实时换脸与视频合成系统 – 完整技术规范 V3.0 (最终版)

1. 概述与目标

本文档是“实时换脸与视频合成系统”在为期两周的概念验证（POC）冲刺阶段的**最终、统一、完整的技术设计与交互规范**。

本文档是 AI 驱动开发（由 Gemini 完成）的**唯一“契约”**。所有开发工作，包括代码生成、模型配置和自动化测试，都必须严格遵守此文档。

1.1 项目目标

- **核心目标**: 构建一个高性能的实时视频处理服务。
- **开发模式**: 采用**“零人工干预”（Zero Human Intervention, ZHI）**模式。系统将自动下载所需模型、自动运行测试并验证功能。
- **后端 (The Engine)**: 一个 FastAPI 应用，负责所有 AI 计算、资源管理和视频流处理。
- **前端 (The Cockpit)**: 一个 Streamlit 应用，作为纯客户端 UI。其代码将基于本规范中的 **Apple 风格 Figma 设计描述**，由 AI 自动生成。

1.2 项目范围 (POC)

- **包含**: 核心的实时视频处理流（输入、AI 处理、合成、WebRTC 输出）、自动模型下载、自动化单元测试和 E2E 测试。
- **不包含**: 用户认证、视频录制与下载功能。
- **范围变更**:
  - .dfm 到 .onnx 的模型转换工作**不在本项目范围之内**。
  - 换脸模型（.onnx）将**不再由用户上传**，而是由后端服务在启动时**自动从指定 URL 下载**。

1.3 核心技术栈

- **后端框架**: FastAPI
- **前端框架**: Streamlit
- **环境**: Ubuntu 22.04, Python 3.12, CUDA 12.8
- **AI 框架**: PyTorch (CUDA 12.x compatible)
- **推理引擎 (POC)**: ONNX Runtime (GPU)
- **实时通信**: WebRTC (via aiortc)
- **并行计算**: threading, asyncio, Queue
- **测试框架**: pytest (后端单元测试), playwright (端到端 E2E 测试)

第 1 部分：后端 (Engine) 详细设计

2. 系统架构

2.1 宏观架构 (前后端分离)

- **后端 (The Engine)**: 一个运行在 uvicorn 上的 FastAPI 应用。它暴露 REST API 用于控制，并提供 WebRTC 端点用于视频流。
- **前端 (The Cockpit)**: 一个 Streamlit 应用。它作为客户端，通过 HTTP 请求调用后端 API，并通过 WebRTC 接收视频。

2.2 AI 处理流水线 (Pipeline)

这是一个多阶段的处理流，旨在将原始视频帧转换为最终合成的画面。

[视频输入] –> [面部检测] –> [人体抠图] –> [精细面部解析] –> [换脸推理] –> [面部融合] –> [最终合成] –> [WebRTC 输出]

2.3 GPU 并行架构

这是本系统的核心设计。**此并行策略可通过 API 配置 (use_multi_gpu 标志) 来启用或禁用。**

2.3.1 双 GPU 并行架构 (Pipeline Parallelism – use_multi_gpu: true)

为了在 1080p@24fps 下实现复杂处理，我们将计算任务拆分到两块 4090 GPU 上，采用**流水线并行**模式。

- **GPU 0: “AI 计算核心” (Compute Core)**
  - **职责**: 专职执行所有最耗时的 AI 推理任务。
  - **任务**:
    1. 人体抠图 (RVM)
    2. 精细面部解析 (BiSeNet)
    3. 核心换脸推理 (DFL/ONNX)
    4. 面部融合 (Blending)
- **GPU 1: “控制与合成核心” (Control & Composition Core)**
  - **职责**: 负责所有 I/O、轻量级 AI、数据调度和最终合成。
  - **任务**:
    1. **视频流管理 (捕获与分发)**
    2. 面部检测 (RetinaFace)
    3. 最终背景合成 (Alpha Blending)
    4. WebRTC 视频流广播
- **并行实现**:
  - 我们将使用 threading.Thread 创建一个主 ProcessingManager 线程，该线程负责协调整个流水线。
  - 数据将在不同的处理单元之间通过 queue.Queue 传递。
  - **GPU 间数据传输**：我们将采用 **CPU 中介的 GPU 间数据传输**。
    1. 帧 CPU –> GPU 1 (面部检测)
    2. 帧 GPU 1 –> CPU (下载)
    3. 帧 CPU –> GPU 0 (上传)
    4. (GPU 0 内部处理...)
    5. 结果 GPU 0 –> CPU (下载)
    6. 结果 CPU –> GPU 1 (上传)
    7. (GPU 1 内部处理...)

8. 最终帧 GPU 1 -> CPU (下载，用于 WebRTC)

2.3.2 单 GPU 运行模式 (Single-GPU Mode - use_multi_gpu: false)

- **职责**: 如果配置为禁用多 GPU，所有 AI 任务和 I/O 任务将全部在 **GPU 0**（或默认 GPU）上串行执行。
- **影响**:
  - 这将**显著增加单帧延迟**，可能无法达到 24fps 的实时目标。
  - 此模式主要用于功能测试、调试或单显卡环境下的部署。
- **实现**: ProcessingManager 将使用一个简化的、单线程的逻辑，在同一个 GPU 设备上按顺序执行所有处理步骤。

3. 后端核心模块详细设计

3.1 F-SYS: 系统状态管理

- **需求**: 提供系统状态查询 (IDLE, PROCESSING, ERROR)。
- **实现**:
  - 在 FastAPI 主应用中，使用一个全局的 Pydantic 模型作为状态管理器。
  - app.state.status = {"state": "IDLE", "error": None}
  - POST /stream/start 和 POST /stream/stop 将负责原子化地更新此状态。
  - GET /stream/status 将直接读取此状态并返回。
  - **新增**: app.state.status 必须增加一个 models_ready: bool 标志。

3.2 F-FILE: 资产与模型管理

- **需求**:
  1. **自动下载模型**: 后端服务在**首次启动时**必须自动从预设 URL 下载所有必需的 AI 模型（RVM, RetinaFace, BiSeNet, 默认换脸模型）。
  2. **管理用户资产**: 接受前端上传的源人脸图片和背景文件。
- **实现**:
  - **模型自动配置 (F-FILE-AUTO)**:

        - **@app.on_event("startup")**: 注册一个 FastAPI 启动事件。
        - **逻辑**: 该事件将触发一个 utils.download_models()函数。
        - download_models():
            1. 检查 /opt/fusion_assets/models/ 目录。
            2. 如果模型文件（如 rvm.onnx, retinaface.onnx 等）不存在，则从 config.py 中定义的 URL（例如 Hugging Face）下载。
            3. 下载时应显示进度条（打印到日志）。
            4. 下载完成后，设置 app.state.status["models_ready"] = True。
    - **用户资产上传 (F-FILE-USER)**:
        - POST /files/upload/face: 接受源人脸图片。
        - POST /files/upload/background: 接受背景图片/视频。
        - **已移除**: POST /files/upload/model 端点**已被移除**。

3.3 F-PIPE: AI 处理流水线控制

- **需求**: 通过 API 启动和停止流水线，并按配置运行。
- **实现**:
    - **ProcessingManager 类**: 这是后端的核心。
    - POST /stream/start:
        - **前置检查**: 必须检查 app.state.status["models_ready"] == True。如果模型未就绪，返回 503 Service Unavailable。
        - 从请求体 (JSON) 中解析配置, **包括 use_multi_gpu 标志**和 input_source。
        - app.state.manager = ProcessingManager(config, use_multi_gpu=config.get('use_multi_gpu', True))
        - 调用 app.state.manager.start()。
        - 设置系统状态为 PROCESSING。
    - POST /stream/stop:
        - 调用 app.state.manager.stop()。
        - 线程退出后，app.state.manager = None。

        - 设置系统状态为 IDLE。

3.4 F-VID: 视频流管理 (输入与输出)

- **模块: 视频输入 (Video Input)**
    - **职责**: 根据配置，从三个不同的源捕获视频。
    - **实现**: ProcessingManager 将根据 input_source.type 启动三个不同的捕获逻辑之一:
        1. **"type": "webrtc_client"**:
            - **技术**: aiortc (后端) + streamlit-webrtc (前端)。
            - **逻辑**: 后端在 WebRTC 连接上**接收**一个 MediaStreamTrack。一个 asyncio 任务将 await track.recv() 循环获取 av.VideoFrame，并将其放入 AI 流水线队列。
        2. **"type": "local_cam"**:
            - **技术**: OpenCV (cv2.VideoCapture)。
            - **逻辑**: 后端启动一个专用 threading.Thread，循环调用 cv2.VideoCapture(input_source.id).read()。
        3. **"type": "rtsp"**:
            - **技术**: OpenCV (cv2.VideoCapture)。
            - **逻辑**: 与 local_cam 类似，但捕获目标是 input_source.url。
- **模块: 视频输出 (Video Output)**
    - **职责**: 将最终合成的帧通过 WebRTC 广播。
    - **实现**:
    - **库**: aiortc (用于 Python 的 WebRTC 实现)。
    - **逻辑**: FastAPI 将提供一个信令端点（例如/sdp）。
    - ProcessingManager 将创建一个自定义的 aiortc.MediaStreamTrack。最终合成的帧（Numpy 数组）将被转换为 av.VideoFrame 并推送到这个 Track 中。

3.5 后端核心 AI 模块详细规格

- **模块 3.5.1: 面部检测 (Face Detection)**

    - **GPU**: **GPU 1** (或 GPU 0，在单卡模式下)
    - **模型**: **RetinaFace (MobileNet backbone)** (ONNX 版)
    - **实现**:
        - 使用 onnxruntime-gpu (CUDA 12.x 版) 加载 ONNX 模型。
        - **优化**: 实现**“检测+跟踪”**逻辑（例如：每 3 帧检测一次，中间 2 帧使用 cv2.Tracker 跟踪）。
- **模块 3.5.2: 人体抠图 (Human Matting)**
    - **GPU**: **GPU 0**
    - **模型**: **RVM (Robust Video Matting) (ONNX 版)**
    - **实现**:
        - 使用 onnxruntime-gpu 加载模型。
        - **优化**: RVM 需要循环状态（r1i, r2i...），这些状态张量需要在帧之间保持和传递。
- **模块 3.5.3: 精细面部解析 (Fine Face Parsing)**
    - **GPU**: **GPU 0**
    - **模型**: **BiSeNet (Bilateral Segmentation Network)** (轻量级 ONNX 版)
    - **实现**:
        - 在**裁剪出的面部**上运行 BiSeNet 推理。
        - **蒙版生成**: 将此分割图转换为一个**精细的融合蒙版**（例如，只包含皮肤，并羽化边缘）。
- **模块 3.5.4: 核心换脸 (Core Face Swapping)**
    - **GPU**: **GPU 0**
    - **模型**: **DFL/SAEHD (已转换为 .onnx)**
    - **实现**:
        - **源人脸**: 源人脸的特征向量（embedding）应在流水线启动时**预先计算并缓存**在 GPU 0 显存中。
        - **推理**: 使用 onnxruntime-gpu 执行换脸模型。
- **模块 3.5.5: 面部融合 (Face Blending)**

    - **GPU**: **GPU 0**
    - **实现**:
        - **输入**: 1. fgr (来自 RVM), 2. “已交换”的面部 (来自 DFL), 3. 精细融合蒙版 (来自 BiSeNet)。
        - **逻辑**: 使用精细蒙版，将“已交换”的面部无缝地粘贴（Alpha Blending）到 fgr（抠出的人物前景）上。
- **模块 3.5.6: 最终合成 (Final Composition)**
    - **GPU**: **GPU 1** (或 GPU 0，在单卡模式下)
    - **实现**:
        - 从背景源（可能是视频文件或图片）获取当前背景帧 bgr。
        - **逻辑**: 在 GPU 1 上执行最终的 Alpha 混合：Final_Image = final_fgr * pha + bgr * (1.0 – pha)。
- **模块 3.5.7: (到 3.4) WebRTC 输出**
    - **GPU**: **GPU 1** (或 GPU 0) –> **CPU**
    - **实现**:
        - 将 Final_Image 从 GPU 显存下载到 CPU 内存（Numpy 数组）。
        - 将 Numpy 数组（BGR 格式）转换为 av.VideoFrame（YUV420p 格式）。
        - 推送 av.VideoFrame 到 aiortc 的 MediaStreamTrack。

第 2 部分：前端 (Cockpit) 功能设计

4. 前端 (Streamlit) 功能设计

4.1 UI 布局

- **设计理念**: 遵循 Apple 的“Human Interface Guidelines”原则，注重清晰、简洁和空间感。
- **布局**:
    - **侧边栏 (st.sidebar)**: 作为控制面板。
    - **主区域 (st.container)**: 作为内容显示区。

4.2 侧边栏 (Sidebar) 详细功能

F-FE-01: 资产上传模块

- **控件**:
    1. **上传源人脸**: st.file_uploader (接受 .png, .jpg)。
    2. **上传背景**: st.file_uploader (接受 .png, .jpg, .mp4)。
- **逻辑**: 文件上传后，立即 POST 到后端。成功后 st.rerun()刷新文件列表。

F-FE-02: 流水线配置模块

- **逻辑**: 所有下拉菜单选项通过 GET /files/list 动态获取。
- **控件**:
    1. st.radio("输入源", ["客户端摄像头 (WebRTC)", "服务器本地摄像头", "网络流 (RTSP)"])
    2. (根据选择动态显示 st.number_input 或 st.text_input)
    3. st.selectbox("源人脸", [从 faces 列表填充])
    4. st.selectbox("换脸模型", [从 models 列表填充])
    5. st.selectbox("背景", [从 backgrounds 列表填充])
    6. st.checkbox("启用双 GPU 并行处理", value=True)

F-FE-03: 主控制模块

- **控件**: st.button("启动流水线") 和 st.button("停止流水线")。
- **状态管理**:
    - st.session_state 跟踪从 API 轮询到的 system_status。
    - **"启动" 按钮**: 仅在 system_status["state"] == "IDLE" **且** system_status["models_ready"] == True 时**启用**。
    - **"停止" 按钮**: 仅在 system_status["state"] == "PROCESSING" 时**启用**。
- **逻辑**:
    - **"启动"**: 收集所有配置，构建 JSON，POST /stream/start。
    - **"停止"**: POST /stream/stop。

4.3 主区域 (Main Area) 详细功能

F-FE-04: 状态显示模块

- **逻辑**: 自动轮询 GET /stream/status (每 2 秒)。
- **控件**:
  - **st.info / st.warning / st.error**:
  - models_ready == False: 显示 st.warning("系统正在初始化：下载 AI 模型中...")。
  - state == "IDLE": 显示 st.info("系统空闲，请在侧边栏配置并启动。")
  - state == "PROCESSING": 显示 st.success("处理中...") (使用绿色指示灯风格)。
  - state == "ERROR": 显示 st.error(f"错误: {error_message}")。

F-FE-05: 视频显示模块

- **逻辑**: 使用 streamlit-webrtc 库。
- **控件**: webrtc_streamer()
- **配置**:
  - key: "webrtc-streamer"
  - mode: WebRtcMode.SENDRECV
  - async_processing: True
  - offer_to_receive_video: True
  - offer_to_receive_audio: False (POC 阶段不处理音频)
- **行为**:
  - 当输入源为 "客户端摄像头": 捕获用户本地摄像头**发送**，并**接收**处理后的流进行显示。
  - 当输入源为 "服务器" 或 "RTSP": **不发送**流，仅**接收**处理后的流进行显示。

4.4 [重写] UI/UX 设计规范 (Apple 风格)

- **免责声明**: AI (Gemini) 无法直接生成 .fig 文件。本规范用于指导 AI 生成一个符合 Apple 设计语言的 Streamlit 应用。
- **核心原则**:
  1. **清晰 (Clarity)**: 字体易读，图标表意明确，控件功能唯一。
  2. **空间 (Depth)**: 利用留白和模糊效果创造层次感。

3. **简洁 (Deference)**: UI 应退居次要地位，让视频内容成为焦点。

- **风格指南**:
  - **字体 (Typography)**: 依赖系统默认字体。Streamlit 应配置为使用 font-family: -apple-system, BlinkMacSystemFont, "Inter", "Segoe UI", "Roboto", "Helvetica Neue", sans-serif;
  - **圆角 (Rounded Corners)**: 所有关键元素（按钮、输入框、视频窗口）应使用统一的圆角，建议 border-radius: 8px。
  - **毛玻璃效果 (Translucency)**:
    - **侧边栏**: 建议使用 st.set_page_config(layout="wide") 并通过自定义 CSS 注入，使侧边栏背景色带有透明度 (e.g., rgba(240, 240, 240, 0.8)) 并启用背景模糊 (backdrop-filter: blur(20px))，以模仿 macOS 的菜单栏效果。
  - **颜色 (Color)**:
    - **主色调**: 采用中性色（黑、白、灰）。
    - **强调色**: 采用单一、鲜艳的强调色（如 Apple 的蓝色）用于按钮和活动指示器。
    - **状态色**: st.success (绿色), st.warning (黄色), st.error (红色) 用于状态显示。
  - **图标 (Icons)**:
    - 按钮和状态指示器应使用简洁的线条图标 (SVG)，风格参考 Apple SF Symbols。例如 "启动" (▶)，"停止" (■)，"处理中" (一个绿点)。
- **布局描述 (同 V2.0)**:
  1. **侧边栏 (Sidebar)**: 宽度固定 (e.g., 300px)，应用上述毛玻璃效果。
  - **Section 1: 资产上传**: (F-FE-01)
  - **Section 2: 流水线配置**: (F-FE-02)
  - **Section 3: 主控制**: (F-FE-03)
  2. **主区域 (Main Area)**:
  - **Top: 状态栏**: (F-FE-04)
  - **Bottom: 视频窗口**: (F-FE-05) streamlit-webrtc 组件，占满剩余空间，并应用圆角。

第 3 部分：API 与数据交换契约

5. 核心交互与数据交换

5.1 状态管理流程 (State Management Flow)

- **变更**:

1. **FE (前端)**: 用户打开网页。
2. **FE**: 立即开始轮询 GET /stream/status。
3. **BE (后端)**: 启动时，startup_event 触发，download_models()开始执行。
4. **BE**: 返回 {"state": "IDLE", "error_message": null, "models_ready": false}。
5. **FE**: 显示 "系统正在初始化：下载 AI 模型中..."。所有控件禁用。
6. (后端模型下载完成...)
7. **BE**: 设置 app.state.status["models_ready"] = True。
8. **FE**: 轮询 GET /stream/status 得到 {"models_ready": true}。
9. **FE**: 立即调用 GET /files/list。
10. **BE**: 返回 {"faces": [], "models": ["rvm.onnx", ...], "backgrounds": []}。
11. FE: 填充下拉菜单，启用"启动"按钮和上传控件。

    (... 后续流程同 V1.0 ...)

5.2 REST API 详细定义 (JSON 契约)

GET /stream/status

- **成功响应 (200 OK) (变更)**:

  ```
  {
    "state": "IDLE" | "PROCESSING" | "ERROR",
    "error_message": "string | null",
    "models_ready": "boolean"
  }
  ```

GET /files/list

- **成功响应 (200 OK) (变更)**:

  {

  "faces": ["source_face_01.png", ...],

  "models": ["rvm.onnx", "retinaface.onnx", "bisenet.onnx", "dfl_saehd.onnx"],

  "backgrounds": ["bg1.png", ...]

  }

POST /files/upload/face

- **请求**: multipart/form-data, name="file"
- **成功响应 (200 OK)**: {"message": "File 'face.png' uploaded successfully.", "filename": "face.png"}

POST /files/upload/background

- **请求**: multipart/form-data, name="file"
- **成功响应 (200 OK)**: {"message": "File 'bg.mp4' uploaded successfully.", "filename": "bg.mp4"}

POST /stream/start

- **请求体 (Request Body) - application/json**:

  {

  "input_source": {

  "type": "webrtc_client" | "local_cam" | "rtsp",

  "id": "integer | null",

  "url": "string | null"

  },

  "face_id": "string",

  "model_id": "string",

  "background_id": "string",

```
  "use_multi_gpu": "boolean"
}
```

- **成功响应 (200 OK)**: {"message": "Stream started", "state": "PROCESSING"}

POST /stream/stop

- **请求**: (无)
- **成功响应 (200 OK)**: {"message": "Stream stopped", "state": "IDLE"}

5.3 WebRTC 数据流契约

1. **信令 (Signaling)**:
    - 后端 FastAPI 必须提供一个用于 WebRTC 信令交换的端点 (例如 POST /sdp)。
    - 前端 streamlit-webrtc 将自动处理与此端点的 HTTP 请求/响应，以协商 (Offer/Answer)连接。
2. **输入数据流 (FE -> BE)**:
    - **条件**: 仅当 POST /stream/start 请求中的 input_source.type == "webrtc_client" 时触发。
    - **格式**: 前端发送一个 H.264 (或其他标准) 编码的视频流。
    - **后端职责**: 后端 aiortc 服务必须准备好**接收**此视频流，将其解码为 av.VideoFrame，然后转换为 numpy 数组，送入 AI 流水线。
3. **输出数据流 (BE -> FE)**:
    - **条件**: 只要流水线处于 PROCESSING 状态，就必须触发。
    - **格式**: 后端 AI 流水线 (模块 3.5.7) 生成 numpy 数组 (1080p, BGR)。
    - **后端职责**: 后端必须将此 numpy 数组转换为 av.VideoFrame (YUV420p 格式)，并将其**发送**到 aiortc 的输出 MediaStreamTrack。
    - **前端职责**: 前端 streamlit-webrtc 控件将自动**接收**此视频流并显示。

第 4 部分：自动化配置与测试 (ZHI)

为实现“零人工干预”的 POC 验证，AI (Gemini) 必须生成以下自动化脚本。

6.1 自动化模型下载

- **后端实现**: backend/app_startup.py
  - 将包含一个 download_model_from_url(url, path) 函数。
  - 将包含一个 async def on_startup() 事件处理器。
- **配置文件**: backend/config.py
  - 将包含一个 MODEL_URLS 字典，映射模型名称到下载 URL。
  - *示例 (URL 为占位符)*:

    MODEL_URLS = {

    "rvm.onnx":

    "https://huggingface.co/models/rvm.onnx",

    "retinaface.onnx":

    "https://huggingface.co/models/retinaface.onnx",

    "bisenet.onnx":

    "https://huggingface.co/models/bisenet.onnx",

    "dfl_saehd.onnx":

    "https://huggingface.co/models/dfl_saehd.onnx"

    }

6.2 自动化测试

AI (Gemini) 将生成以下测试文件：

6.2.1 后端单元测试 (pytest)

- **文件**: backend/test_api.py

- **职责**: 使用 pytest 和 httpx 对 FastAPI 应用进行单元测试（不依赖 AI 流水线）。
- **测试用例**:
    - test_get_status_idle(): 验证启动时状态为 IDLE 和 models_ready=False (在 on_startup 完成前)。
    - test_file_uploads(): 测试 /files/upload/face 和 /files/upload/background 端点。
    - test_get_file_list(): 测试 /files/list 是否能正确返回上传的文件。
    - test_stream_start_stop(): (使用 mock) 测试 /stream/start 和 /stream/stop 能否正确改变系统状态。

6.2.2 端到端 (E2E) 测试 (playwright)

- **文件**: tests/test_e2e.py
- **职责**: 模拟真实用户，对**完整运行的前后端系统**进行自动化 E2E 测试。
- **测试流程 (test_full_pipeline)**:
    1. **启动**: playwright 启动浏览器并打开 Streamlit 应用的 URL。
    2. **等待模型下载**: 轮询页面，直到 "系统正在初始化..." 消息消失，且 "启动" 按钮变为可用。
    3. **上传资产**:
        - 使用 page.set_input_files() 自动化上传一个测试用的人脸图片 (tests/assets/test_face.png)。
        - 自动化上传一个测试用的背景图片 (tests/assets/test_bg.png)。
    4. **配置流水线**:
        - 验证 "源人脸" 下拉菜单中出现了 test_face.png 并选中它。
        - 验证 "背景" 下拉菜单中出现了 test_bg.png 并选中它。
        - 验证 "换脸模型" 下拉菜单中自动填充了模型 (e.g., dfl_saehd.onnx) 并选中它。
    5. **启动**: 点击 "启动流水线" 按钮。
    6. **验证处理中**:
        - 验证页面状态变为 "处理中..."。

    - 验证 "启动" 按钮被禁用，"停止" 按钮被启用。
    - **关键**: 验证 streamlit-webrtc 的视频播放器元素已加载并可见。
7. **停止**: 等待 5 秒后，点击 "停止流水线" 按钮。
8. **验证空闲**: 验证页面状态变回 "系统空闲..."，"启动" 按钮重新启用。