玻璃幕墙缺陷智能检测系统
软件设计文档

版本：v1.1

日期：2025.1.2

作者：陈柏熙、黄景胤、王雷、袁艺铭

SDD 修订记录表

名称	版本号	时间	撰写人	评审人
玻璃幕墙缺陷智能检测系统软件设计文档	V1.0	2025.12.31	陈柏熙、王雷、袁艺铭、黄景胤	
玻璃幕墙缺陷智能检测系统软件设计文档	V1.1	2025.1.2	陈柏熙、王雷、袁艺铭、黄景胤	
玻璃幕墙缺陷智能检测系统软件设计文档				

同濟大學
TONGJI UNIVERSITY

目录

1. 引言

1.1. 文档目的

本文档为幕墙玻璃缺陷智能检测系统的软件设计说明书（Software Design Document, SDD），用于描述系统在需求分析完成之后形成的软件设计模型。文档重点说明系统的总体架构、数据设计、接口设计、构件级设计以及部署设计，为系统的实现、测试与后续维护提供统一、可追溯的设计依据。

本说明书主要面向系统的开发人员、测试人员以及维护人员，作为系统设计与实现过程中的指导性文档。通过本设计说明书，可以确保系统的实现符合既定需求，降低开发过程中的不确定性，提高软件系统的可维护性、可扩展性和可靠性。

1.2. 文档范围

本系统的设计严格依据《软件需求规格说明书》确定的业务与性能指标，结合选定的技术框架与开发手册规范，并充分考量网络环境与数据传输要求进行编制。本设计说明书描述的是玻璃幕墙缺陷智能检测系统的软件设计方案。该系统的设计内容来源于已完成的软件需求规格说明书，涵盖系统的主要功能需求及非功能需求。

本 SDD 重点说明系统的总体结构设计、模块划分思路、数据设计以及接口设计方法，明确各组成部分之间的关系与协作方式。文档不涉及具体的源代码实现细节，而是从设计层面对系统进行抽象描述，为后续的软件实现提供清晰、统一的设计依据。

1.3. 定义、缩写与术语

本节对软件设计文档中涉及的专业术语、缩写和特定概念进行统一定义，以避免歧义，确保文档理解的一致性。

1.3.1. 业务与领域术语

幕墙（Curtain Wall）

建筑外围护结构的一种形式，不承重，通常由玻璃、金属框架等构成。本系统的检测对象为建筑玻璃幕墙。

玻璃幕墙缺陷（Defect of Glass Curtain Wall)

指玻璃幕墙在施工或使用过程中产生的质量问题，包括裂纹、不平整度超标等。

裂纹（Crack）

玻璃表面出现的线状或网状破坏特征，可能影响幕墙结构安全和外观质量。

自爆

不平整度（Flatness Deviation）

描述玻璃幕墙表面相对于理想平面的偏离程度，通常通过点云数据计算得到定量指标。

检测流水线（Detection Pipeline）

由多个数据处理构件按顺序组成的处理流程，用于完成从原始数据输入到检测结果输出的全过程。

检测结果（Detection Result）

系统对输入数据进行分析后得到的结构化输出，包含检测状态、定性描述、定量指标及可视化结果。

1.3.2. 数据与算法相关术语

图像数据（Image Data）

由图像采集设备获取的二维图像数据，包括裂纹检测图像和双目结构光图像。

结构光（Structured Light）

通过向被测物体投射特定光学图案，并结合成像结果进行三维测量的光学方法。本系统用于玻璃幕墙不平整度检测。

双目视觉（Stereo Vision）

通过左右两个相机获取图像，利用视差信息进行三维重建的方法。

差分图（Difference Image）

由“有光图像”和“无光图像”配准并相减得到的图像，用于增强结构光信息并抑制环境光干扰。

角点（Corner Point）

图像中局部亮度变化显著的特征点，在本系统中主要用于结构光标定与立体匹配。

亚像素角点（Sub-pixel Corner）

通过数值优化方法对角点位置进行精细化计算，使其定位精度优于单个像素。

点云（Point Cloud）

由大量三维空间点组成的数据集合，表示玻璃幕墙表面的三维形态。

平面拟合（Plane Fitting）

利用最小二乘等方法，对点云数据拟合得到理想参考平面，用于后续偏差分析。

偏差分析（Deviation Analysis）

将实际点云与参考平面进行比较，计算表面偏离程度的过程。

1.3.3. 指标与统计术语

RMS（Root Mean Square，均方根偏差）

用于衡量点云相对于参考平面的整体偏差程度，是不平整度检测的重要定量指标。

最大偏差（Maximum Deviation）

点云中与参考平面距离最大的偏差值。

平均偏差（Mean Deviation）

点云偏差值的算术平均，用于描述整体偏移趋势。

P95（95% 分位数偏差）

点云偏差分布中 95% 数据点不超过的偏差值，用于衡量整体偏差水平并抑制极端异常点影响。

1.3.4. 软件与架构相关术语

SDD（Software Design Document）

软件设计说明书，用于描述系统在需求分析之后形成的软件设计模型，是系统实现和维护的重要依据。

SRS（Software Requirements Specification）

软件需求规格说明书，用于描述系统的功能需求和非功能需求，是本设计文档的输入依据。

数据流体系结构（Data-Flow Architecture）

以数据流动和处理阶段为核心的软件体系结构风格。

管道-过滤器架构（Pipe-and-Filter Architecture）

一种数据流架构模式，其中过滤器表示数据处理构件，管道表示构件之间的数据传输路径。

构件（Component）

具有明确职责、输入和输出的软件单元，在本系统中主要指数据处理流水线中的功能模块。

接口契约（Interface Contract）

对构件之间交互数据格式、语义和异常处理规则的统一约定。

1.3.5. 技术与实现相关缩写

HTTP（HyperText Transfer Protocol）

系统各层之间进行通信所采用的应用层协议。

JSON（JavaScript Object Notation）

系统内部和对外接口统一使用的数据交换格式。

ROI（Region of Interest）

图像中的感兴趣区域，用于减少无关区域对检测算法的干扰。

CPU（Central Processing Unit）

系统当前基线实现中用于执行算法计算的主要计算资源。

1.3.6. 规范与标准术语

JGJ/T 139-2020

《玻璃幕墙工程质量检验标准》，用于约束不平整度检测指标的工程规范。

GB/T 21086-2007

《建筑幕墙》国家标准，用于指导幕墙工程质量验收。

2. 设计总体说明

2.1. 设计目标与原则

2.1.1. 设计目标

本系统的软件设计以需求规格说明书为依据，围绕幕墙玻璃缺陷检测的业务流程与数据处理特点，确立以数据流驱动和构件化设计为核心的软件设计目标。系统设计的主要目标包括：

1. **数据流清晰化目标**

   通过明确数据在系统中的流转路径和处理阶段，将系统建模为由多个数据处理构件组成的处理管道，避免数据处理过程混乱和模块职责交叉。

2. **构件职责单一化目标**

   每个系统构件仅承担一个明确的数据处理职责，如图像预处理、裂纹检测、点云生成或平整度分析，从而降低系统复杂度。

3. **算法可替换性目标**

   系统在设计层面支持不同检测算法和处理方法的替换与扩展，在不改变整体数据流结构的前提下实现算法升级。

4. **系统可维护性与可扩展性目标**

   通过稳定的数据模型和清晰的模块边界，使系统能够在后续维护和功能扩展过程中保持良好的可理解性和可演化性。

5. **需求一致性与可追溯性目标**

   系统设计应能够清晰映射需求模型中的功能与数据要求，为设计评审和系统验证提供依据。

2.1.2. 设计原则

结合系统的数据处理特性，软件设计遵循以下原则：

1. **数据流导向原则**

   系统以数据流为核心组织方式，数据按照预定义的处理顺序在构件之间传递，构件之间不共享内部状态。

2. **构件化与单一职责原则**

   系统被划分为多个松耦合的数据处理构件，每个构件仅完成单一处理阶段的功能，避免功能重叠。

3. **高内聚、低耦合原则**

   构件内部聚焦于自身的数据处理逻辑，构件之间仅通过明确的数据接口进行交互。

4. **抽象与信息隐藏原则**

   构件对外仅暴露必要的数据输入与输出接口，隐藏内部处理细节和算法实现。

5. **面向变化的设计原则**

   系统在设计时充分考虑算法替换、参数调整和处理流程变化等需求，使整体结构具有良好的适应性。

2.2. 设计方法与建模手段

2.2.1. 设计方法

本系统采用以模型为中心的软件设计方法。在此设计中，核心模型为“数据流处理模型”与“构件化流水线模型”。设计过程以需求分析阶段形成的数据流图为输入，通过自顶向下的方式逐层细化系统结构。

首先，从系统整体层面确定系统的处理流程和主要子系统划分；随后，将各处理阶段细化为独立的数据处理构件模型，明确构件的职责、输入与输出；最终形成可用于指导实现的软件设计模型。

在整个设计过程中，系统设计始终围绕数据流模型展开，避免过早引入具体实现技术，从而保证设计模型的抽象性和稳定性。

2.2.2. 建模手段

为描述系统的结构与行为，本系统在设计阶段采用以下建模手段：

1. **体系结构建模**

   通过体系结构图对系统的整体处理流程、构件划分以及数据流向进行建模，描述系统的总体组织方式。

2. **构件级建模**

   通过构件图描述各数据处理构件的职责及其相互关系，用于表达系统的静态结构。

3. **数据模型建模**

   通过对核心数据类型及其关系的建模，明确系统中原始数据、中间数据和结果数据的逻辑结构。

4. **行为过程建模**

   通过活动图或顺序图描述数据在各处理阶段中的流转过程，帮助理解构件之间的协作关系。

3. 体系结构设计

3.1. 体系结构设计方法说明

本系统采用数据流体系结构（Data-Flow Architecture），具体实现形式为管道-过滤器（Pipe-and-Filter）架构模式。

该体系结构适用于输入数据需要经过一系列独立处理步骤逐步转换为输出数据的应用场景。本系统中，无论是裂纹检测还是玻璃幕墙平整度检测，均表现为典型的数据处理流水线：原始图像作为输入，经多阶段计算与分析后，生成结构化检测结果与最终检测报告。

在该架构中：

1. 过滤器（Filter） 表示独立的数据处理构件；
2. 管道（Pipe） 表示在构件之间传输的中间数据；
3. 各过滤器之间不存在对彼此内部实现的依赖，仅依赖输入输出数据格式。

3.2. 系统上下文建模

在体系结构设计初期，首先明确系统与外部实体之间的交互关系。该信息直接来源于 SRS 中的用例、活动图与用户故事。

3.2.1. 外部实体识别

系统涉及的外部实体包括：

1. **用户（物业维护人员、检测工程师、工程验收人员）**

   通过 Web 前端与系统交互，触发检测流程并查看结果。

2. **图像采集设备**

产生裂纹检测图像与双目结构光图像。

3. **结构光投影与双目相机系统**

为不平整度检测提供原始输入数据。

4. **数据存储系统（文件系统 / 数据库）**

为系统提供数据持久化与历史回查能力。

3.2.2. 上下文交互说明

采集系统定期推送设备数据至本系统，包括区域信息以及原始拍摄图片等，操作人员通过系统界面进行操作指令，经过缺陷检测系统的内部处理，向用户输出检测结果与最终检测报告，同时将处理后的信息按照数据模型写入数据库，保证数据完整性和一致性。

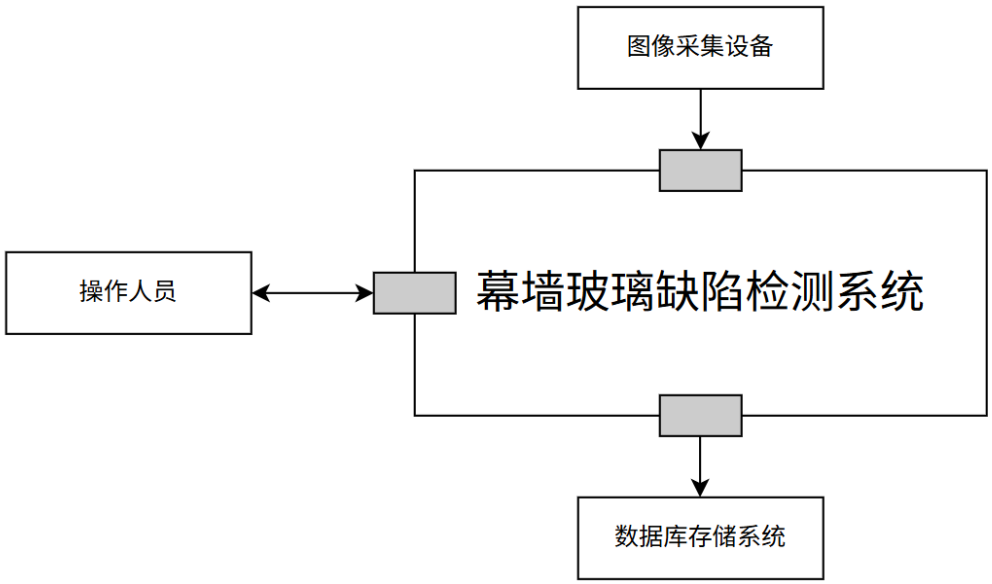


图 3.1 体系结构上下文图

3.3. 体系结构原型设计

在数据流体系结构下，系统的核心行为可抽象为一组稳定的体系结构原型。这些

原型不是具体实现构件，而是描述系统在架构层面的核心职责角色。

结合系统需求，定义如下原型：

1. **数据输入原型（Data Source）**

   表示所有外部输入数据的抽象来源，包括裂纹图像与双目结构光图像。

2. **数据预处理原型（Preprocessor）**

   表示对输入数据进行初步处理的抽象，如区域提取、差分计算等。

3. **分析处理原型（Analyzer）**

   表示核心检测与计算过程，包括裂纹检测、不平整度分析与指标计算。

4. **结果整合原型（Aggregator）**

   表示对多个检测结果进行汇总与结构化的处理过程。

5. **输出与归档原型（Sink）**

   表示结果输出、报告生成与数据归档行为。

这些原型构成了系统数据流体系结构的稳定骨架。

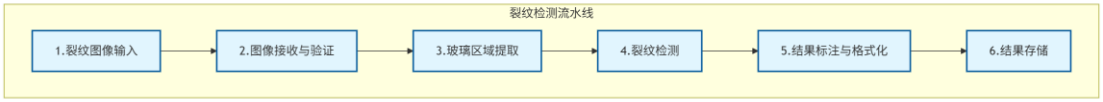
3.4. 架构向构件的细化

在上述原型基础上，体系结构被进一步细化为可实现的软件构件。各构件在数据流中作为独立的过滤器存在。具体构件级设计模式将会在 **6.构件级设计**部分进行详细描述。

3.4.1. 裂纹检测流水线构件

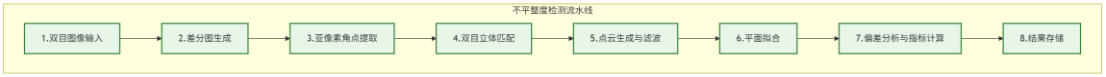
1. 裂纹图像输入构件
2. 图像接收与验证构件
3. 玻璃区域提取构件
4. 裂纹检测构件（传统算法 / 深度学习）

5. 裂纹结果标注与格式化构件
6. 裂纹检测结果存储构件



3.4.2. 不平整度检测流水线构件

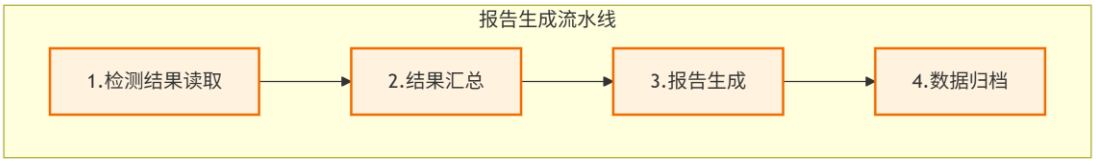
1. 双目结构光图像输入构件
2. 差分图生成构件
3. 亚像素角点提取构件
4. 双目立体匹配构件
5. 点云生成与滤波构件
6. 平面拟合构件
7. 偏差分析与指标计算构件
8. 不平整度检测结果存储构件



3.4.3. 报告生成流水线构件

1. 检测结果读取构件
2. 结果汇总构件
3. 报告生成构件
4. 数据归档构件

各构件通过明确定义的数据接口进行连接，形成完整的数据处理管道。



3.5. 架构实例化说明

在本系统的具体实例中，数据流体系结构被实例化为三条主要处理流水线：

1. 裂纹检测数据流
2. 不平整度检测数据流
3. 检测结果汇总与报告生成数据流
4. 预警

两条流水线在逻辑上相互独立，但在最终阶段通过结果汇总构件进行整合，满足系统对并行检测与统一输出的需求。

预警分支

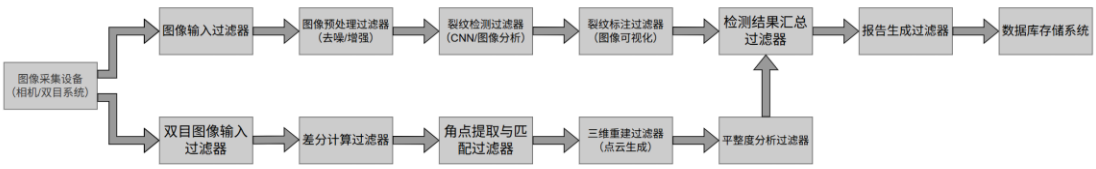


图 3.2 数据流体系结构

4. 构件级设计

本章以“流水线构件”视角描述系统设计，同时结合本项目实际代码结构（前端 React、后端 Spring Boot、算法服务 FastAPI、算法模块 Python）给出可落地的构件划分与接口定义。对“规划中但当前未实现/未独立拆分”的构件，会在表格中标注为“可选/预留”，不影响当前系统基线功能。

4.1. 构件定义与职责

4.1.1. 构件清单（系统级）

系统按部署与职责可划分为四类构件：

- 前端展示构件：负责图像上传、发起检测请求、展示结果与可视化
- 后端接入构件：负责对外 API、文件接收、请求转发、统一响应
- 算法服务构件：负责对外算法 API、输入校验/编排、结果封装
- 算法处理构件：负责核心算法流程（裂纹/平整度）

4.1.2. 裂纹检测流水线构件：职责/输入输出/依赖

构件名称	职责描述	输入（数据类型）	输出（数据类型）	依赖构件	错误处理机制（概要）
裂纹图像输入构件（前端）	选择/预览裂纹图片，发起检测请求	`File`（浏览器文件对象）	`FormData`（multipart）	UI组件、HTTP客户端	前端捕获网络异常/超时，返回"上传失败"提示
图像接收与转发构件（后端）	接收上传文件、写临时文件、转发算法服务、返回统一结果	`MultipartFile[] images`	`DetectionResult`（JSON）	算法服务接口（HTTP）	发生异常返回 `status="error"` 的 `DetectionResult`
图像接收构件（算法服务）	接收 `UploadFile`，读取字节流并交给裂纹服务	`UploadFile image`	`dict`（DetectionResult结构）	裂纹服务构件	try/except 捕获异常并返回错误结构（截断错误信息）
裂纹服务构件（算法服务）	bytes→data URI，调用算法主流程	`bytes` / `str`	`dict`（DetectionResult结构）	裂纹检测算法主流程	转换失败/类型识别失败返回异常或错误结构
图像预处理构件（算法模块）	resize、去噪、灰度化、增强	`np.ndarray`（BGR）	`np.ndarray`（灰度）	配置参数	输入图像为空/尺寸异常触发异常，由上层捕获
裂纹特征提取构件（算法模块）	Canny边缘、裂纹占比、GLCM特征、自爆轮廓特征	`np.ndarray`（灰度/边缘）	`features: dict`	OpenCV、skimage	找不到轮廓时返回 False；其余异常上抛
分类决策构件（算法模块）	基于特征输出 `status/description`	`features: dict`	`(status, desc, damage_area)`	配置参数	规则分支，默认返回 success；异常上抛
结果格式化构件（算法模块）	将检测状态/描述/细节封装为统一输出	`(status, desc, ...)`	`dict`（DetectionResult）	统一结果契约	发生异常返回 `status="error"` 的错误结构
裂纹结果存储构件（可选/预留）	持久化检测结果与产物（DB/文件/对象存储）	`DetectionResult` + 产物	`id/url/path`	存储服务	当前基线未提供统一持久化接口，可扩展
玻璃区域提取构件（可选/预留）	提取玻璃 ROI/Mask 降噪	图像	ROI/Mask	预处理构件	当前基线未独立实现，可扩展
深度学习推理构件（可选/预留）	使用检测/分割模型提升鲁棒性	图像/ROI	裂纹掩膜/框	GPU/模型文件	当前代码中相关逻辑为注释/未启用，可扩展

表 4.1 裂纹检测流水线构件表

4.1.3. 不平整度检测流水线构件：职责/输入输出/依赖

构件名称	职责描述	输入（数据类型）	输出（数据类型）	依赖构件	错误处理机制（概要）
双目结构光图像输入构件（前端）	选择 4 张图（left/right × env/mix），发起检测请求	`Filex4`	`FormData`	UI组件、HTTP客户端	前端捕获网络异常，展示失败提示
图像接收与转发构件（后端）	接收 4 文件、写临时文件、转发算法服务	`MultipartFilex4`	`DetectionResult`（JSON）	算法服务接口（HTTP）	异常统一封装为 error DetectionResult
平整度服务构件（算法服务）	校验后缀、会话目录隔离、运行算法管道、封装指标/点云/可视化	`UploadFilex4`	`dict`（DetectionResult）	平整度算法主流程	不支持格式→HTTP 400；算法异常→error结构
差分图生成构件（算法模块）	env/mix 配准、亮度匹配、差分、掩膜生成与对比度增强	图像路径/数组	`detect.png + mask.png`（左右）	配准/差分模块	对齐失败等输出警告或异常，上层处理
亚像素角点提取构件（算法模块）	角点检测、裁剪、亚像素精化、左右匹配	`detect.png + mask.png`	`corners_left/right`（坐标）	角点检测/匹配模块	角点检测失败→抛异常；上层返回 error结构
双目立体匹配构件（算法模块）	基于匹配点计算视差/深度	`uv_left/right`	`Z/xyz`	相机模型	无有效点→异常/空结果，上层处理
点云生成与滤波构件（算法模块）	反投影、MAD 去异常、可选稠密化	`xyz`	点云/距离/投影点	点云工具模块	稠密化缺参数→异常；上层捕获
平面拟合构件（算法模块）	最小二乘拟合平面与法向量	点云	plane/normal	平面拟合模块	点数不足→退化处理或异常
偏差分析与指标计算构件（算法模块）	计算 RMS、p95、range 等	距离数组	指标字典	numpy	空值/NaN 使用稳健统计；异常上抛
不平整度结果存储构件（算法服务）	以会话临时目录保存中间产物并在响应中携带关键结果	指标/点云/图像	`image(data URI)`、`pointcloud(json)`	文件系统（临时）	会话结束自动清理；读取失败不阻断主流程

表 4.2 不平整度检测流水线构件表

4.2. 系统构件接口设计

本节仅定义“构件接口”的方法签名、参数类型、返回类型、同步/异步与异常，便于实现与测试。类型以伪代码/接口契约形式表达。

4.2.1. 前端展示构件接口（React）

构件：裂纹检测页面构件

1. 方法：detectCrack(files: File[]) -> Promise<DetectionResult>
2. 调用方式：异步（HTTP）
3. 异常：
   (1) NetworkError：网络/跨域/超时
   (2) ValidationError：用户未选择文件/文件数不符合

构件：平整度检测页面构件

1. 方法：detectFlatness(files: {left_env: File; left_mix: File; right_env: File; right_mix: File}) -> Promise<DetectionResult>
2. 调用方式：异步（HTTP）
3. 异常：
   (1) NetworkError
   (2) ValidationError：缺少任意一张图片

4.2.2. 后端接入构件接口（Spring Boot）

构件：裂纹检测 Controller

1. 路由：POST /api/detect/glass-crack
2. 接口：detectGlassCrack(images: MultipartFile[]) ->

CompletableFuture<DetectionResult>

3. 调用方式：异步（服务端 `@Async` + HTTP 转发）
4. 异常：

(1) UploadError：写临时文件失败

(2) AlgoCallError：算法服务调用失败/超时

构件：平整度检测 Controller

1. 路由：POST /api/detect/glass-flatness
2. 接口：detectGlassFlatness(left_env: MultipartFile, left_mix: MultipartFile, right_env: MultipartFile, right_mix: MultipartFile) -> CompletableFuture<DetectionResult>
3. 调用方式：异步
4. 异常：同上

构件：算法转发客户端（HTTP）

1. 方法：postImage(files: Path[], url: string) -> DetectionResult
2. 方法：postImageWithFieldNames(files: Path[], fieldNames: string[], url: string) -> DetectionResult
3. 调用方式：同步（后端线程内），外层 Controller 异步
4. 异常：捕获所有异常并返回 status="error"` 的 DetectionResult

4.2.3. 算法服务构件接口（FastAPI）

构件：裂纹检测 API

1. 路由：POST /api/detect/glass-crack
2. 接口：detect_glass_crack(image: UploadFile) -> dict
3. 调用方式：同步/异步混合（读取文件为 async；算法计算为同步 CPU）

4. 异常：

5. ImageReadError：文件流读取失败

6. AlgorithmError：算法执行失败（封装为 error 结果）

构件：平整度检测 API

1. 路由：POST /api/detect/glass-flatness

2. 接口：detect_glass_flatness(left_env: UploadFile, left_mix: UploadFile, right_env: UploadFile, right_mix: UploadFile) -> dict

3. 调用方式：异步（保存文件时 await），核心算法同步 CPU

4. 异常：

   (1) HTTPException(400)：文件格式不支持

   (2) AlgorithmError：流程失败（封装为 error 结果）

4.2.4. 算法处理模块构件接口

构件：裂纹检测算法主流程

1. 方法：run(image_input: str) -> dict

2. 输入：image_input：... 或本地路径

3. 输出： dict （DetectionResult 结构： status/title/description/details）

4. 异常：内部捕获并返回 status="error"` 的结果

构件：平整度检测流程入口

1. 方法：main(data_dir: str, result_dir: str) -> None

2. 输入：数据目录（含 4 张图），结果目录（写入产物）

3. 输出：通过文件写出 flatness_metrics.json / pointcloud_data.json / flatness.png

4. 异常：抛出异常由上层服务捕获并封装为错误结果

4.3. 内部构件设计

4.3.1. 裂纹检测流水线内部构件设计：

（1）设计目标

裂纹检测构件负责对单张玻璃图像进行缺陷分析并输出统一格式的检测结果。其内部设计目标是将图像处理、特征分析和决策逻辑解耦，避免单体算法模块过于复杂。

（2）内部子构件划分

裂纹检测构件在内部被划分为以下逻辑子模块：

输入解析子模块

负责将上传的图像数据（bytes / base64 / 本地路径）转换为统一的图像数据结构。

图像预处理子模块

对输入图像进行尺寸归一化、去噪、灰度转换与增强处理，为后续分析提供稳定输入。

特征提取子模块

从预处理后的图像中提取裂纹相关特征，包括边缘特征、纹理统计特征及裂纹区域占比等。

分类与决策子模块

基于提取的特征，按照预设规则或模型输出裂纹状态与定性描述。

结果封装子模块

将检测状态、描述信息和辅助指标封装为统一的 DetectionResult 数据结构。

（3）内部数据流说明

图像数据在构件内部按照“输入解析 → 预处理 → 特征提取 → 决策 → 结果封装”的顺序依次流转，各子模块之间仅通过明确的数据结构进行交互。

（4）异常处理策略

各子模块内部捕获异常并向上传递统一的错误状态，由结果封装模块将异常转换为标准错误输出，避免异常直接传播至构件外部。

4.3.2. 不平整度流水线内部构件设计：

（1）设计目标

不平整度分析构件负责对双目结构光数据进行三维重建与平整度评估。其内部结构设计目标是将复杂的三维处理流程组织为清晰的数据处理流水线。

（2）内部子构件划分

该构件在内部逻辑上由以下子模块组成：

1. **输入校验与会话管理子模块**

   对输入图像完整性与格式进行校验，并创建独立的处理会话环境。

2. **差分图生成子模块**

   对环境光图像与混合光图像进行配准与差分计算，生成后续分析所需的差分图像。

3. **角点提取与匹配子模块**

   从差分图中提取亚像素级角点，并完成左右视角的匹配。

4. **立体重建子模块**

   基于匹配结果计算视差与深度信息，生成三维点云数据。

5. **点云处理子模块**

   对生成的点云进行滤波、异常点剔除和可选稠密化处理。

6. **平面拟合与指标计算子模块**

对点云进行平面拟合，并计算平整度相关的定量指标。

7. **结果组织与输出子模块**

将计算结果、可视化产物与指标信息整理并输出。

（3）内部数据流说明

数据在构件内部以流水线形式依次流转，各处理阶段相互独立，前一阶段的输出作为后一阶段的输入。

（4）异常处理策略

当任一子模块发生异常时，构件通过异常标识或空结果向上传递，由上层算法服务构件统一处理并返回错误信息。

4.4. 构件交互设计

4.4.1. 裂纹检测构件交互顺序（序列图）

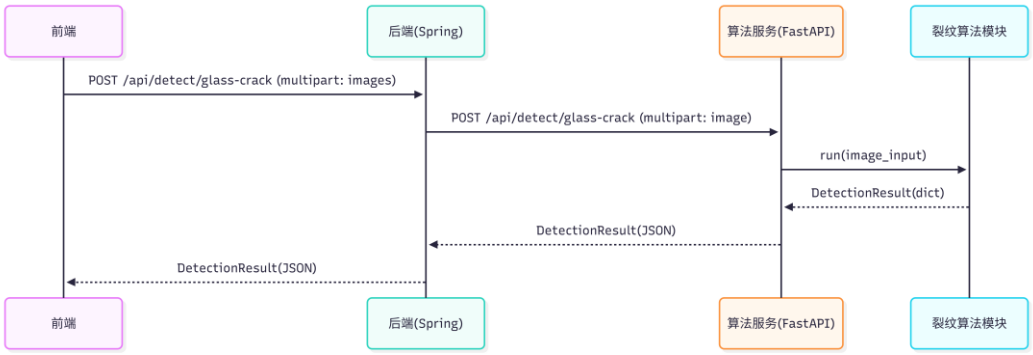


图 4.1 裂纹检测构件交互顺序序列图

4.4.2. 不平整度检测构件交互顺序（序列图）

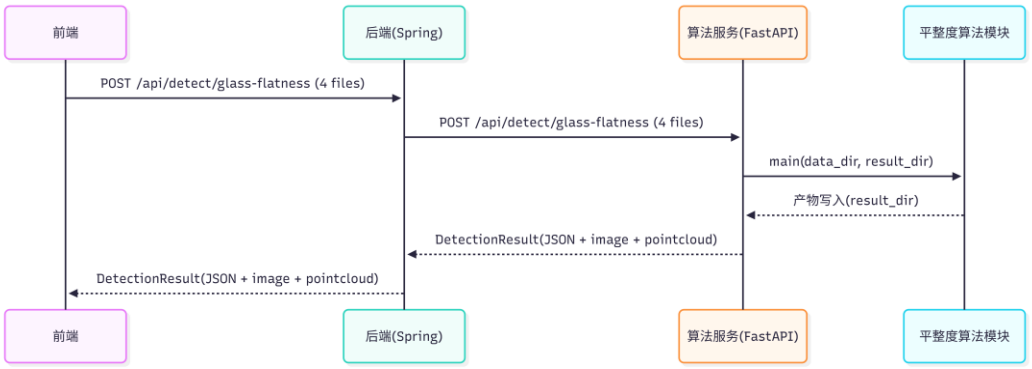


图 4.2 不平整度检测构件交互顺序序列图

4.5. 构件部署与资源需求

4.5.1. 部署方式

构件	部署形态	说明
前端展示构件	静态站点/SPA（Vite构建产物）	浏览器运行，通过环境变量配置后端地址
后端接入构件	Java 服务进程（Spring Boot）	对外 API；异步处理；转发算法服务
算法服务构件	Python 服务进程（FastAPI + Uvicorn）	对外提供算法 API；编排算法模块
算法模块构件	Python 类库	被算法服务调用（同进程）

表 4.4 构件部署方式表

4.5.2. 资源需求（建议值，按基线实现）

构件	CPU	内存	GPU	备注
前端	-	-	-	浏览器侧消耗与图像大小相关
后端（Spring）	1–2 核	512MB–1GB	不需要	主要开销为文件转发与序列化
算法服务（FastAPI）	2–4 核	1–4GB	不需要（基线）	平整度流程包含点云/插值，CPU与内存更敏感
深度学习推理（预留）	4+ 核	4GB+	建议	若启用模型推理需评估 GPU/显存

表 4.5 构件资源需求表

4.5.3. 数据存储需求

数据类型	存储需求	说明
上传图片	临时存储	后端/算法侧均会使用临时目录（或可配置持久化）
中间产物（平整度）	临时存储（会话隔离）	指标 JSON、点云 JSON、可视化 PNG 等
检测结果（业务）	可选持久化	当前基线返回给前端，不强制落库；若需报告/追溯建议增加存储构件

表 6.6 数据存储需求表

5. 接口设计

5.1. 接口设计目标

接口设计的目标是明确系统内部各构件之间以及系统与外部实体之间的数据交互规则和契约，保证系统模块的低耦合性、高内聚性和可替换性。本章接口设计不关注具体实现技术，而是从构件级的黑盒视角描述系统输入、输出及数据交互方式。

具体目标包括：

1. **构件解耦**

   确保各处理构件之间仅通过明确的接口进行数据交互，减少内部依赖。

2. **数据契约清晰**

   对每个接口的数据类型、结构和约束进行定义，保证数据在系统中流转的正确性和一致性。

3. **支持扩展与替换**

   接口设计应允许算法或模块的替换而不破坏整体架构。

4. **可追溯性**

   接口设计应与需求模型保持一致，为需求验证和系统维护提供依据。

5.2. 数据流接口设计

5.2.1. 裂纹检测流水线接口

接口名称	输入数据	输出数据	说明
`CrackImageIn`	裂纹检测原始图像（单张/多张），图像文件/字节流	原始图像数据包	裂纹检测流水线的统一输入接口
`CrackImageValidated`	原始图像数据包	合法图像数据包（含格式/尺寸/完整性校验结果）	接收并完成图像格式、大小、完整性等校验
`GlassROIOut`	合法图像数据包	玻璃区域图像/区域掩膜（ROI/Mask）	提取玻璃区域，减少背景干扰（可选步骤）
`CrackDetectOut`	玻璃区域图像/预处理图像	裂纹检测中间结果（裂纹掩膜/边缘图/候选区域）	执行裂纹检测（传统算法或深度学习），输出裂纹候选结果
`CrackResultFormatted`	裂纹检测中间结果 + 原图/ROI	裂纹检测结果数据（状态、指标、可视化信息）	对裂纹结果进行标注、指标统计与结构化封装，形成统一输出
`CrackResultStored`	裂纹检测结果数据 + 相关产物（可视化图、日志、原图索引）	存储引用（ID/路径/URL）	将结果与产物落库/落盘，便于检索、追溯与报告生成

表 5.1 裂纹检测流水线接口定义表

5.2.2. 不平整度检测流水线接口

接口名称	输入数据	输出数据	说明
`StereoStructuredLightIn`	双目结构光图像（左/右 × 环境光/混合光，共4张）	原始双目数据包	不平整度检测流水线统一输入接口
`DiffImageOut`	原始双目数据包	差分增强图（左右各一）+ 掩膜（左右各一）	生成差分图与有效区域掩膜，为角点提取提供更清晰的结构光图案
`SubpixelCornersOut`	差分增强图 + 掩膜	左右亚像素角点集合（坐标序列）	提取棋盘格/结构光角点并进行亚像素精化
`StereoMatchesOut`	左右角点集合	匹配点对（对应关系）	建立左/右视图特征点匹配关系，为三维重建提供输入
`PointCloudOut`	匹配点对 + 相机参数（内参/基线等）	三维点云（稀疏/稠密）+ 过滤后点云	三维重建并进行异常点过滤/稠密化（可选）
`PlaneFitOut`	过滤后点云	拟合平面参数（平面方程/法向量）	对点云进行平面拟合，建立参考平面
`FlatnessMetricsOut`	点云 + 平面参数	偏差场/统计指标（RMS、范围、分位数等）	计算不平整度偏差并输出量化指标
`FlatnessResultStored`	指标 + 可视化产物（偏差图、点云数据等）	存储引用（ID/路径/URL）	保存不平整度检测结果与产物，支持追溯与复用

表 5.2 平整度检测流水线接口定义表

5.3. 外部接口规范设计

接口名称	路由	输入数据	输出数据	说明
`CrackDetectionAPI`	`POST /api/detect/glass-crack`	裂纹图像（文件上传）	裂纹检测结果数据	面向用户的裂纹检测对外服务接口
`FlatnessDetectionAPI`	`POST /api/detect/glass-flatness`	双目结构光图像（4张文件上传）	不平整度检测结果数据（含指标/可视化/点云可选）	面向用户的不平整度检测对外服务接口
`UnifiedDetectionResultContract`	`-`	任一检测结果（裂纹/不平整度）	统一结果对象（状态、描述、指标、可视化可选）	统一数据契约，保证前端展示/报告生成可复用同一结果结构
`ResultStorageAPI`	`-`	统一结果对象 + 产物	存储引用（ID/路径/URL）	统一的结果存储/查询接口，为报告与归档提供数据来源
`ReportGenerationAPI`	`-`	检测结果引用集合 + 报告配置	报告文件 + 归档引用	面向用户的一键生成/下载报告接口（如需）

表 5.3 外部接口规范表

5.4. 接口约定与数据契约

5.4.1. 数据一致性约定

1. 构件间传递的数据应符合接口定义的数据语义
2. 系统内统一数据坐标系、计量单位和标识规则
3. 中间数据在必要节点进行完整性校验

5.4.2. 异常处理约定

1. 构件在接收到异常或非法数据时，应返回异常状态信息
2. 构件不应直接影响其他构件的运行状态
3. 异常信息应支持后续问题定位与分析

5.5. 接口稳定性与扩展性设计

为保证系统的长期可维护性和可扩展性，接口设计遵循以下原则：

1. 接口稳定性

   已定义接口在系统生命周期内保持稳定，避免频繁变更。

2. 扩展优先于修改

   新功能通过新增接口或扩展数据字段实现，而非修改已有接口。

3. 实现可替换性

   构件内部算法或实现方式可以替换，但对外接口保持不变。

通过上述设计，系统能够在不破坏现有结构的前提下支持功能扩展和性能优化。

6. 数据设计

6.1. 数据设计概述

顶层描述，数据设计概述

数据设计旨在明确系统中数据的组织方式、数据流转路径以及数据存储策略，为体系结构和构件设计提供稳定的数据基础。

数据设计是本系统软件设计的核心组成部分。系统以幕墙玻璃检测数据为中心，围绕图像数据、结构光数据、点云数据以及检测结果数据构建统一的数据模型。该数据模型源自需求分析阶段的数据流分析，并逐步细化为可支持系统处理与存储的逻辑结构。

6.2. 核心数据类型定义

系统涉及的数据主要分为以下几类：

6.2.1. 原始数据

裂纹检测原始图像

双目结构光图像（有光 / 无光）

6.2.2. 中间处理数据

玻璃区域提取结果

差分图

亚像素角点数据

三维点云数据

6.2.3. 分析结果数据

裂纹检测结果（缺陷坐标 / 掩码 / 无缺陷标记）

平整度定性等级

平整度量化指标（最大偏差、平均偏差、RMS、热图）

6.2.4. 输出与归档数据

最终检测报告

原始数据与结果的关联索引信息

6.3. 数据流设计

数据在系统中的流动遵循以下原则：

1. 数据只通过管道在构件间传递
2. 每个构件只对输入数据负责，不直接访问其他构件内部状态

3. 中间结果在必要节点写入临时存储或数据库，以支持异常恢复与回溯

数据在系统中的具体流转结构和处理阶段划分将在体系结构设计章节中通过数据流体系结构进行详细描述。

6.4. 数据存储策略

系统采用集中式存储策略，将数据按处理阶段分类存储：

1. 原始数据存储区
2. 中间结果存储区
3. 最终结果与报告存储区

该设计支持工程验收、质量追溯与历史对比分析等需求。

7. 部署设计

7.1. 部署架构概述

系统采用 基于云服务器的集中式部署方案。整体架构按照功能职责划分为三层，分别为前端层、后端服务层和算法处理层。各层通过标准 HTTP 接口进行通信，逻辑上相互独立，物理上部署于同一云服务器环境中。

该部署结构实现了系统功能模块的分层组织，降低了模块之间的耦合度，为系统的维护、升级与扩展提供了良好的基础。

7.2. 部署架构组成

7.2.1. 前端层部署

前端层负责系统的人机交互与请求发起，采用 React 框架进行开发。前端应用在构建后生成静态资源文件，通过 Nginx 进行部署与管理。

前端服务以云服务器公网 IP 作为访问入口，对外提供页面访问能力。Nginx 负责静态资源的托管与 HTTP 请求处理，保障前端页面的稳定访问。

7.2.2. 后端服务层部署

后端服务层基于 Spring Boot 框架实现，主要承担业务逻辑处理、请求调度以及与算法层的通信功能。

后端服务直接部署于云服务器操作系统环境中，通过指定端口对外提供接口服务。系统运行过程中，后端服务通过 tmux 工具在服务器后台持续运行，以保证在终端会话断开后服务的稳定性与连续性。

7.2.3. 算法处理层部署

算法处理层基于 Flask 框架实现，主要用于执行系统中的核心算法计算任务。

算法服务以独立进程形式部署于云服务器上，并通过 HTTP 接口与后端服务层进行通信。

算法服务同样采用 tmux 方式在后台运行，确保算法计算服务的长期可用性与稳定性。

7.3. 层间通信与运行流程

系统运行过程中，各层之间的交互流程如下：

时序图或活动图，异常处理

1. 前端层向后端服务层发起 HTTP 请求；
2. 后端服务层接收请求后，根据业务逻辑向算法处理层发送计算指令；
3. 算法处理层执行相应算法并返回计算结果；
4. 后端服务层对算法返回结果进行处理与封装；
5. 处理结果以响应形式返回至前端层进行展示。

该通信流程保证了系统数据处理的有序性与逻辑一致性。

7.4. 部署层级交流示意图

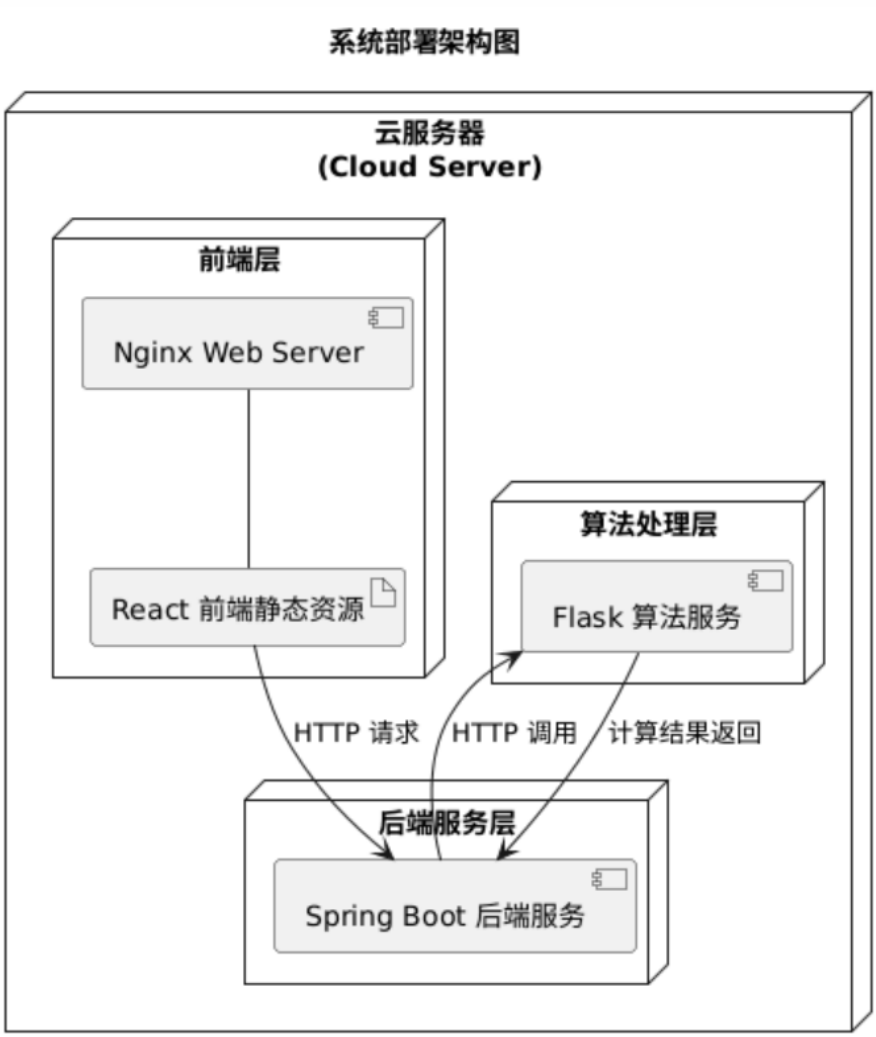


图 7.1 系统部署架构图

8. 设计约束与非功能设计

8.1. 设计约束

本系统的软件设计在实现过程中需满足以下约束条件，这些约束来源于项目背景、运行环境以及课程/工程要求，是设计阶段必须遵循的前提条件。

8.1.1. 技术与平台约束

1. 运行环境约束

系统部署运行于云服务器环境，操作系统为 Linux 系统，要求所采用的软件框架与依赖能够稳定运行于该环境中。

2. 技术栈约束

- 前端采用 React 框架实现，用于实现系统的人机交互界面；
- 后端采用 Spring Boot 框架实现，对外提供统一的业务接口；
- 算法服务采用 Python 生态（FastAPI / Flask）实现，用于执行核心检测算法；

各技术栈之间通过 HTTP 接口进行通信，不允许直接跨层调用内部实现。

3. 通信方式约束

系统各层之间统一采用 HTTP/REST 风格接口进行通信，数据格式以 JSON 为主，避免使用与具体实现强绑定的通信机制。

8.1.2. 数据与设备约束

1. **数据来源约束**

   系统输入数据来源于图像采集设备与结构光系统，数据格式、分辨率及拍摄方式需符合系统设计假设，否则可能影响检测结果准确性。

2. **数据规模约束**

   在当前系统基线实现中，设计目标为单次检测任务的数据规模在合理范围内（单批图像数量有限），暂不考虑大规模并发或海量历史数据的高频访问场景。

3. **算法运行约束**

   算法处理过程主要为 CPU 密集型计算，系统暂未引入 GPU 加速或分布式计算机制，因此对算法执行时间和资源占用进行合理控制。

8.1.3. 项目与实现约束

1. **实现复杂度约束**

   系统以教学与工程实践为导向，在保证设计完整性和合理性的前提下，不引入过度复杂的中间件或分布式架构。

2. **阶段性实现约束**

   本设计说明书中部分构件为“预留构件”或“规划构件”，当前版本系统以核心功能实现为主，未实现的构件不影响整体设计结构的完整性。

8.2. 非功能需求设计

在满足功能需求的基础上，系统设计同时关注非功能需求，以保证系统在实际运行中的可用性、可靠性和可维护性。

8.2.1. 性能需求

1. 系统应支持单次裂纹检测或平整度检测任务在合理时间内完成，满足工程检测的基本时效要求；
2. 系统在设计上支持检测流程的并行执行（裂纹检测与不平整度检测流水线逻辑独立）；
3. 算法计算过程与业务处理过程解耦，避免阻塞前端交互。

8.2.2. 可靠性与稳定性需求

1. 系统在接收到异常数据或算法执行失败时，应能够返回明确的错误信息，而不导致系统整体崩溃；
2. 各构件之间相互独立，单个构件异常不应影响其他构件的正常运行；
3. 中间结果在关键节点进行持久化存储，支持异常恢复和问题回溯。

8.2.3. 可维护性需求

1. 系统采用构件化设计，各构件职责清晰，便于定位问题与独立维护；
2. 接口定义稳定，内部实现可替换，降低系统后期维护成本；
3. 数据模型与接口契约清晰，为后续功能扩展和算法升级提供基础。

8.2.4. 可扩展性需求

1. 系统设计支持新增检测算法或改进现有算法，而不影响整体架构；
2. 系统支持通过增加新的流水线或构件来扩展检测功能；
3. 接口设计预留扩展字段，避免因需求变化频繁修改已有接口。

8.2.5. 易用性需求

1. 前端界面应提供清晰的操作流程，用户无需了解算法细节即可完成检测；
2. 检测结果以结构化信息和可视化方式呈现，便于用户理解和使用；
3. 系统操作过程应具备明确的状态反馈和错误提示。

9. 设计总结

本文档对玻璃幕墙缺陷智能检测系统的软件设计进行了系统化、结构化的描述。在需求分析完成的基础上，围绕系统的业务流程和数据处理特点，构建了以数据流体系结构和流水线构件模型为核心的软件设计方案。

在设计过程中，系统以数据为中心，将裂纹检测、不平整度检测以及检测结果汇总与报告生成抽象为多条独立但可协同的数据处理流水线。通过管道-过滤器架构模式，实现了处理流程清晰、构件职责明确、模块之间低耦合的设计目标。

本文档详细说明了系统的总体设计思想、数据设计、体系结构设计、接口设计、构件级设计以及部署设计，为系统的实现、测试和维护提供了完整、可追溯的设计依据。同时，通过对设计约束和非功能需求的分析，明确了系统在性能、可靠性、可维护性和可扩展性等方面的设计考虑。

总体而言，本软件设计在保证工程可实现性的前提下，兼顾了系统的规范性、可扩展性与教学实践价值。该设计不仅能够指导当前版本系统的实现，也为后续功能扩展、算法升级和系统优化提供了稳定的结构基础。

10. 附录

10.1. 参考文献

1. 中华人民共和国住房和城乡建设部. *玻璃幕墙工程质量检验标准* [S]. 北京: 中国建筑工业出版社, 2020. JGJ/T 139-2020. 发布日期: 2020-04-16, 实施日期: 2020-10-01.

2. 中铁建设集团有限公司, 中国铁路成都局集团有限公司客站建设指挥部, 重庆大学. *基于计算机视觉的玻璃幕墙施工误差检测方法* [P]. CN 119991654 A, 2025-04-11.

3. 中华人民共和国国家质量监督检验检疫总局, 中国国家标准化管理委员会. *建筑幕墙* [S]. GB/T 21086-2007. 发布日期: 2007-09-11, 实施日期: 2008-02-01.

4. OpenCV Development Team. *OpenCV: Open Source Computer Vision Library* [M]. 2024.

5. PyTorch Development Team. *PyTorch: An Imperative Style, High-Performance Deep Learning Library* [M]. 2024.

6. React Documentation. *React – A JavaScript Library for Building User Interfaces* [EB/OL]. 2024. https://react.dev

7. FastAPI Documentation. *FastAPI – High Performance Python Web Framework* [EB/OL]. 2024. https://fastapi.tiangolo.com

8. Spring Boot Documentation. *Spring Boot Reference Guide* [EB/OL]. 2024. https://spring.io/projects/spring-boot

10.2. 三级目录：表目录和图目录