, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=soC4n-nKWF8
90. Building AI Agents with Model Context Protocol: From Specification to Implementation, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=oSGVQIZxi7s
91. three-vrm.md - GitHub Gist, 4月 12, 2025にアクセス、https://gist.github.com/ahuglajbclajep/6ea07f6feb250aa776afa141a35e725b
92. Modeling Response Consistency in Multi-Agent LLM Systems: A Comparative Analysis of Shared and Separate Context Approaches - arXiv, 4月 12, 2025にアクセス、https://arxiv.org/html/2504.07303v1
93. Emotion Analysis AI Model for Sensing Architecture Using EEG - MDPI, 4月 12, 2025にアクセス、https://www.mdpi.com/2076-3417/15/5/2742
94. UCREL/Session-6-LLM-Based-Emotion-Analysis - GitHub, 4月 12, 2025にアクセス、https://github.com/UCREL/Session-6-LLM-Based-Emotion-Analysis
95. Announcing the Agent2Agent Protocol (A2A) - Google for Developers Blog, 4月 12, 2025にアクセス、https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/
96. vrm-specification/specification/VRMC_vrm-1.0/README.md at master · vrm-c/vrm ... - GitHub, 4月 12, 2025にアクセス、https://github.com/vrm-c/vrm-specification/blob/master/specification/VRMC_vrm-1.0/README.md

97. Tutorial - Introduction - Socket.IO, 4月 12, 2025にアクセス、https://socket.io/docs/v4/tutorial/introduction
98. Tutorial - Overview of the API - Socket.IO, 4月 12, 2025にアクセス、https://socket.io/docs/v4/tutorial/api-overview
99. Get started - Socket.IO, 4月 12, 2025にアクセス、https://socket.io/get-started/chat
100. Get started - Socket.IO, 4月 12, 2025にアクセス、https://socket.io/get-started/
101. Issues · aatikturk/obsws-python - GitHub, 4月 12, 2025にアクセス、https://github.com/aatikturk/obsws-python/issues
102. Evaluating LLM-based Agents for Multi-Turn Conversations: A Survey - arXiv, 4月 12, 2025にアクセス、https://arxiv.org/html/2503.22458v1
103. A Survey on LLM-based Multi-Agent System: Recent Advances and New Frontiers in Application - arXiv, 4月 12, 2025にアクセス、https://arxiv.org/html/2412.17481v2
104. Research on map emotional semantics using deep learning approach - ResearchGate, 4月 12, 2025にアクセス、https://www.researchgate.net/publication/368696901_Research_on_map_emotional_semantics_using_deep_learning_approach
105. How to Swap VRoid VRM textures with blendshapes in Unity - YouTube, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=MlzJtdvgdb0
106. three.js Basic Character Controls - GLTFLoader, AnimationMixer, 3rd Person Controller, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=C3s0UHpwlf8
107. Implementation of VRM on Three.js - W3C, 4月 12, 2025にアクセス、https://www.w3.org/2019/09/Meetup/YutakaObuchi_VRM.pdf
108. @pixiv/three-vrm examples - CodeSandbox, 4月 12, 2025にアクセス、https://codesandbox.io/examples/package/@pixiv/three-vrm
109. Lerping camera and lookAt together with dynamic taret (React three fiber) - Questions, 4月 12, 2025にアクセス、https://discourse.threejs.org/t/lerping-camera-and-lookat-together-with-dynamic-taret-react-three-fiber/28896
110. 3DCGソフトウェア - Wikipedia, 4月 12, 2025にアクセス、https://ja.wikipedia.org/wiki/3DCG%E3%82%BD%E3%83%95%E3%83%88%E3%82%A6%E3%82%A7%E3%82%A2
111. Group Emotion Detection Based on Social Robot Perception - MDPI, 4月 12, 2025にアクセス、https://www.mdpi.com/1424-8220/22/10/3749
112. Emotion detection with Python and OpenCV | Computer vision tutorial - YouTube, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=Vq_01gFG2vk
113. Multimodal Feature Learning for Video Captioning - ResearchGate, 4月 12, 2025にアクセス、https://www.researchgate.net/publication/323285196_Multimodal_Feature_Learning_for_Video_Captioning
114. SURP Archives - - Cal Poly College of Engineering, 4月 12, 2025にアクセス、https://ceng.calpoly.edu/surp-archives/

115. UNIVERSAL REGISTRATION DOCUMENT 2023 - Crédit Agricole CIB, 4月 12, 2025にアクセス、https://www.ca-cib.com/sites/default/files/2024-03/URD_CACIB_2023_EN.pdf
116. How to create a Motion Vector map - Questions - three.js forum, 4月 12, 2025にアクセス、https://discourse.threejs.org/t/how-to-create-a-motion-vector-map/75319
117. TensorFlow.js in the browser for expression prediction - Extra Ordinary, the Series, 4月 12, 2025にアクセス、https://extra-ordinary.tv/2024/01/11/tensorflow-js-in-the-browser-for-expression-prediction/
118. README.md - vladmandic/human-three-vrm - GitHub, 4月 12, 2025にアクセス、https://github.com/vladmandic/human-three-vrm/blob/main/README.md
119. Create three.js components with map method - Stack Overflow, 4月 12, 2025にアクセス、https://stackoverflow.com/questions/64956323/create-three-js-components-with-map-method
120. Create a vector map and make user move through with three.js - Stack Overflow, 4月 12, 2025にアクセス、https://stackoverflow.com/questions/18393502/create-a-vector-map-and-make-user-move-through-with-three-js
121. I made a Vtuber Web App with face and full body tracking! : r/VirtualYoutubers - Reddit, 4月 12, 2025にアクセス、https://www.reddit.com/r/VirtualYoutubers/comments/ojmn73/i_made_a_vtuber_web_app_with_face_and_full_body/
122. 1月 1, 1970にアクセス、https://pixiv.github.io/three-vrm/docs/modules/three-vrm