北京郵電大學


# 作业 3：
# 信息抽取系统实验报告

学院：计算机学院（国家示范性软件学院）

专业：计算机科学与技术

班级：2022211305

学号：2022211683、2022211124、2022211130

姓名：张晨阳、梁维熙、金建名

2025 年 6 月 6 号

# 目录

# 1. 项目概述

信息提取工具是一个基于 PyQt5 的桌面应用程序，用于从用户输入的文本中提取结构化信息，并提供反馈机制来改进提取结果的准确性。该工具与后端服务(运行在 127.0.0.1:5000)进行交互，实现信息提取和反馈管理功能。

# 2. 功能特性

## 2.1. 核心功能

- 文本信息提取：用户输入文本后，工具将文本发送到后端服务进行处理，返回结构化提取结果
- 结果展示：以清晰易读的格式展示提取的字段和对应值
- 详细内容查看：对于长文本内容，提供"查看详细"功能
- 反馈机制：允许用户标记不正确的字段，帮助改进提取算法

## 2.2. 辅助功能

- 历史记录查看：显示过去的信息提取记录和字段准确率
- 错误标记提示：对曾被标记为错误的字段进行醒目提示
- 响应式 UI 设计：适配不同大小的窗口和内容

# 3. 技术架构

## 3.1. 前端技术

PyQt5：用于构建 GUI 界面

QSS 样式表：用于美化界面元素

网格布局(GridLayout)：用于灵活排列提取结果字段

## 3.2. 后端交互

RESTful API：通过 HTTP POST 请求与后端服务通信

JSON 数据格式：用于前后端数据交换

## 3.3. 主要组件

主窗口(InfoExtractorApp)：包含文本输入区域和功能按钮

提取结果对话框：展示提取的字段和值

历史记录对话框：以表格形式展示历史记录

详细内容对话框：显示字段的完整内容

# 4. 后端设计与实现

## 4.1. 语料准备

在作业三中，我们同样利用作业二中设计的爬虫程序从 zenodo 中爬取相应的学术论文摘要文档共 500 篇，作为系统的测试语料。语料格式如下：

```
Title: Machine Learning at Telescope Array
Authors: Kharuk, Ivan
Keywords:
Publication date: 2025-03-12
Description:
Telescope Array is a large-scale cosmic-ray observatory studying ultra-high-energy cosmic rays. Its Surface Detector array consists of 507 scintillation stations arranged in a rectangular grid covering approximately 700 km². This talk presents our deep learning approach to reconstructing cosmic ray properties from Telescope Array Surface Detector data. We demonstrate how combining multiple data representations with various neural architectures (convolutional, recurrent, and transformer networks) enhances reconstruction accuracy of primary particle properties, including arrival direction and energy. Finally, we present post-processing techniques developed for searching for rare event, such as ultra-high-energy photons.
Contents:
 - TA_ML_Delaware_final.pdf
```

语料中包含明确的关键信息，如标题，作者，关键词等，我们完成程序对语料中的关键信息进行提取。

## 4.2. 程序设计

在后端中，我们需要完成使用正则表达式进行相应的关键信息的提取，并提供人工评价功能，对于之前出错的提取文本进行提示。

### 信息提取

我们在程序中使用 python 提供的正则表达式包 re 进行正则表达式的书写以进行相应的信息的提取，我们书写的正则表达式如下：

- 标题：

```
match = re.search(r"Title:\s*(.+)", text)
```

- 作者：

```
match = re.search(r"Authors:\s*(.+)", text)
```

- 关键词：

```
match = re.search(r"Keywords:\s*(.+)", text)
```

- 发布日期：

```
match = re.search(r"Publication date:\s*([\d-]+)", text)
```

- 介绍：

```
description_match = re.search(r"Description:\s*(.*?)(?:\Z)", text,
re.DOTALL)
```

- 网页：

```
url_pattern = re.compile(r'https?://[^\s"<]+|www\.[^\s"<]+')
```

- DOI：

```
doi_pattern = re.compile(r'10\.\d{4,9}/[^\s"<]+')
```

在提取完成后，我们会进行记录的保存。同时，使用 uuid 模块分配一个独一无二的 uuid 来记录本次的提取，给我们的人工评价部分留下使用的接口。uuid 模块是 Python 的标准库之一，用于生成通用唯一标识符（UUID）。UUID 是一种软件构建中常用的标识符，广泛用于数据库主键、会话标识、分布式系统中的唯一标识等场景。服务端本地的记录如下：


```
  "018f35e7-0ad7-4d5f-b544-a09cc6c6a73f": {
    "Title": "Machine Learning at Telescope Array",
    "Authors": [
      "Kharuk",
      "Ivan"
    ],
    "Keywords": [
      "Publication date: 2025-03-12"
    ],
    "Publication Date": "2025-03-12",
    "Description": "Telescope Array is a large-scale cosmic-ray
observatory studying ultra-high-energy cosmic rays. Its Surface
Detector array consists of 507 scintillation stations arranged in a
rectangular grid covering approximately 700 km². This talk presents our
deep learning approach to reconstructing cosmic ray properties from
Telescope Array Surface Detector data. We demonstrate how combining
multiple data representations with various neural architectures
(convolutional, recurrent, and transformer networks) enhances
```

```
reconstruction accuracy of primary particle properties, including
arrival direction and energy. Finally, we present post-processing
techniques developed for searching for rare event, such as ultra-high-
energy photons.\n\n\nContents:\n - TA_ML_Delaware_final.pdf",
    "URLs": [],
    "DOIs": [],
    "GitHub Link": null,
    "DOI": null
  },
```

通过 uuid 进行提取记录的索引，并记录提取过的字段信息。

## 人工评价

在该部分，我们通过使用 json 文件记录每次提取文件中的字段的正确与否，用户在前端进行相对应的提取错误标记时，服务端会对相对应的提取记录中的内容进行错误记录。实现代码如下：

```
def save_feedback(uuid, field, correct):
    """
    保存用户对提取结果的反馈。
    """
    entry = {"correct": correct}
    key = generalize_key(uuid, field)
    feedback_data = []
    if os.path.exists(FEEDBACK_FILE):
        try:
            with open(FEEDBACK_FILE, "r", encoding="utf-8") as
feedback_file:
                feedback_data = json.load(feedback_file)
        except json.JSONDecodeError:
            # 如果文件为空或损坏，则重新初始化
            feedback_data = {}
    feedback_data[key] = entry
    with open(FEEDBACK_FILE, "w", encoding="utf-8") as feedback_file:
        json.dump(feedback_data, feedback_file, indent=2,
ensure_ascii=False)
```

由于 FEEDBACK_FILE 中需要使用的主键为复合健，我们通过字符串拼接的方式将复合键转换为单主键：

```
def generalize_key(uuid, field):
    return "uuid:" + uuid + "field" + field
```

程序将记录的用户反馈输出至 FEEDBACK_FILE 中。同时，用户还可以查

看提取历史，用户指定想要知道的提取历史数量，服务端会根据数量从保存提取记录的 json 文件中读取最新的历史记录，同时会根据这些记录的 uuid 从 FEEDBACK_FILE 中读取相对应的字段评价记录，并使用提取记录相同的方式组成字典，将两个字典发送给前端交予其进行展示。

```
        # 获取最新的 num 条记录
        latest_records = dict(list(all_extracted_data.items())[-num:])
        correct_history = {}  # 初始化 correct_history 为一个空字典

        for uuid, data in latest_records.items():
            total_fields = 0
            correct_fields = 0
            correct_history[uuid] = {}
            for field in data.keys():
                key = generalize_key(uuid, field)
                if key in feedback_data:
                    total_fields += 1
                    correct_history[uuid][field] =
feedback_data[key]["correct"]
                    if feedback_data[key]["correct"]:
                        correct_fields += 1
                else:
                    correct_history[uuid][field] = None  # 如果没有反馈，则
标记为 None
            if total_fields > 0:
                accuracy = correct_fields / total_fields
            else:
                accuracy = None
            data["accuracy"] = accuracy
        return jsonify({"status": "success", "history": latest_records,
"correct_history": correct_history}), 200
```

## 后端服务支持

在本程序中使用 flask 模块进行相对应的后端服务支持。Flask 是一个用 Python 编写的轻量级 Web 应用框架。它简单易用，适合快速开发小型到中型的 Web 应用。鉴于本程序仅为简单的交互 api 支持，需要快速开发，故使用本模块。

该模块通过使用装饰器@app.route()来进行使用的域名的设置，当前端访问装饰器中指定的域名时会调用该函数，如果是通过 POST 方法进行调用那么 flask 模块会将调用时携带的 request 交予该函数，函数中直接使用 request 变量即可进

行访问。flask 的服务启动可以通过调用 app.run()函数或是通过命令

```
gunicorn -w 4 app:app
```

运行。

除此之外，flask 模块提供了 jsonify 函数以简单生成相对应的 json 数据包 response。使用@app.route()装饰器进行修饰的函数返回值除了 response 中的内容还需要携带网络代码。

通过使用该模块，本程序共提供以下 api 便于前后端交互

- "/extract"：该接口接受 POST 方法，其接受的 request 和 response 如下：

```
    @request:
        - Content-Type: application/json
        - Body: {"text": "文本内容"}
    @response:
        - 200 OK: {"status": "success", "data": {"Title": "标题",
"Authors": ["作者 1", "作者 2"], ...}, "uuid": uuid}
        - 400 Bad Request: {"error": "请求必须是 JSON 格式"}
        - 400 Bad Request: {"error": "请求体中缺少 'text' 字段"}
        - 500 Internal Server Error: {"error": "服务器内部错误：错误信息"}
```

- "/feedback"：该接口接受 POST 方法，其接受的 request 和 response 如下：

```
    @request:
        - Content-Type: application/json
        - Body: {"uuid": uuid, "field": "字段名", "correct": true/false}
    @response:
        - 200 OK: {"status": "success", "message": "反馈已保存"}
        - 400 Bad Request: {"error": "请求必须是 JSON 格式"}
        - 400 Bad Request: {"error": "请求体中缺少 'field', 'value' 或
'correct' 字段"}
        - 400 Bad Request: {"error": "'correct' 字段必须是布尔值
(true/false)"}
        - 500 Internal Server Error: {"error": "服务器内部错误：错误信息"}
```

- "/feedback/check_incorrect"：该接口接受 POST 方法，其接受的 request 和 response 如下：

```
    @request:
        - Content-Type: application/json
        - Body: {"uuid": uuid, "field": "字段名"}
    @response:
        - 200 OK: {"status": "success", "was_incorrect": true/false}
        - 400 Bad Request: {"error": "请求必须是 JSON 格式"}
```

```
        - 400 Bad Request: {"error": "请求体中缺少 'field' 或 'value' 字段
"}
        - 500 Internal Server Error: {"error": "服务器内部错误：错误信息"}
```

- "/feedback/history"：该接口接受 POST 方法，其接受的 request 和 response 如下：

```
    @request
        - Content-Type: application/json
        - Body: {"Num": num}
    @response:
        - 200 OK: {"status": "success", "history": [uuid1: data, uuid2:
data, ...], "correct_history": [uuid1: data, uuid2: data, ...]}
        - 400 Bad Request: {"error": "请求体中缺少 'Num' 字段"}
        - 400 Bad Request: {"error": "'Num' 字段必须是正整数"}
        - 500 Internal Server Error: {"error": "服务器内部错误：错误信息"}
```

# 5. 代码结构分析

## 5.1. 主要类和方法

### InfoExtractorApp 类

- __init__：初始化主窗口和 UI 组件
- init_ui：设置界面布局和样式
- extract_info：发送文本到后端进行信息提取
- show_extracted_info：展示提取结果对话框
- show_detail：显示字段详细内容
- submit_extracted_feedback：提交用户反馈到后端
- show_history：显示历史记录对话框
- load_history：从后端加载历史记录数据

## 5.2. 关键设计点

**1. 响应式布局：**

使用 QScrollArea 确保大量提取结果可滚动查看

字段采用网格布局，保持整齐排列

**2. 用户体验优化：**

长文本内容截断显示，提供"查看详细"按钮

曾被标记为错误的字段有醒目提示

统一的样式和配色方案

**3. 错误处理：**

网络请求异常捕获和处理

后端返回错误信息的友好展示

# 6. 关键代码分析

## 6.1. 主窗口初始化与 UI 构建

### 主窗口设置

```
def __init__(self):
    super().__init__()
    self.setWindowTitle("信息提取工具")  # 设置窗口标题
    self.setGeometry(100, 100, 800, 600)  # 设置窗口位置和大小
    self.init_ui()  # 初始化 UI
```

### UI 组件初始化

```
def init_ui(self):
    # 中央部件和主布局
    self.central_widget = QWidget()
    self.setCentralWidget(self.central_widget)
    self.layout = QVBoxLayout()
    self.central_widget.setLayout(self.layout)

    # 文本输入区域
    self.text_edit = QTextEdit()
    self.text_edit.setPlaceholderText("请输入文本...")
    self.layout.addWidget(self.text_edit)

    # 提取按钮
    self.extract_button = QPushButton("提取信息")
    self.extract_button.clicked.connect(self.extract_info)
    self.layout.addWidget(self.extract_button)

    # 历史记录按钮
    self.history_button = QPushButton("查看反馈历史")
    self.history_button.clicked.connect(self.show_history)
    self.layout.addWidget(self.history_button)
```

## 6.2. 核心功能实现

### 信息提取功能

```
def extract_info(self):
    text = self.text_edit.toPlainText()  # 获取用户输入的文本
    if not text:
        QMessageBox.warning(self, "警告", "请输入文本！")
        return

    try:
        # 发送 POST 请求到后端服务
        response = requests.post("http://127.0.0.1:5000/extract",
json={"text": text})
        response.raise_for_status()  # 检查 HTTP 错误
        data = response.json()  # 解析 JSON 响应

        if data["status"] == "success":
            # 成功获取数据后显示提取结果
            extracted_info = data["data"]
            self.show_extracted_info(extracted_info, data["uuid"])
        else:
            QMessageBox.warning(self, "错误", data.get("error", "未知错误
"))
    except requests.RequestException as e:
        QMessageBox.warning(self, "错误", f"请求失败: {str(e)}")
```

### 提取结果展示

```
def show_extracted_info(self, extracted_info, uuid):
    # 创建结果展示对话框
    self.extracted_info_dialog = QDialog(self)
    self.extracted_info_dialog.setWindowTitle("提取结果")
    self.extracted_info_dialog.setGeometry(100, 100, 1200, 800)

    # 使用滚动区域容纳大量字段
    scroll_area = QScrollArea()
    scroll_area.setWidgetResizable(True)
    self.extracted_info_dialog.layout = QVBoxLayout()
    self.extracted_info_dialog.setLayout(self.extracted_info_dialog.layo
ut)

    # 内容部件和网格布局
```

```
    content_widget = QWidget()
    content_layout = QGridLayout()
    content_widget.setLayout(content_layout)

    self.checkboxes = {}  # 存储字段复选框
    row = 0
    for field, value in extracted_info.items():
        # 字段名标签
        field_label = QLabel(f"{field}:")
        field_label.setStyleSheet("font-weight: bold;")
        content_layout.addWidget(field_label, row, 0)

        # 处理字段值显示
        if isinstance(value, list):
            truncated_value = ", ".join(value)
        elif value is None:
            truncated_value = "None"
        else:
            truncated_value = value

        # 长文本截断处理
        if isinstance(truncated_value, str) and len(truncated_value) >
50:
            truncated_value = truncated_value[:50] + "..."

        # 值标签
        value_label = QLabel(truncated_value)
        content_layout.addWidget(value_label, row, 1)

        # 详细查看按钮
        detail_button = QPushButton("查看详细")
        detail_button.clicked.connect(lambda checked, v=value:
self.show_detail(v))
        content_layout.addWidget(detail_button, row, 2)

        # 检查字段是否曾被标记为错误
        is_incorrect = self.check_field_incorrect(uuid, field)
        if is_incorrect:
            error_label = QLabel("曾被标记为错误，注意鉴别")
            error_label.setStyleSheet("color: red; font-weight: bold;")
            content_layout.addWidget(error_label, row, 4)

        # 反馈复选框
        checkbox = QCheckBox("标记为不正确")
```

```
            checkbox.setChecked(is_incorrect)
            self.checkboxes[field] = checkbox
            content_layout.addWidget(checkbox, row, 3)

            row += 1

        scroll_area.setWidget(content_widget)
        self.extracted_info_dialog.layout.addWidget(scroll_area)

        # 反馈提交按钮
        submit_button = QPushButton("提交所有反馈")
        submit_button.clicked.connect(lambda:
self.submit_extracted_feedback(uuid))
        self.extracted_info_dialog.layout.addWidget(submit_button)

        self.extracted_info_dialog.exec_()
```

## 6.3. 辅助功能实现

### 详细内容查看

```
def show_detail(self, value):
    # 创建详细内容对话框
    detail_dialog = QDialog(self)
    detail_dialog.setWindowTitle("详细内容")
    detail_dialog.setGeometry(200, 200, 600, 400)

    layout = QVBoxLayout()
    detail_dialog.setLayout(layout)

    # 处理不同类型的内容显示
    if isinstance(value, list):
        value = ", ".join(value)
    elif value is None:
        value = "None"

    # 使用 QTextBrowser 显示完整内容
    text_browser = QTextBrowser()
    text_browser.setText(value)
    layout.addWidget(text_browser)

    detail_dialog.exec_()
```

## 反馈提交功能

```
def submit_extracted_feedback(self, uuid):
    # 收集所有字段的反馈状态
    feedback_data = {field: checkbox.isChecked() for field, checkbox in
self.checkboxes.items()}

    try:
        # 为每个字段提交反馈
        for field, is_incorrect in feedback_data.items():
            response = requests.post(
                "http://127.0.0.1:5000/feedback",
                json={"uuid": uuid, "field": field, "correct": not
is_incorrect}
            )
            response.raise_for_status()
            data = response.json()
            if data["status"] != "success":
                QMessageBox.warning(self, "错误", data.get("error", "未知
错误"))

        QMessageBox.information(self, "成功", "反馈已保存！")
        self.extracted_info_dialog.close()
    except requests.RequestException as e:
        QMessageBox.warning(self, "错误", f"请求失败: {str(e)}")
```

## 历史记录功能

```
def show_history(self):
    # 创建历史记录对话框
    self.history_dialog = QDialog(self)
    self.history_dialog.setWindowTitle("历史记录")
    self.history_dialog.setGeometry(100, 100, 800, 600)

    # 创建表格显示历史记录
    self.history_table = QTableWidget()
    self.history_table.setRowCount(0)
    self.history_table.setColumnCount(3)
    self.history_table.setHorizontalHeaderLabels(["UUID", "字段名", "准确
率"])
    self.history_table.horizontalHeader().setStretchLastSection(True)
    self.history_table.horizontalHeader().setSectionResizeMode(QHeaderVi
ew.Stretch)
```

```
    # 加载历史数据
    self.load_history()

    # 布局设置
    layout = QVBoxLayout()
    layout.addWidget(self.history_table)

    # 关闭按钮
    close_button = QPushButton("关闭")
    close_button.clicked.connect(self.history_dialog.close)
    layout.addWidget(close_button)

    self.history_dialog.setLayout(layout)
    self.history_dialog.exec_()

def load_history(self):
    try:
        # 从后端获取历史记录
        response =
requests.post("http://127.0.0.1:5000/feedback/history", json={"Num":
10})
        response.raise_for_status()
        data = response.json()

        if data["status"] == "success":
            # 更新表格数据
            self.update_history_table(data["history"],
data["correct_history"])
        else:
            QMessageBox.warning(self, "错误", data.get("error", "未知错误
"))
    except requests.RequestException as e:
        QMessageBox.warning(self, "错误", f"请求失败: {str(e)}")

def update_history_table(self, history, correct_history):
    self.history_table.setRowCount(0)  # 清空表格

    # 填充表格数据
    for uuid, data in history.items():
        row_position = self.history_table.rowCount()
        self.history_table.insertRow(row_position)

        # 添加 UUID、字段名和准确率数据
```

```
        self.history_table.setItem(row_position, 0,
QTableWidgetItem(uuid))
        for field, correct in correct_history[uuid].items():
            self.history_table.setItem(row_position, 1,
QTableWidgetItem(field))
            self.history_table.setItem(row_position, 2,
QTableWidgetItem(str(correct)))
```

## 6.4. 关键设计亮点

### 网格布局与滚动区域结合：

- 使用 QGridLayout 整齐排列提取结果字段
- QScrollArea 确保大量字段可滚动查看

### 反馈机制实现：

- 为每个字段提供复选框标记正确性
- 曾被标记为错误的字段有醒目提示
- 批量提交所有字段反馈

### 历史记录管理：

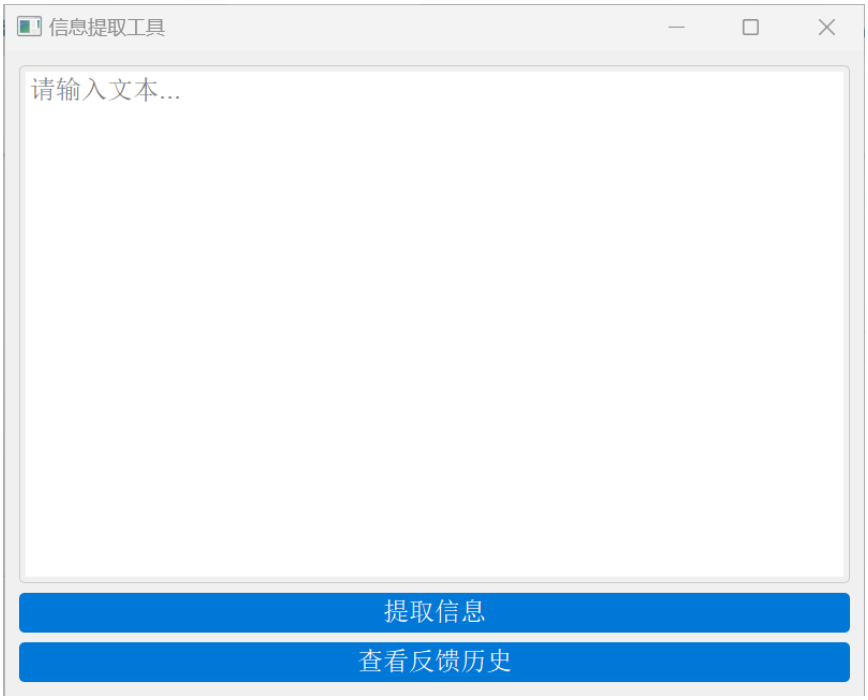
- 表格形式清晰展示历史记录
- 支持查看多个字段的准确率统计

### 错误处理与用户体验：

- 网络请求异常捕获和友好提示
- 长文本内容的截断显示和详细查看
- 统一的样式和交互设计

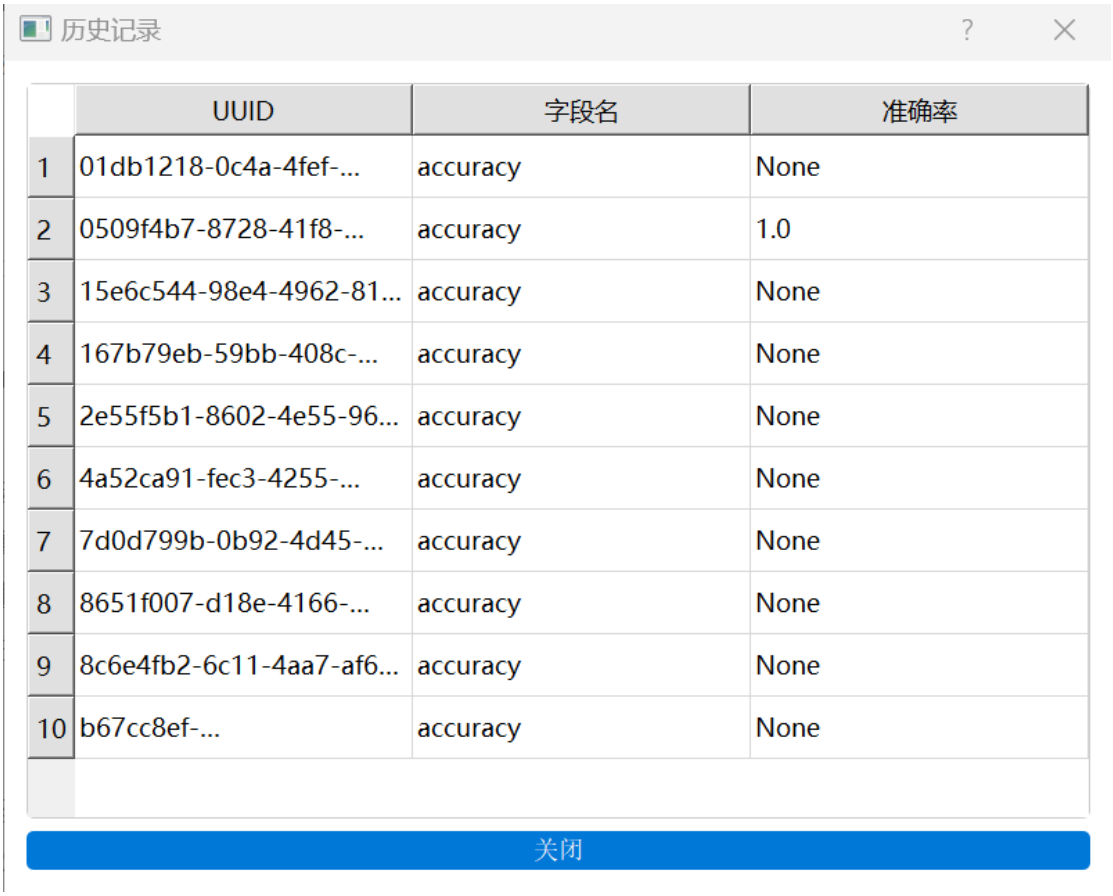
# 7. 项目概览

## 7.1. 主界面




## 7.2. 提取信息结果界面

## 7.3. 历史记录界面




# 8. 总结

该信息提取工具设计合理，功能完整，具有良好的用户体验。通过 PyQt5 实现了美观的界面，并通过与后端服务的交互实现了核心的信息提取和反馈功能。代码结构清晰，易于维护和扩展，为进一步功能增强奠定了良好基础。