5 目标检测系统设计

5.1 数据库设计

5.1.1 ER 图模式设计相关

为了实现一个将 ultralytics 库上的 yolo 模型管理，管理原图片，以及保存并管理检测结果以及其附属下的小物体的网站，根据上面的需求进行分析，并进行概念结构设计。

如图的 E-R 表，首先设计存储 model 模型信息的表，里面有 model 的相关信息，然后是 file 文件信息的表，里面包括了图片文件，也是 result 结果的来源，因为为了支持多个源图片多次检测，有多个结果，以及源图片被删不影响结果信息，所以结果信息和源图片文件信息之间不存在外键关系，用蓝色弧线表示之间存在关系。

model 文件和 result 图片文件之间是一对多关系，也就是多个目标检测结果来自一个模型上，而 result 和 obj 是一对多关系，也就是 result 结果相关信息下面包含了有那些被检测到的小物体信息。

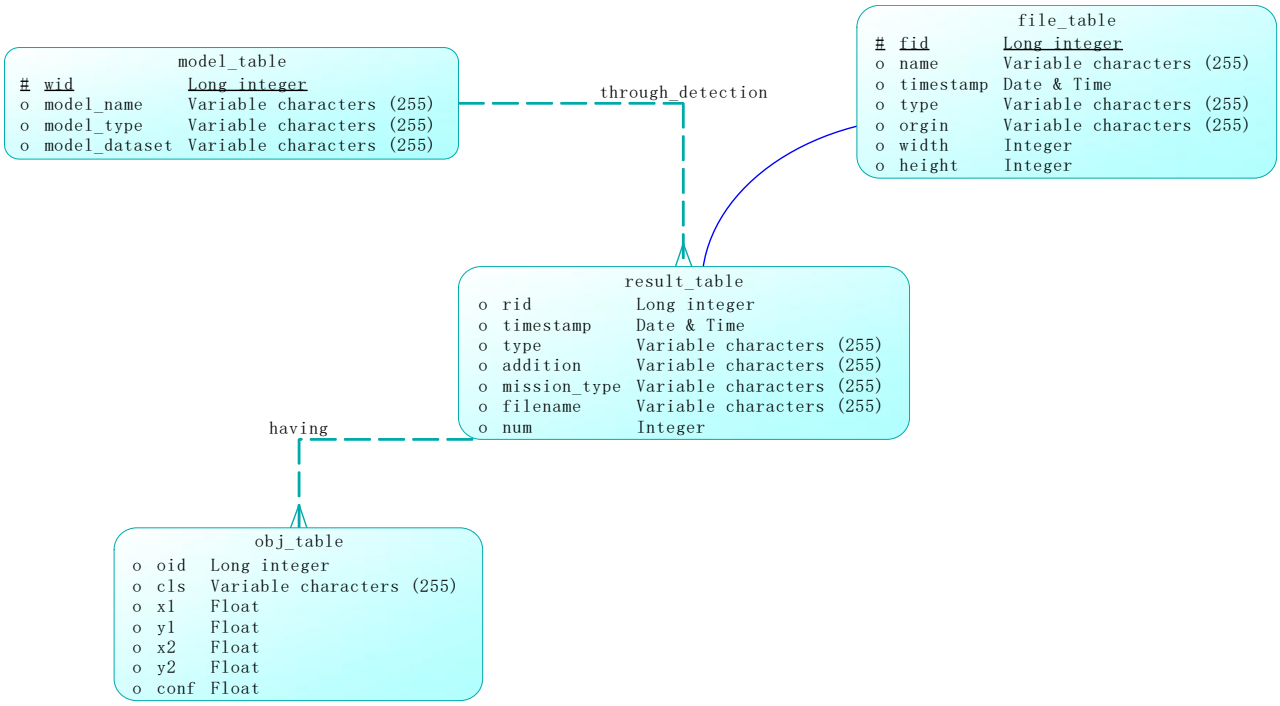

图 37 实现 VDD 算法系统平台的 E-R 图

5.1.2 数据库建表

根据该网站需要三个功能模块，以及后面的 E-R 图，进行逻辑结构设计阶段，设计出相应数据库表，下面将会用表格列出的数据库中这些表的字段对应的中英文名称、字

段类型、字段长度、以及是否主键，然后阐述这些字段的大意。

（1）model_table 文件表

model_table 表是存放相关模型文件信息的一个表，模型 ID 为其主键，文件名为这个模型名称，和后面读取模型文件、添加文件、切换文件和删除文件有很大关联。模型的类别就是这个模型是那种，yolov8 还是 yolov5,对应的数据集是从那个数据集训练来的。

表 9　模型表信息

字段中文名称	字段英文名称	字段类型	长度	是否主键
模型 ID	wid	Long integer	—	是
文件名	model_name	Variable characters	255	否
模型类别	model_type	Variable characters	255	否
对应数据集	model_dataset	Variable characters	255	否

（2） file_table 文件表

file_table 表是管理图片文件相关的表，模型 ID 为其主键，文件名为这个图片名称，时间为上传这个图片的时刻，文件类别是文件后缀，文件来源是来自本地还是其他网站，宽度和高度为这个图片的基本属性。

表 10　文件表信息

字段中文名称	字段英文名称	字段类型	长度	是否主键
文件 ID	fid	Long integer	—	是
文件名	name	Variable characters	255	否
时间	timestamp	Date &Time	—	否
文件类别	type	Variable characters	255	否
文件来源	origin	Variable characters	255	否
宽度	width	Integer	—	否
高度	height	Integer	—	否

（3） result_table 文件表

result_table 表是管理结果相关的表，结果 ID 为其主键，外键是模型 ID，来自

model_table，时间为结果出来的时间戳。

表 11　结果表信息

字段中文名称	字段英文名称	字段类型	长度	是否主键
结果 ID	rid	Long integer	—	是
模型 ID	wid	Long integer	—	否
时间	timestamp	Date &Time	—	否
任务类型	type	Variable characters	255	否
额外说明	addition	Variable characters	255	否
任务类型	mission_type	Variable characters	255	否
文件来源	filename	Variable characters	255	否
检测数目	num	Integer	—	否

（4） obj_table 文件表

file_table 表是管理图片文件相关的表，模型 ID 为其主键，文件名为这个模型名称，和后面读取模型文件、添加文件、切换文件和删除文件有很大关联。模型的类别就是这个模型是那种，yolov8 还是 yolov5,对应的数据集是从那个数据集训练来的。

表 12　Ratio 与相应变体对比表

字段中文名称	字段英文名称	字段类型	长度	是否主键
物体 ID	oid	Long integer	—	是
结果 ID	rid	Long integer	—	否
类别	cls	Variable characters	255	否
左上角横坐标	x1	Float	—	否
左上角纵坐标	y1	Float	—	否
右下角横坐标	x2	Float	—	否
右下角纵坐标	y2	Float	—	否
置信度	conf	Float	—	否

5.2 后端设计

5.2.1 对数据操作汇总

（1） GET /files/：列出所有文件信息。

（2） GET，DELETE /files/<fid>：列出单个或者是删除文件信息。

（3） POST /files：添加文件信息。

（4） GET /wmodel/：列出所有模型信息。

（5） GET, DELETE, PUT /wmodel/<wid>： 列出单个或者是删除修改模型信息，删除时会把文件模型删除掉，耦合。

（6） POST /wmodel：添加模型信息。

（7） GET /result/：列出所有结果信息。

（8） GET, DELETE /result/<rid>：列出单个或者是删除结果信息，删除时会把结果文件删掉，耦合。

（9） POST /result：添加结果信息。

（10） GET /obj/：列出所有物体信息。

（11） GET, DELETE /obj/<rid>：列出相应或者是删除外键为 rid 的所有物体信息。

（12） POST /obj：添加相应的物体信息，一般是添加好了 result 之后就会执行。

5.2.2 实现分页功能接口

（1） GET /page：列出某个表（obj_table 以外）分页之后相应的信息。

（2） GET /objectPage/<rid>：列出 obj_table 上某个 rid 分页之后相应的信息。

（3） GET /num：列出某个表（obj_table 以外）所拥有信息总数。

（4） GET /deblurS：列出某个表（obj_table 以外）实现模糊查找并分页之后相应的信息。

（5） GET /deblurSNum：列出某个表（obj_table 以外）模糊查找之后所拥有信息总数。

5.2.3 图片显示和上传文件

（1） GET /img/<name>：显示原图片到前端上。

（2） GET /resultImg/<name>：显示结果图片到前端上。

（3） POST /upload/model/：上传模型信息。

（4） POST /upload/pic/：上传图片信息。

（5） POST /upload/url：从网站爬取图片并放到后端上。

5.2.4 模型切换和目标检测

（1） GET /current：查看当前使用模型的接口。

（2） GET /switch：切换模型的接口。

（3） POST /detect：实现目标检测的功能。

（4） DELETE /delete/result：删除结果文件，在执行之后会将相关数据删除，解耦操作，和删除相应数据不在一个接口。

（5） DELETE /delete/file：删除原图片文件功能，在执行之后会将图片数据删除，解耦操作，和删除原图片的信息不在一个接口上。

5.2.5 核心功能实现

（1）爬取图片

这是/upload/url 接口的实现过程，从某个公开的图片网站上，使用 request 库上的爬虫爬取相应的图片，并保存到后端服务器的伪代码流程如下。

```
@app.route('/upload/url', methods=['POST'])
def get_url_pic():
    try:
        爬虫使用伪装头访问网站
        if 状态码 == 200:
          获取相应的内容，命名好相应文件名称
          保存文件，并从中读取到相应高度宽度信息
          return 访问成功的信息
        else:
          return 访问失败的信息
    except Exception as e:
        return 发出异常信息
```

（2）目标检测

实现后端接口/detect 的目标检测功能的伪代码流程，首先先从前端上面点击检测按钮，前端的 request 把 json 信息传入后端上，里面包括了文件名称，然后后端调用 ultralytics 库进行目标检测，把得到的图片保存下来，并返回检测得到的相关结果。

然后前端把从/detect 接口得到的信息依次 POST 到/result 和/obj，把得到的检测信息

添加到 result_table 表中和 obj_table 表中，为解耦操作。

```
@app.route('/detect', methods=['POST'])
def detection():
    try:
        从前端获取 request 的 json 信息
        命名好相应结果文件名称
        info = []
        调用 cuda 设备进行目标检测，等待的时间由自身设备算力决定
        将收集到的结果放入到 info 里面去
        对结果进行绘图，并保存命名好的文件中
        return  并检测成功的消息，并附带相关信息，为后面 POST 到 result_table 和 obj_table 提供信息
    except Exception as e:
        return  遇到异常，检测失败的信息
```

（3）上传模型文件

首先在前端把 pt 文件进行上传，然后后端判定是否为 pt 文件，最后把这个上传好的文件缓存在缓冲区中，后面把模型文件信息提交上去的话把缓冲区的文件复制到存放模型文件的目标文件夹就可以了。

```
@app.route('/upload/model/', methods=["POST"])
def save_model_cache():
    try:
        从数据流当中获取文件的 data 数据以及文件的名称，并得到文件种类 type
        if type != 'pt':
            return 不是 pt 文件的信息
        缓冲区保存文件数据
        return 模型暂缓成功的信息
    except Exception as e:
        return 模型暂缓异常信息
```

5.3 前后端核心功能交互设计

5.3.1 上传交互相关

首先用户先将图片拖入到上传区域，或者是点击上传区域把本地文件发送到前端上去，不需要用户直接填写图片的相关信息，然后前端读取到 response 中把这里的信息送到后端服务器的/files 接口，后端把上传的图片文件给解析出来，获取上面的相关消息，比如说图片的宽度、高度、来源、图片文件类别和名称等信息，把处理结果汇总之后，后端把这些信息上传到数据库以及把图片文件写入到相应收集图片的文件夹中，最后后端把操作结果返回到前端上去，如果操作结果状态是成功的话，刷新前端显示的数据。

```
// 本地文件上传相关操作
const upload = (response: any, file: any, filelist: any) => {
    axios.post(
        '/files', {
          width: response.results.width,
          height: response.results.height,
          origin: response.results.origin,
          type: response.results.type,
          name: response.results.name,
        }
    ).then(res=>{
      if(res.data.status == 'success'){
        getData()
      }
    })
};
```

5.3.2 目标检测交互相关

首先用户点击检测按钮，前端对后端的/detect 接口发送请求，后端从前端那里收到的信息进行目标检测处理，然后从得到的目标检测的数据返回前端，前端需要两次 POST 依次到不同的后端接口上，分别是/result 和/obj，添加从目标检测上得到的数据。

```
const detection = (index: number, row: any) => {
  axios({ // 首先执行检测任务，生成结果图片
    method: 'POST',
    url: '/detect',
    data: row}).then(res1=>{
    if(res1.data.status == 'success'){ // 然后再把 result 得到的 json 传入到数据库中
      axios({
        method: 'POST',
        url: '/result',
        data: res1.data.results.log
      }).then(res2=>{
        if(res2.data.status == 'success'){ // 最后再把 obj 信息传入到 obj 中
          axios({
            method: 'POST',
            url: '/obj',
            data: res1.data // 在/detect 后端哪里得到的 json 信息
          }).then(res3 => {
              if(res3.data.status == 'success'){
                ElMessage.success('目标图片检测成功，保存的文件名称是:\n'
                    + res1.data.results.log.filename + '_' +
                    res1.data.results.log.timestamp + '.' + res1.data.results.log.type);
              }
          })
        }
      ``
```

5.3.3 显示结果交互相关

用户在结果管理的页面点击查询，则会显示出结果细节等相关页面和信息，其中前端会像后端/objectPage/发送请求，把用户点击相应的 rid 信息先送到这个接口上去，然后后端在处理完相应之后把结果返回到前端上去。

```
const getResultDetail = () => {
  axios({
    method: 'GET',
    url: '/objectPage/' + obj_query.rid,
    params:{
      'curPage': obj_query.curPage,
      'pageSize': obj_query.pageSize
    }
  }).then(res=>{
    objData.value = res.data.results
    for(var i = 0;i < objData.value.length;i++){
      objData.value[i].conf = objData.value[i].conf.toFixed(3)
    }
  })
}
```

5.4 展示效果

5.4.1 图片管理页面

这是上传本地文件的前端页面，如图 38，用户直接把文件上传，或者拖拽图片文件到这个组件上就可以了，它这里上传的文件种类是有限制的，用户上传之后前端从后端处理的结果中返回相应的信息。

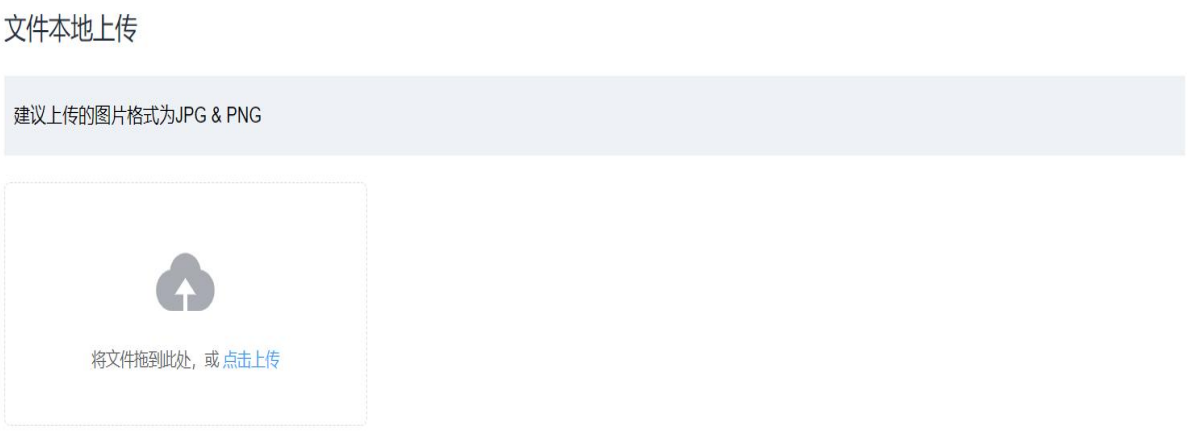

图 38 文件本地上传模块

然后，如图 39，这是 URL 地址上传，用户按照下面参考的格式，把 url 输入到输入框中，点击上传即可，然后后端爬取这个网站的图片，并把这个图片的信息添加到数据库中，最后后端把处理好的信息返回到前端上去。

如图 40，显示的页面，前端会发送分页相关的参数，从后端拉回相应的数据，其中，这个表格很清晰的把后端保存的图片信息显示了出来，后面还要删除操作，如果用户点击删除，系统就会提示确定删除的页面，用户点击是，然后前端向后端响应删除操作，后端把结果返回到前端上，刷新页面。

如图 41，这是一个实现模糊查找的 UI 接口，用户把文件名称输入到这个输入框中，然后点击搜索，就可以显示这个文件名称相关的结果，并且是分页的，如果要跳出模糊查找，则点击列出全部的按钮即可。

图 39 URL 地址上传模块

204	MVI_40853_img01154		Tue, 28 Feb 2023 09:32:58 GMT	jpg	localhost://MVI_40853_img01154.jpg	960	540	删除
205	MVI_40854_img00109		Tue, 28 Feb 2023 09:32:58 GMT	jpg	localhost://MVI_40854_img00109.jpg	960	540	删除
206	MVI_40855_img00649		Tue, 28 Feb 2023 09:32:58 GMT	jpg	localhost://MVI_40855_img00649.jpg	960	540	删除
207	t01e1dfab1a5278d1a5_1677575202		Tue, 28 Feb 2023 17:06:42 GMT	jpg	https://p1.ssl.qhmsg.com/t01e1dfab1a5278d1a5.jpg	1280	853	删除
208	8114-e6ab336af96e393f3f312a58349114d8_1677575687		Tue, 28 Feb 2023 17:14:48 GMT	png	https://n.sinaimg.cn/sinakd20220516s/290/w1080h810/20220516/8114-e6ab336af96e393f3f312a58349114d8.p	1080	810	删除

图 40 图片管理模块

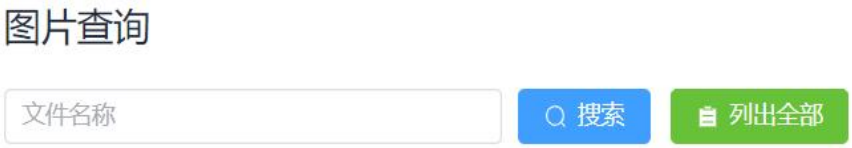

图 41 实现分页模糊查找的输入框

5.4.2 目标检测和模型管理页面

用户首先先填好模型名称，如图 42，对应数据集名称，选好模型种类，因为文件读取操作很大程度上是由这个模型名称决定的，所以模型名称不能修改，编辑页面图 43 也是如此，除非把它删除再提交上去，然后上传文件那里用户要把文件上传，前端把文件交到后端缓冲区上，最后提交表单的时候，前端向后端发出请求，后端保存模型的数据信息到数据库上，同时把缓冲区的文件保存到后端相应放模型文件的文件夹中。

图 42 模型信息提交模块

图 43 编辑模型信息表

如图 44 和图 45，当前后端使用的模型前端会显示出来，此外，用户可以在模型管理中选择相应的模型，来切换当前后端正在使用的模型，此外还有编辑删除操作，有模糊查找模型名称并且的分页功能。

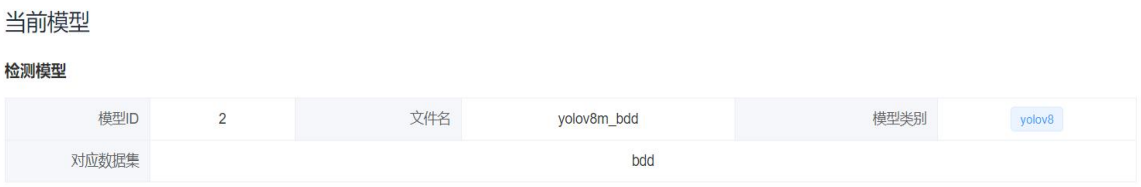

图 44 当前模型信息显示模块

图 45 模型管理模块

这是图片检测的页面相关，如图 46，这个输入框功能和上面实现模糊分页查找的原理是差不多的，首先在图片管理检测模块点击用户想要检测的图片，点击检测按钮一次，然后等待几秒种，最后页面会通知相关信息，如果检测成功则弹出绿色成功框等信息。

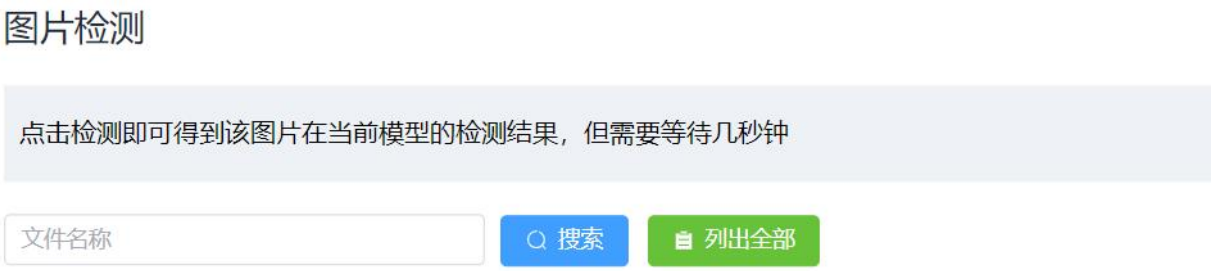

图 46 实现分页模糊查找的输入框

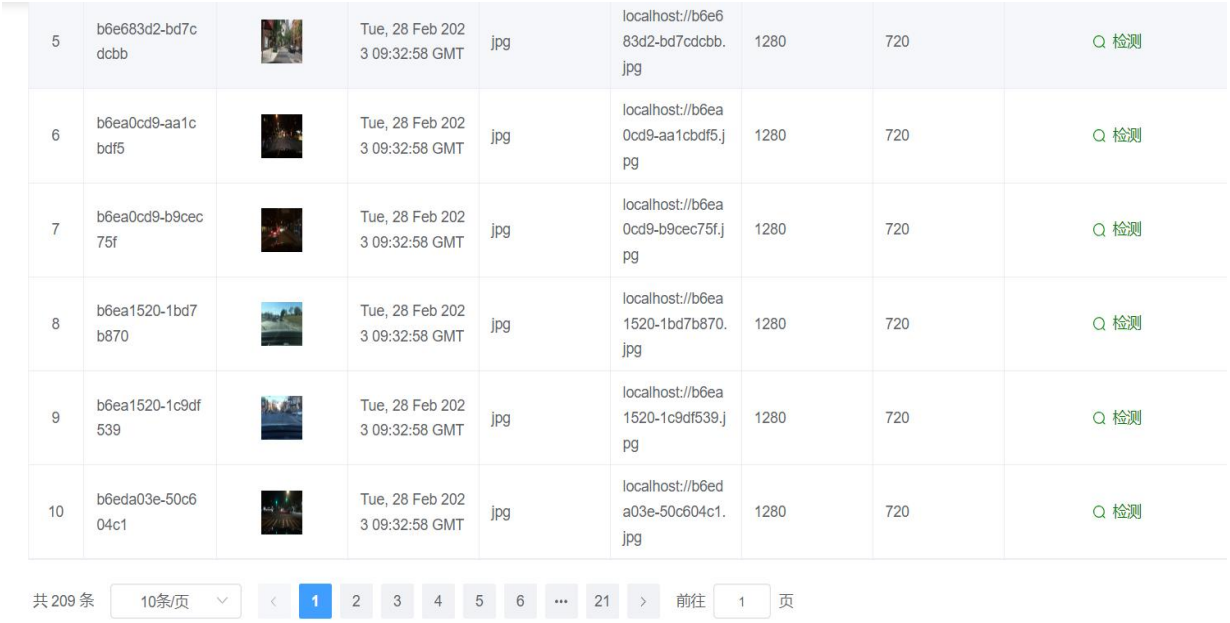

图 47 图片管理检测模块

5.4.3 检测结果页面

检测结果页面上面有模型和图片相关信息，只不过只提供查询不通供删除，和上面之前讲的原理一样，如图所示。

记录管理也有实现模糊分页查找的输入框、列出结果结束模糊查找之类的功能，这个表格清晰显示出检测结果等相关信息，操作有查看、编辑和删除，图为编辑页面。

点击查看的话会看到放大的原图和其他详细信息的内容，如图和图所示，编辑结果的话只能编辑额外说明和任务类别内容，删除是讲这个对应结果删除掉并刷新页面。

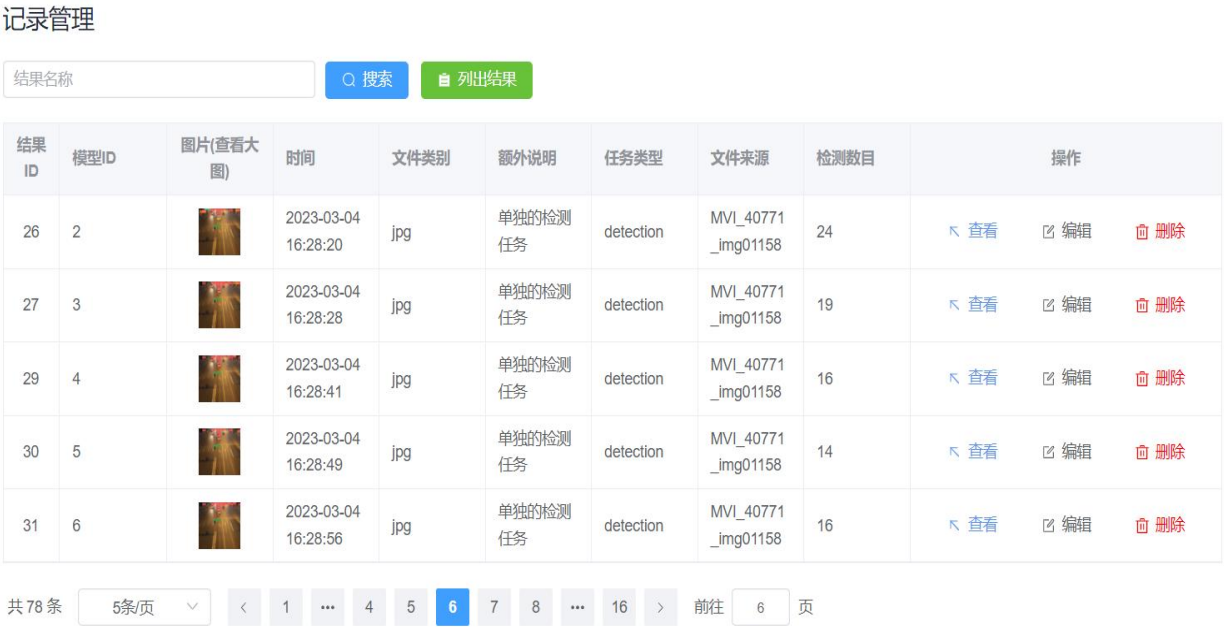

图 48 记录管理模块

图 49 修改记录模块

原图显示

图 50 记录原图显示模块

任务信息

时间	2023-03-04 16:28:20	文件类别	jpg	额外说明	单独的检测任务
任务类型	detection	文件来源	MVI_40771_img01158		

检测模型

模型ID	2	文件名	yolov8m_bdd	模型类别	yolov8
对应数据集	bdd				

物体细节

物体ID	结果ID	类别	置信度	左上角横坐标	左上角纵坐标	右下角横坐标	右下角纵坐标
483	26	car	0.824	538	1	570	24
484	26	car	0.805	413	4	455	44
485	26	car	0.804	420	57	470	96
486	26	car	0.804	699	11	774	39
487	26	car	0.786	436	85	498	141

共 24 条 5条/页 1 2 3 4 5 前往 1 页

图 51 记录其他信息模块

5.5 总结

本章介绍了使用 flask 后端和 vue + vite 前端的大体实现过程。首先先介绍数据库的设计过程，然后是后端的接口列出，并伪代码讲解关键核心功能，然后讲前后端交互逻辑，最后展示前端的结果以及相关模块大体实现的功能是什么样子的，具体开源代码在附录上。