zhejiang-yige

智能软件综合研究

实训报告



学期 2024-2025-1

专业 计算机科学与技术

成员 张雅瑞 2022334323029

雷心成 2022331201045

褚昊 2022333500054

韩圣杰 2022330300160

教师 马明泽、吴婷婷

1. **引言**

1.1项目背景

在现代数字化时代，文字识别（OCR）技术变得越来越重要。随着办公自动化和数字化转型的需求增长，快速、准确地从图像、截图、文档中提取文字成为许多用户和工作场景的迫切需求。传统的OCR工具往往操作复杂、效率低下，或者需要付费使用。

1.2项目目的

Umi-OCR的主要目的是开发一款开源、免费、高效的文字识别工具，主要特点包括：

1.提供轻量级、快速的OCR文字识别解决方案

2.降低OCR使用的技术门槛

3.支持多种图像识别场景

4.提供跨平台的桌面应用程序

5.优化用户体验，使OCR操作变得简单直观

1.3项目范围

Umi-OCR的项目范围覆盖了多语言文字识别、截图和图片文件识别等核心功能，通过Python开发并基于PaddleOCR引擎，为Windows平台用户提供了一款轻量级、高效的跨平台桌面OCR工具，可广泛应用于办公、学习、文字提取等多种场景，致力于降低文字识别的技术门槛，提供简单直观的用户体验。

1. **项目成员职务及任务分解**

2.1项目成员职务及任务分解

项目团队由四名成员组成，分别担任不同的关键角色，确保项目的顺利进行。

张雅瑞：作为项目经理、产品经理，负责整体项目规划与管理，并完成相关代码的测试和汇报答辩。

韩圣杰：作为系统架构师和测试研发工程师，负责产品需求分析和文档的撰写，定义了系统架构，发现了测试中的问题。

雷心成：作为算法工程师和研发工程师，负责后端的算法和相关内容的实现，并完成相关文档的撰写和代码的测试。

褚浩：作为研发工程师和UI设计师，负责前端的UI设计和相关内容的实现，完成相关文档的撰写和代码测试。

1. **可行性及需求分析**

3.1可行性分析

3.1.1技术科学性

从技术角度来看，Umi-OCR是基于计算机视觉和深度学习技术实现的开源文字识别平台。项目选择PaddleOCR作为底层识别引擎，这是一个成熟且高效的开源OCR框架，具有较高的识别准确率和处理速度。通过Python语言开发，并采用现代化的开发框架，能够确保系统的技术可靠性和创新性。项目支持多语言识别、多种图像处理模式，技术实现路径清晰可行。

3.1.2经济可行性

从经济角度分析，Umi-OCR作为开源免费项目，开发成本相对较低。主要投入包括开发人员的人力成本、服务器维护费用等。项目通过开源社区协作模式，可以显著降低持续开发和维护的expenses。未来可以通过提供高级功能的增值服务、企业定制等方式实现潜在的商业价值，如提供专业版本、技术支持或定制开发等。

3.1.3市场可行性

在文字识别需求日益增长的数字化时代，Umi-OCR具有广阔的市场前景。随着办公自动化、数字化转型的推进，对快速、准确的文字识别工具的需求持续增加。项目瞄准了个人用户和中小企业的实际需求，提供轻量级、易用的OCR解决方案。开源免费的特性，以及支持多语言识别的功能，能够吸引大量潜在用户。

3.1.4可行性风险和挑战

Umi-OCR项目面临的主要风险和挑战包括技术层面的OCR识别准确率和性能优化、多语言识别的算法复杂性以及不同图像质量下的识别稳定性，以及项目发展层面的开源社区持续维护、技术迭代更新的及时性和用户体验的持续优化。这些挑战需要项目团队保持技术敏锐度，不断改进算法模型，优化用户交互，并积极响应社区反馈，以确保项目的长期可持续发展和竞争力。

3.2需求分析

3.2.1 功能需求

1.图像识别：提供多种图像文字识别功能，支持截图、本地图片文件的文字识别。能够快速准确地提取图像中的文字内容，并支持文字复制、导出等操作。

2.识别模式：支持多种识别模式，包括全文识别、区域识别、智能识别等，满足不同场景下的文字提取需求。

3.语言支持：提供多语言文字识别功能，支持中文、英文、日文等多种语言的文字识别，适应不同用户的使用需求。

4.快捷操作：支持快捷键截图识别，提供便捷的操作方式，提高用户使用效率。

5.结果管理：提供识别结果的管理功能，支持识别历史记录查看、导出等操作。

6.系统设置：提供用户个性化设置，包括识别语言、快捷键、界面主题等自定义选项。

7.更新与支持：提供软件版本更新检测和用户反馈渠道。

3.2.2 数据需求

1.识别数据：收集图像文字识别所需的图像数据，确保能够准确提取各类图像中的文字内容。

2.用户配置：存储用户个人配置信息，包括识别语言偏好、界面设置等。

3.识别历史：记录用户的文字识别历史,支持后续查询和管理。

4.语言库：维护多语言识别所需的语言模型和字符库。

3.2.3 性能需求

1.识别速度：确保文字识别的实时性和高效性，快速响应用户的识别请求。

2.识别准确性：提高OCR识别的准确率，降低错误识别的可能性，确保识别结果的可靠性。

3.系统兼容性：支持多种操作系统和硬件平台，具备良好的跨平台兼容性。

4.并发处理：支持多图像同时识别，满足用户批量处理文字识别需求。

5.资源占用：优化系统资源使用，保持较低的内存和CPU占用率。

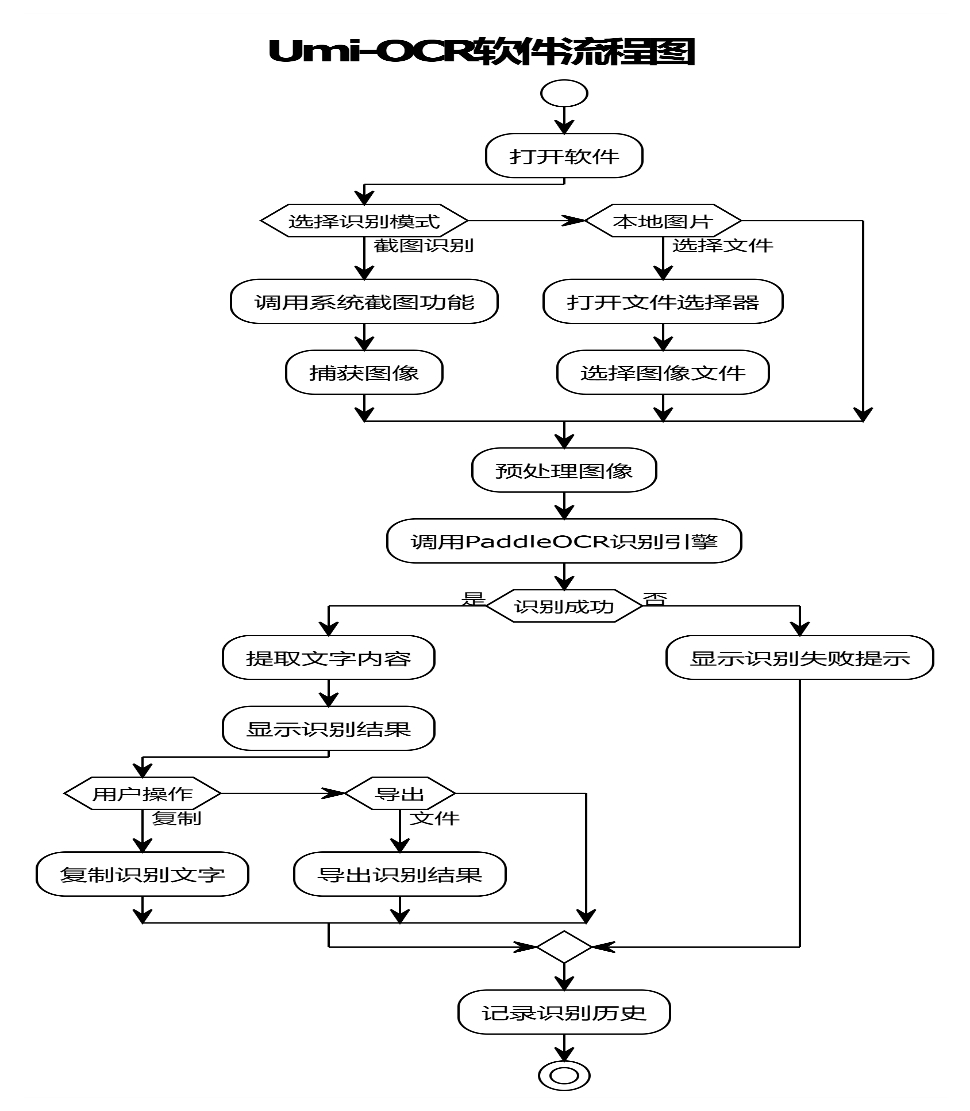
6.安全性：确保用户数据和识别结果的安全性，防止数据泄露和未经授权的访问。

1. **系统设计**

4.1系统设计概述

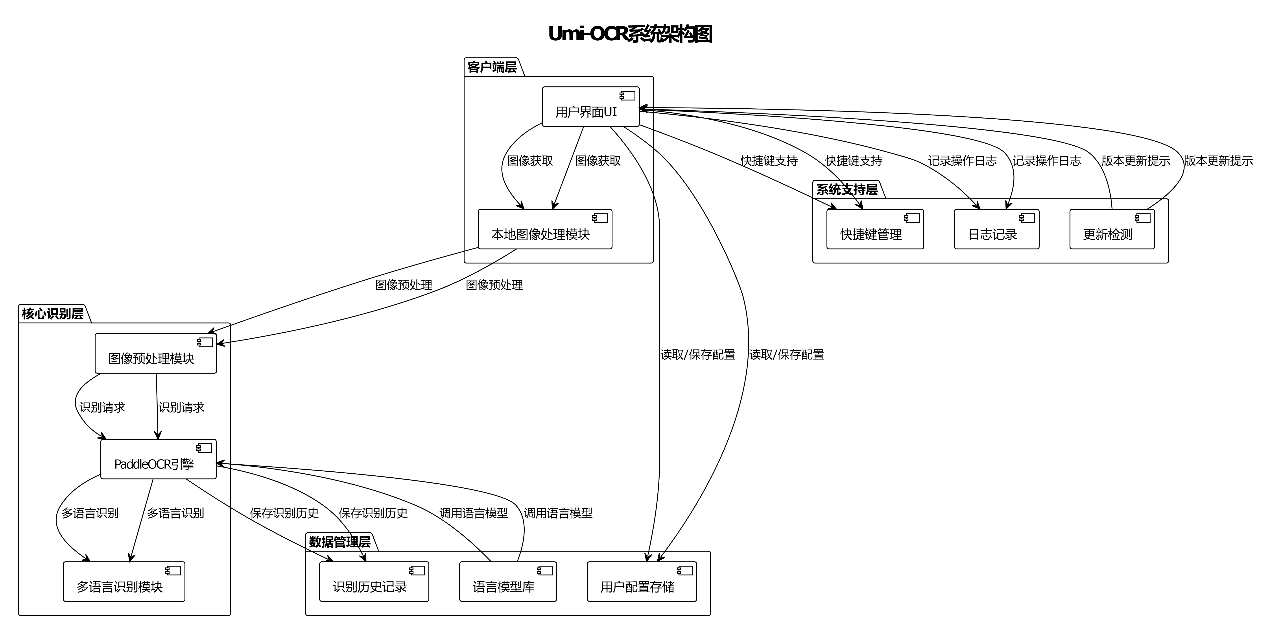
Umi-OCR是一款基于PaddleOCR引擎的开源跨平台文字识别工具，致力于为用户提供快速、准确、便捷的图像文字提取解决方案。系统通过计算机视觉技术，支持多语言识别、截图识别、本地图片文件识别等功能，满足办公、学习、研究等多样化的文字识别需求。软件架构简单直观，操作流程清晰，用户可以轻松实现图像文字的快速提取和管理。

4.1.1软件流程图



**图4-1 OCR软件流程图**

4.1.2系统架构图



**图4-2 OCR系统架构图**

4.2前端设计

前端设计是Umi-OCR系统中至关重要的部分，负责提供直观、高效的用户交互界面。

1.QtQuick框架和QML语言

在前端开发中，选择了QtQuick框架和QML语言作为主要的开发框架和语言。QtQuick以其高效的性能和跨平台的特性而闻名，采用QML语言进行声明式UI设计。这种设计方式使得用户界面的开发更加直观和快速，同时提高了代码的可读性和可维护性。每个功能或界面元素都可以被封装成一个独立的QML组件，这样整个应用的结构更清晰。

2. QtQuick.Controls库的使用

为了更高效地进行界面布局和交互设计，采用了QtQuick.Controls库提供的丰富组件。QtQuick.Controls提供了一套完整的界面元素，包括按钮、文本框、滑动条等，使得开发者可以更专注于业务逻辑的实现而不必过多关注样式和布局。

3. QtQuick.Dialogs和文件对话框

通过QtQuick.Dialogs模块，实现了文件对话框的功能。这种方式通过提供标准的文件选择对话框，使得用户可以方便地选择文件和目录，同时保持了界面的一致性和用户体验。

4. Qt.labs.settings进行本地化保存

使用Qt.labs.settings模块进行本地化保存，通过Settings类管理应用的配置和状态。这种方式使得应用的设置可以在用户会话之间持久化，提供了良好的用户体验。

5. QtQml.Models和数据绑定

通过QtQml.Models模块，实现了数据绑定和模型-视图编程。这种方式使得UI组件可以自动更新以反映数据的变化，减少了手动更新UI的需要，提高了开发效率。

6. QtGraphicalEffects和视觉效果

使用QtGraphicalEffects模块，为应用添加了视觉效果，如阴影和渐变。这些效果增强了应用的视觉吸引力，提升了用户体验。

7. QtQuick.Layouts进行布局管理

通过QtQuick.Layouts模块，实现了灵活的布局管理。这种方式使得开发者可以轻松地创建复杂的布局，同时自动处理子组件的大小和位置，适应不同的屏幕尺寸和方向。

8. RESTful API和网络通信

通过QtNetwork模块，实现了RESTful API的网络通信。这种方式使得应用可以与后端服务器进行高效的数据交换，支持异步请求和错误处理，确保了通信的可靠性和效率。

4.3后端设计

后端设计是Umi-OCR系统的核心技术支撑，负责提供高效的文字识别服务。

1. Bottle框架

选择了 Bottle 作为主要框架，它是一个轻量级且易于扩展的 Python Web 框架。通过 Bottle，搭建了 RESTful API 接口，使得前后端之间的数据交换更加规范化和高效。功能与特点：提供核心的 API 服务，例如 /api/ocr（OCR 服务）、/api/doc/upload（文档上传处理）等。使用 RESTful 设计风格，明确分离请求方法（如 GET、POST），便于扩展和维护。

2. 任务管理与调度

2.1任务管理是后端设计的核心模块，集中处理复杂的 OCR 和文档任务：

2.2任务队列与状态管理：使用 MissionQueue 维护任务的运行状态（如等待、进行中、暂停、完成）。

2.3异步任务调度：提供任务的回调机制，支持高效的异步处理。

2.4灵活任务定义：独立设计 OCR、文档解析和二维码处理任务，方便扩展新任务类型。

2.5任务管理的设计确保了高效的资源调度和系统的稳定运行。

3. 输出系统：多格式支持

后端输出模块设计了丰富的结果导出功能，满足不同场景需求：支持的格式：纯文本（TXT）、表格（CSV）、Markdown（MD）、双层 PDF、JSON 行（JSONL）等。模块化扩展：每种输出格式由独立类实现，统一继承自基类 Output，便于新增格式支持。

4. 文块解析与后处理

在 OCR 结果处理方面，后端设计了强大的文块解析能力：

4.1解析模式：支持多栏和单栏排版解析，适应不同文档结构。

4.2区域过滤：用户可配置忽略区域，以避免干扰 OCR 结果的噪声。

4.3段落合并：基于自然段识别技术，优化结果的可读性。

4.4通过 tbpu 模块，系统实现了对复杂文档排版的准确还原。

5. 图像处理支持

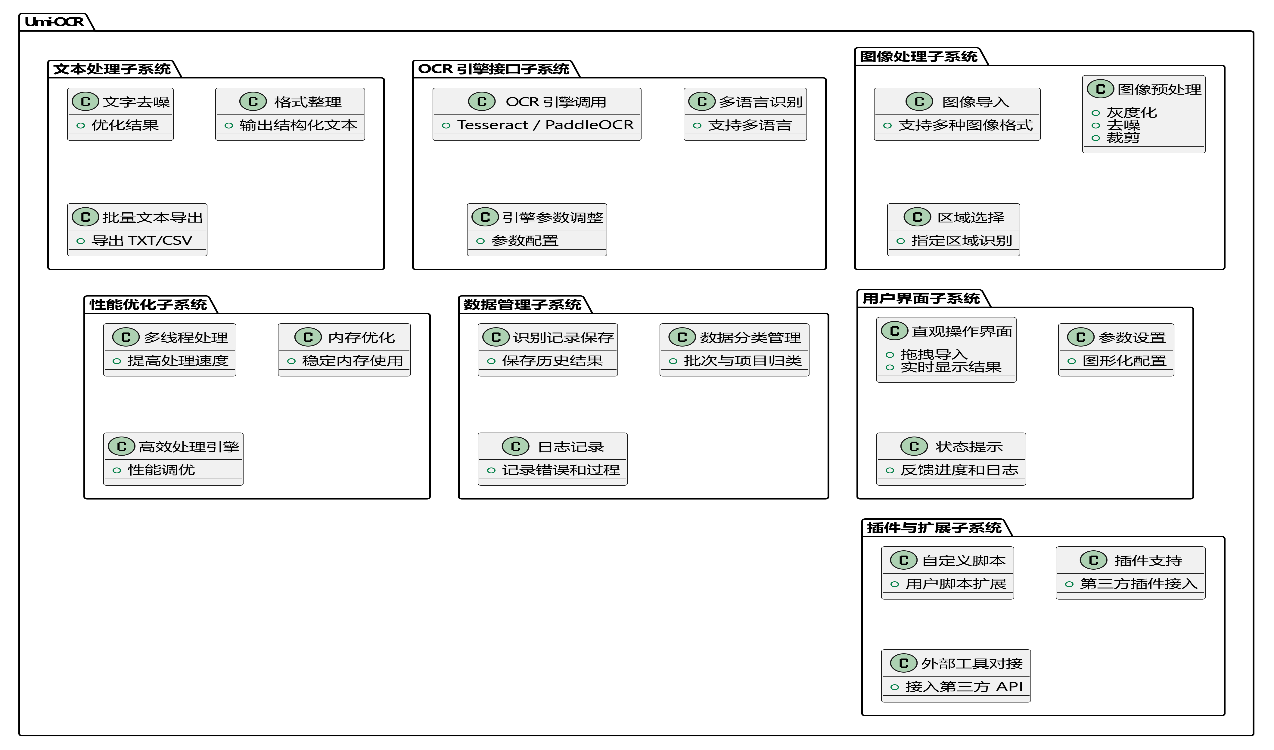
图像处理模块（image\_controller）为 OCR 任务提供了高质量的输入支持：

1. **模块设计**

5.1.1模块汇总

|  |  |
| --- | --- |
| 模块名称 | 功能描述 |
| 图像处理子系统 | |
| 图像导入 | 支持导入多种图像格式，如 PNG、JPG 等，进行批量处理。 |
| 图像预处理 | 对图像进行灰度化、去噪、裁剪等处理，提升识别准确率。 |
| 区域选择 | 允许用户选择图像的特定区域，针对性进行 OCR 识别。 |
| OCR引擎接口子系统 | |
| OCR引擎调用 | 调用 PaddleOCR引擎进行文字识别。 |
| 多语言识别 | 支持多种语言与字体的识别，用户可自定义语言包。 |
| 引擎参数调整 | 允许用户配置 OCR 引擎参数，如精度与速度的权衡。 |
| 文本处理子系统 | |
| 文字去噪 | 对识别结果进行优化，去除多余换行、空格等干扰字符。 |
| 格式整理 | 对输出文本进行分组整理，支持段落和行结构的重建。 |
| 批量文本导出 | 将识别结果批量导出为 TXT、CSV 等格式文件。 |
| 用户界面子系统 | |
| 直观操作界面 | 提供拖拽图像导入、实时显示识别结果的用户界面。 |
| 参数设置 | 提供图形化界面调整识别引擎的参数与预处理选项。 |
| 状态提示 | 实时反馈识别进度、错误状态和日志信息。 |
| 数据管理子系统 | |
| 识别记录保存 | 保存每次 OCR 识别的结果，方便用户查询与管理。 |
| 数据分类管理 | 提供按项目或批次对识别结果进行分类与存档功能。 |
| 日志记录 | 记录识别过程与错误日志，便于排查问题。 |
| 性能优化子系统 | |
| 多线程处理 | 支持多线程图像处理与识别，提高批量识别的速度。 |
| 内存优化 | 优化内存使用，确保系统在处理大量图像时的稳定性。 |
| 高效处理引擎 | 对 OCR 处理流程进行性能调优，减少处理时间。 |
| 插件与扩展子系统 | |
| 自定义脚本 | 允许用户添加自定义脚本，扩展功能与自动化处理。 |
| 插件支持 | 提供接口支持用户开发并接入第三方插件。 |
| 外部工具对接 | 可对接其他文本处理工具或第三方 API 进行功能扩展。 |

5.1.2模块框架图



**图5-1 模块框架图**

5.2重要模块设计实现

5.2.1截图OCR模块

|  |  |
| --- | --- |
| 模块名称 | 截图OCR |
| 功能描述 | 用户可以通过截图，将图片文件拖拽到应用界面中或从其他应用复制图片，然后粘贴到应用中进行OCR识别。 能够管理和执行批量OCR任务，显示任务进度和状态。 允许用户暂停、恢复和停止OCR任务。 |
| 主要算法实现 | 截图，拖入，粘贴图片：  用户调用screenshot函数开始截图，该函数利用qmlapp.imageManager.screenshot进行屏幕捕捉。 截图完成后，screenshotEnd函数被调用，并将截图ID传递给后端进行OCR处理。用户调用paste函数开始粘贴操作，该函数利用qmlapp.imageManager.getPaste获取剪贴板内容。 如果剪贴板中包含图片，将图片ID传递给后端进行OCR处理。 如果剪贴板中包含文件路径，调用ocrPaths函数处理这些路径。  // 开始截图  function screenshot() {  qmlapp.imageManager.screenshot(screenshotEnd)  }  // 截图完毕  function screenshotEnd(clipID) {  popMainWindow()  if(!clipID) { // 截图取消或失败  tabPage.callPy("ocrImgID", undefined, undefined)  return  }  const configDict = configsComp.getValueDict()  tabPage.callPy("ocrImgID", clipID, configDict)  qmlapp.tab.showTabPageObj(tabPage) // 切换标签页  imageText.showImgID(clipID) // 展示图片  }  // 鼠标拖入图片  DropArea\_ {  anchors.fill: parent  callback: tabPage.ocrPaths  }  // 开始粘贴  function paste() {  popMainWindow()  const res = qmlapp.imageManager.getPaste()  if(res.error) {  const t = qsTr("获取剪贴板异常")  qmlapp.popup.simple(t, res.error)  tabPage.callPy("ocrImgID", `[Error] ${t} ${res.error}`, undefined)  return  }  if(res.text) {  const t = qsTr("剪贴板中为文本")  qmlapp.popup.simple(t, res.text)  tabPage.callPy("ocrImgID", `[Warning] ${t}`, undefined)  return  }  qmlapp.tab.showTabPageObj(tabPage) // 切换标签页  if(res.imgID) { // 图片  imageText.showImgID(res.imgID)  const configDict = configsComp.getValueDict()  tabPage.callPy("ocrImgID", res.imgID, configDict)  }  else if(res.paths) { // 地址  ocrPaths(res.paths)  }  }  图片预览和编辑 :  用户通过界面上的控件选择图片的区域后，调用 cropImage 函数，该函数利用 qmlapp.imageManager.cropImage 方法进行裁剪。  // 图片预览  function previewImage(imageID) {  imagePreviewComponent.source = imageID;  imagePreviewComponent.visible = true;  }  // 图片编辑（例如裁剪）  function cropImage(x, y, width, height) {  const croppedImageID = qmlapp.imageManager.cropImage(mainImageID, x, y, width, height);  mainImageID = croppedImageID; // 更新主图片ID为裁剪后的图片ID  previewImage(croppedImageID); // 预览裁剪后的图片  } |
| 界面展示 |  |

5.2.2批量OCR模块

|  |  |
| --- | --- |
| 模块名称 | 批量OCR模块 |
| 功能描述 | 允许用户选择单个图片文件或整个文件夹进行批量OCR处理。 显示已选择文件的列表，并提供删除或清空列表的功能。能够管理和执行批量OCR任务，显示任务进度和状态，允许用户暂停，恢复和停止OCR任务 |
| 主要算法实现 | 文件选择与管理:  addFiles 函数接收文件路径列表，过滤出有效的图片文件，并添加到文件列表中。 removeSelectedFiles 函数从文件列表中移除用户选定的文件。  // 添加文件或文件夹  function addFiles(paths) {  const validPaths = filterValidImages(paths); // 过滤有效图片文件  filesTableView.append(validPaths); // 将有效文件添加到表格视图中  }  // 删除选定的文件  function removeSelectedFiles() {  const selectedFiles = filesTableView.getSelectedFiles();  selectedFiles.forEach(file => filesTableView.remove(file)); // 从表格视图中移除  }  OCR任务管理:  startOcrTasks 函数获取所有待处理文件，并携带OCR设置发送到后端开始处理。 stopOcrTasks 函数向服务器发送停止当前任务的请求。  // 开始批量OCR任务  function startOcrTasks() {  const files = filesTableView.getAllFiles();  const ocrSettings = configsComp.getValueDict(); // 获取OCR设置  // 发送任务到后端，并接收任务ID  const taskID = tabPage.callPy("startBatchOcr", files, ocrSettings);  setTaskRunning(true); // 设置任务运行状态  }  // 停止批量OCR任务  function stopOcrTasks() {  tabPage.callPy("stopBatchOcr", currentTaskID);  setTaskRunning(false); // 设置任务停止状态  } |
| 界面展示 |  |

5.2.3PDF识别模块

|  |  |
| --- | --- |
| 模块名称 | PDF识别 |
| 功能描述 | PDF 识别功能通过解析 PDF 文件的文本和图片内容，结合 OCR 技术生成可搜索的双层 PDF、纯文本 PDF ，支持批量处理、忽略区域设置和文本后处理优化。 |
| 主要算法实现 | PDF解析与任务拆分  mission\_doc.py 中的 MissionDOC 类是 PDF 识别的核心，它负责解析PDF文档并将任务拆分到页级别：  # 添加一个文档任务  def addMission(self, msnInfo, msnPath, pageRange=None, pageList=[], password=""):  try:  doc = fitz.open(msnPath) # 打开PDF文档  except Exception as e:  return f"[Error] fitz.open error: {msnPath} {e}"  if doc.is\_encrypted and not doc.authenticate(password): # 解密文档  return "[Error] Doc encrypted. 文档已加密，请提供密码。"  # 处理页码范围或页面列表  page\_count = doc.page\_count  if len(pageList) == 0: # 如果未指定页面列表，默认处理全部页面  pageList = list(range(0, page\_count))  # 检查页面合法性  for p in pageList:  if not 0 <= p < page\_count:  return f"[Error] 页数列表超出 1~{page\_count} 范围"  msnInfo["pageList"] = pageList  msnInfo["doc"] = doc # 文档对象保存到任务信息中  # 创建任务队列  return self.addMissionList(msnInfo, pageList)  页面内容解析  PDF 每页的内容会被解析为 图片和文本块，OCR 针对图片部分处理，而文本部分直接提取。  def msnTask(self, msnInfo, pno):  doc = msnInfo["doc"] # 获取文档对象  page = doc[pno] # 获取当前页  extractionMode = msnInfo["argd"]["doc.extractionMode"] # OCR模式  # 图片提取  imgs = [] # 待OCR图片列表  if extractionMode == "fullPage": # 整页OCR  rect = page.rect  matrix = fitz.Matrix(2, 2) # 提高渲染分辨率  pixmap = page.get\_pixmap(matrix=matrix)  imgs.append({"bytes": pixmap.tobytes("png"), "xy": (0, 0)})  # 文本提取  tbs = []  if extractionMode == "mixed": # 混合模式：提取图片和文本  page\_data = page.get\_text("dict")  for block in page\_data["blocks"]:  if block["type"] == 1: # 图片块  imgs.append({"bytes": block["image"]})  elif block["type"] == 0:  # 文本块  tbs.append({"text":block["lines"][0]["spans"][0]["text"]})  # 调用OCR  if imgs:  ocr\_results = MissionOCR.addMissionWait(msnInfo["argd"], imgs)  for ocr\_result in ocr\_results:  tbs.append(ocr\_result)  return {"code": 100, "data": tbs}  结果后处理与格式化  OCR结果会经过后处理模块 tbpu，对文本段落进行优化或忽略特定区域。  # 忽略区域处理  if msnInfo["ignoreArea"] and tbs:  tbs = msnInfo["ignoreArea"]["obj"].run(tbs)  # 段落解析器  if msnInfo["tbpu"] and tbs:  for tbpu in msnInfo["tbpu"]:  tbs = tbpu.run(tbs)  PDF输出  OCR结果可被叠加到原始PDF中或生成新的文本PDF，通过 output\_pdf\_layered.py 和 output\_pdf\_one\_layer.py 实现。  def print(self, res):  page = self.pdf[res["page"] - 1] # 获取目标页  for tb in res["data"]: # 遍历文本块  text = tb["text"]  box = tb["box"]  fontsize = self.\_calculateFontSize(text, box[2] - box[0], box[3] - box[1])  point = fitz.Point(box[0], box[3])  page.insert\_text(  point,  text,  fontsize=fontsize,  fontname="cjk",  fill\_opacity=self.opacity,  ) |
| 界面展示 |  |

5.2.4公式识别模块

|  |  |
| --- | --- |
| 模块名称 | 公式识别 |
| 功能描述 | 基于 Pix2Text 引擎，支持从图像中精准识别数学公式（输出为 LaTeX 格式）和混合文本，通过模块化设计与标准化接口，实现高效的公式提取与文本整合。 |
| 主要算法实现 | 图像处理：输入的加载与预处理  图像的加载与预处理是公式识别的基础，涉及图像的旋转矫正、裁剪和缓存。  // 添加文件或文件夹  function addFiles(paths) {  const validPaths = filterValidImages(paths); // 过滤有效图片文件  filesTableView.append(validPaths); // 将有效文件添加到表格视图中  }  def addPixmap(self, pixmap):  imgID = str(uuid4())  self.pixmapDict[imgID] = pixmap # 缓存图像  return imgID  def linePreprocessing(textBlocks):  rotation\_rad = \_estimateRotation(textBlocks) # 估计旋转角度  bboxes = \_getBboxes(textBlocks, rotation\_rad) # 获取标准化bbox  for i, tb in enumerate(textBlocks):  tb["normalized\_bbox"] = bboxes[i] # 写入标准化的包围盒  textBlocks.sort(key=lambda tb: tb["normalized\_bbox"][1]) # 按Y轴排序  return textBlocks  模型加载  Pix2Text 模型被分为三个部分：  分类模型：  用于检测图片中的不同区域，区分公式、文本等内容。  使用 mfd-yolov7\_tiny 模型。  公式识别模型：  公式识别通过 ONNX 模型（如 mfr-onnx）实现。  专为数学公式设计，支持 LaTeX 表达式输出。  self.p2t = Pix2Text(  analyzer\_config=dict(model\_name="mfd", model\_type="yolov7\_tiny", ...),  formula\_config=dict(model\_backend="onnx", model\_dir="mfr-onnx", ...),  text\_config=dict(det\_model\_backend="onnx", ...)  ) |
| 界面展示 |  |

5.2.5二维码测试

|  |  |
| --- | --- |
| 模块名称 | 二维码识别模块 |
| 功能描述 | 二维码识别模块通过解析图片中的二维码，支持路径、PIL对象和Base64数据作为输入，利用 zxingcpp 解析二维码内容与位置信息，提供多任务管理和图像预处理功能。 |
| 主要算法实现 | 任务执行流程  接收图片路径、PIL 对象或 Base64 数据作为输入，创建任务队列。  def msnTask(self, msnInfo, msn):a  if not zxingcpp: # 检查 zxingcpp 库是否成功导入  return {  "code": 901,  "data": f"【Error】无法导入二维码解析器 zxingcpp。",  }  try:  # 读取图像  if "pil" in msn:  img = msn["pil"]  elif "path" in msn:  img = Image.open(msn["path"])  elif "base64" in msn:  img = Image.open(BytesIO(base64.b64decode(msn["base64"])))  except Exception as e:  return {"code": 202, "data": f"【Error】图片读取失败。 {e}"}  # 预处理图像  if "argd" in msnInfo:  img = self.\_preprocessing(img, msnInfo["argd"])  # 调用二维码解析  try:  codes = zxingcpp.read\_barcodes(img)  except Exception as e:  return {"code": 204, "data": f"【Error】二维码解析失败。 {e}"}  # 转换解析结果为统一格式  return self.\_zxingcpp2dict(codes)  二维码解析层  mission\_qrcode.py 中通过 zxingcpp 完成二维码的生成与解析。  def \_zxingcpp2dict(self, codes):  if not codes:  return {"code": 101, "data": "QR code not found in the image."}  data = []  for c in codes:  if not c.valid:  continue  d = {  "orientation": c.orientation,  "box": [  [c.position.top\_left.x, c.position.top\_left.y],  [c.position.top\_right.x, c.position.top\_right.y],  [c.position.bottom\_right.x, c.position.bottom\_right.y],  [c.position.bottom\_left.x, c.position.bottom\_left.y],  ],  "text": c.text if c.content\_type.name == "Text" else None,  }  data.append(d)  return {"code": 100, "data": data} |
| 界面展示 |  |

1. **测试用例设计**

1.截图OCR测试

测试用户是否可以通过截图、将图片文件或者从其他应用拖拽到应用中来进行识别。测试用户是否能暂停、恢复和停止OCR任务。

2.批量OCR测试

测试用户是否能选择单个图片文件或者整个文件夹进行批量OCR处理，测试是否提供删除或清空列表功能。

3.PDF识别测试

测试能否通过解析PDF文件的文本和图片内容，测试是否能结合OCR技术实现测试。

4.公式识别测试

测试是否能从图像中精准识别数学公式和混合文本，实现高效的公式提取与文本结合。

5.二维码识别测试

测试能否通过解析图片中的二维码，测试是否支持路径、PIL对象和Base64数据作为输入，是否提供多任务管理和图像预处理。

1. **测试结果**

|  |  |
| --- | --- |
| 模块 | 测试结果演示 |
| 截图模块 |  |
| 批量模块 |  |
| PDF模块 |  |
| 公式模块 |  |
| 二维码模块 |  |

1. **项目日常沟通机制及记录**

8.1日常沟通机制

我们使用QQ群实时地进行信息交流和讨论。这个QQ群成为了日常沟通的主要平台，允许团队成员轻松分享信息和协调工作。这种群聊的沟通方式更适合处理这种协作的事务。

在固定的时间通过腾讯会议进行简短的会议，分享各自的工作进展、遇到的难题以及需要协调的事项。这样的会议会协调进度的一致性。

8.2沟通内容记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 时间 | 主题 | 内容概要 | 参与人员 | 记录 |
| 12.11 | 确认选题 | 确认选题 | 全体成员 |  |
| 12.13 | 分工设计 | 发布确认任务，开始相关学习 | 全体成员 |  |
| 12.18 | 前端设计 | 设计和开始前端任务 | 全体成员 |  |
| 12.18 | 后端设计 | 设计和开始后端 | 全体成员 |  |
| 12.23 | 相关部分整合 | 将代码整合 | 全体成员 |  |
| 12.24 | 测试 | 将测试工作开展 | 全体成员 |  |
| 12.25 | 文档撰写 | 相关人员进行文档的撰写 | 全体成员 |  |

1. **项目管理机制及执行记录**

9.1项目管理机制

项目使用文档的方式传递，结合远程等方式，实现相关的项目管理，使得项目的进度和完成度处于管控之内。

9.2项目执行记录

|  |  |
| --- | --- |
| 时间 | 记录 |
| 12.18-12.20 |  |
| 12.20-12.22 |  |
| 10.22-23 |  |
| 10.24-12.25 |  |

**10. 组长对组内成员的考评打分及评价理由**

在项目过程中，每位团队成员都在自己的岗位上做出了积极的贡献，推动了项目的顺利进行。组长张雅瑞对团队成员的评分如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 褚昊 | 韩圣杰 | 雷心成 |
| 评分 | 98 | 97 | 97 |

表10-1 团队成员评分表

10.1成员评价理由

褚昊，作为UI设计师和研发工程师，在管理员用户界面的设计和实现上做出了显著的贡献。他的工作确保了系统的高效管理和监控功能。

韩圣杰，作为系统架构师和测试工程师，参与了系统整体设计，并在测试研发中发挥了重要作用。他的工作对于系统的稳定性和可靠性至关重要。

雷心成，作为算法工程师和研发工程师，在算法设计和研发上投入了大量工作，为提升系统质量提供了重要保障。

在这个项目中，每个团队成员都发挥了自己独特的技能和才能，共同努力确保了项目的成功，展示了团队合作和个人专业技能的完美结合。

**11.参考文献**

[1] Ruibin, Sun; Kui, Qian; Weimin, Xu; Hong, Lu.  Nanjing Xinxi Gongcheng Daxue Xuebao: Journal of Nanjing University of Information Science & Technology; Nanjing Vol. 13, Iss. 3, (2021): 349-354. DOI:10.13878/j.cnki.jnuist.2021.03.012

[2]郭军.基于数字图像处理的文本型数字图像OCR识别准确度提高策略研究[J].网络安全技术与应用, 2017, 000(006):143-145.DOI:CNKI:SUN:WLAQ.0.2017-06-086.

[3]宋国梁,颜长华.一种OCR图像识别的图像处理方法及系统:202010338815[P][2024-12-21].

[4]郭军.信息资源数字化文本型数字图像OCR识别准确度影响因素及提高策略研究[D].郑州大学[2024-12-21].DOI:10.7666/d.y1930059.

[5]臧国全.文本数字化图像OCR识别的准确度测度实验与提高[J].图书情报知识, 2010(3):6.DOI:CNKI:SUN:TSQC.0.2010-03-013.