**哈尔滨工业大学（深圳）**

**大一年度项目结题报告**

**项目名称：** 通知内容管理App的设计与实现

**项目负责人：** 王靳  **学号：** 220111012

**联系电话：** 15816870583 **电子邮箱：** 220111012@stu.hit.edu.cn

**院系及专业：** 计算机科学与技术学院

**指导教师：** 吴宇琳  **职称：** 助理教授

**联系电话：** 18126282493  **电子邮箱：** [wuyulin@hit.edu.cn](mailto:wuyulin@hit.edu.cn)

**院系及专业：** 计算机科学与技术学院

**填表日期： 2023 年 12 月 13 日**

**一、项目团队成员**（包括项目负责人、按顺序）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 性别 | 所在学院 | 学号 | 联系电话 | 本人签字 |
| 王靳 | 男 | 计算机科学与技术学院 | 220111012 | 15816870583 |  |
| 吴语诗 | 女 | 计算机科学与技术学院 | 220110928 | 15967167116 |  |
| 蔡德林 | 男 | 理学院 | 220810316 | 15919094899 |  |
| 邹悦 | 女 | 理学院 | 220810424 | 18820366233 |  |
|  |  |  |  |  |  |

**二、指导教师意见**

|  |
| --- |
| 签 名：  年 月 日 |

**三、项目专家组意见**

|  |
| --- |
| 组长签名：  年 月 日 |

**四、项目成果**

|  |
| --- |
| 1、实现了LayUI + Flask的前端，LayUI是一个Web UI包，提供了美观的前端界面设计。通过LayUI和Flask的结合，实现了用户友好的前端交互界面。  2、通过SQLAlchemy构架了Notice的ORM模型，实现了app与数据库（mysql/maridb）之间的交互。这样可以方便地对数据库进行操作，实现数据的存储和检索。  3、通过Hugging Face的Transformers，进行了迁移学习，将一个二分类的模型修改输出层，使其支持12输出，实现了12分类。这样可以更加灵活地应用模型进行多类别的分类任务。  4、通过正则表达式，能够将通知中的日期进行正则匹配，并将其输出到app内置的日历中。这样可以方便用户查看通知中的时间信息，并进行日程安排。  5、通过ajax实现了网页与服务端的交互，提升了用户体验，使得前端界面更加流畅和便捷。这些为app的功能和性能提升提供了有力支持。 |

**五、项目研究结题报告**（字数应在3000字左右）

**（一）主要内容：**

1 课题背景

课题背景随着信息化发展，通知更多借助网络渠道。学校目前使用的飞书App仍存在重要通知被淹没、通知对象针对性不强、通知本身信息冗杂等问题。学长学姐开发的App不支持自动生成关于截止日期的提醒。大多数同学需要在群消息中反复寻找、查看同一条通知，时间利用效率低。项目计划设计并实现一个通知内容管理App，实现学校通知精准分类、要点捕捉与简化、重要通知收藏与推荐、日程安排表个性化生成五项功能，希望服务于学院通知发布工作。

2 课题研究内容与方法

#### 2.1 研究内容

本项目通过对 “文本挖掘”的研究，利用相关算法将学院以大段文本形式呈现、信息糅合一体的通知抽象成一个个简单标签，可以实现对标签的分类，并且查询到相关的日期。此外，我们还实现了“文本摘要”，可以将大段冗余的通知总结成一段精简的通知

#### 2.2 实施方案

本次项目的实施方案分为三个阶段：基础知识的学习阶段、前端开发阶段和后端开发阶段。

##### 2.2.1 第一阶段：基础学习

小组成员学习了相关知识：python 的基本用法，学习了基于 pytorch 的模型搭建与调试，学习了 transformers 的调试方法，学习了 “迁移学习” 的 机器学习方法，学习了 css，javascript 语法，学习了 flask 框架、ajax 开发前端，学习了如何使用 SQLAlchemy。学习了 docker 部署等技巧，以及通过 git 进行版本的管理。

##### 2.2.2 第二阶段：前端开发

在前端开发中，项目采用了非常简单易用的 LayUI 组件库。Layui 是一套免费的开源 Web UI 组件库，在 flask 中的 jinja2 模板中调用了 LayUI的相关组件。通过 SQLAlchemy创建了通知（notice）的 ORM 模型，这使得开发人员能够简单且安全地与本地数据库进行交互。下面是项目server的目录结构：

1. ╰─➤  tree
2. .
3. ├── app.py
4. ├── blueprints
5. │   └── notice.py
6. ├── config.py
7. ├── data\_utils.py
8. ├── exts.py
9. ├── models.py
10. ├── model
11. │   ├── config.json
12. │   ├── pytorch\_model.bin
13. │   ├── special\_tokens\_map.json
14. │   ├── tokenizer\_config.json
15. │   └── vocab.txt
16. ├── statics
17. │   ├── image（文件夹，jinja2 模板中一些图片）
18. │   ├── js（文件夹，里面有jQuery，被layui、templates/\*依赖）
19. │   └── layui（文件夹，web ui 组建库）
20. ├── templates
21. │   ├── add.html
22. │   ├── base.html
23. │   ├── calendar.html
24. │   ├── search.html
25. │   └── categories.html
26. ├── tokenizers\_pegasus.py

下面是这个目录中的每个文件的介绍（根据拓扑顺序）：

* templates 文件夹里面存放的是 jinja2 模板，json 数据传递通过 ajax 实现
  + base.html是其他 html 的底板，里面是整个网页的主题，可以在其他的 templates html 中看到有{% extends "base.html" %}这句话，也可以理解成面向对象的继承关系
  + add.html是添加通知的前端页面，能够与我们的 server 进行交互
  + calender.html用到了 layUI 的日历组件，能够展示日期与对应的通知
  + categories.html是显示通知及其分类的页面
  + search.html是搜索页面
* exts.py只进行了一个简单的功能——打开数据库
* config.py是与数据库交流的必要配置，帐号，密码，端口等。
* models.py并不是人工智能的模型，而是与数据库交流的 ORM 模型
* data\_utils.py是 fengshen 模型库处理数据的辅助文件，被tokenizer\_pegasus.py依赖。fengshen github仓库的README中推荐我们将data\_utils.py与tokenizers\_pegasus.py这两个文件放到项目中。（fengshen 是 idea-ccnl 研究院针对中文训练的一系列大模型）
* tokenizers\_pegasus.py是 fengshen 模型库的 tokenizers。这个文件被notice.py依赖，在文本摘要的时候，用来将输入的文本编码。pegasus就是一个专门用来处理文本摘要的模型。
* static只是静态文件，简而言之就是——资源包。
* blueprints/notices.py用来处理通知的，里面有文本摘要，模糊搜索，添加通知，文本分类的方法。
* model文件夹里存放着：字典vocab.txt，超参数tokenizer\_config.json，special\_tokens\_map.json，config.json，以及训练好的模型pytorch\_model.bin。
* app.py是top\_module，能路由网页、启动网页，调用了config.py，exts.py，notice.py，templates等文件。

##### 2.2.3 第三阶段：后端开发

这就需要翻阅server之外的文件了。

1. (base) ╭─wangfiox@localhost ~/Documents/freshman\_project/utils  ‹main\*›
2. ╰─➤  tree
3. .
4. ├── classification
5. │   ├── classification.ipynb
6. │   └── logs（文件夹，训练的日志）
7. ├── data
8. │   ├── x\_通知搜集.md （人工搜集的400条通知，有几个文件）
9. │   ├── combined\_data.csv
10. │   ├── csv\_into\_db.ipynb
11. │   ├── dates\_combined\_data.csv
12. │   ├── dates\_combined\_data.ipynb
13. │   ├── dates\_combined\_data\_summarized.csv
14. │   ├── extract\_content\_to\_csv.ipynb
15. │   └── test.csv
16. └── summary
17. ├── data\_utils.py
18. ├── summary.ipynb
19. └── tokenizers\_pegasus.py

* summary文件夹下，小组成员测试了“文本摘要”的效果
  + summary.ipynb用于测试IDEA-CCNL/Randeng-Pegasus-523M-Summary-Chinese-V1模型的效果
  + tokenizers\_pegasus.py 与 data\_utils.py 在上文已经介绍过了
* data文件夹中有：小组成员搜集的若干数据的原始文件，处理原始文件的脚本，将文本导入数据库的脚本
* classification实现了“迁移学习”——将“二分类”转化为了“12分类”。

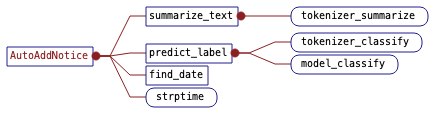
2.3 具体功能的实现

为了明确将要操作的数据，让我们先来看看ORM模型：

1. class NotificationModel(db.Model):
2. \_\_tablename\_\_ = "notification\_data"
4. id = db.Column(db.Integer, primary\_key=True, autoincrement=True)
5. title = db.Column(db.String(200), nullable=False)
6. content = db.Column(db.Text, nullable=False)
7. *# 提取日期*
8. date = db.Column(db.DateTime, nullable=True)
9. *# 摘要*
10. summary = db.Column(db.Text, nullable=True)
11. created\_date = db.Column(db.DateTime, default=datetime.now)

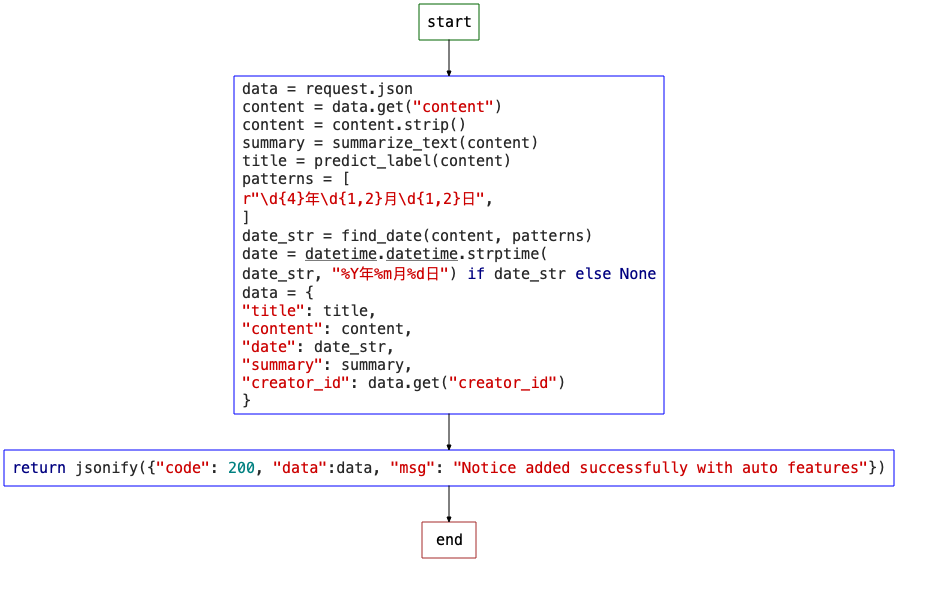
在接下来的介绍中，将以“自动识别”功能为核心，根据依赖关系递归地介绍具体功能的实现。

（1）“自动识别”（对应于前端的“自动识别”按钮）功能的实现：下面是该功能（方法）的调用关系图



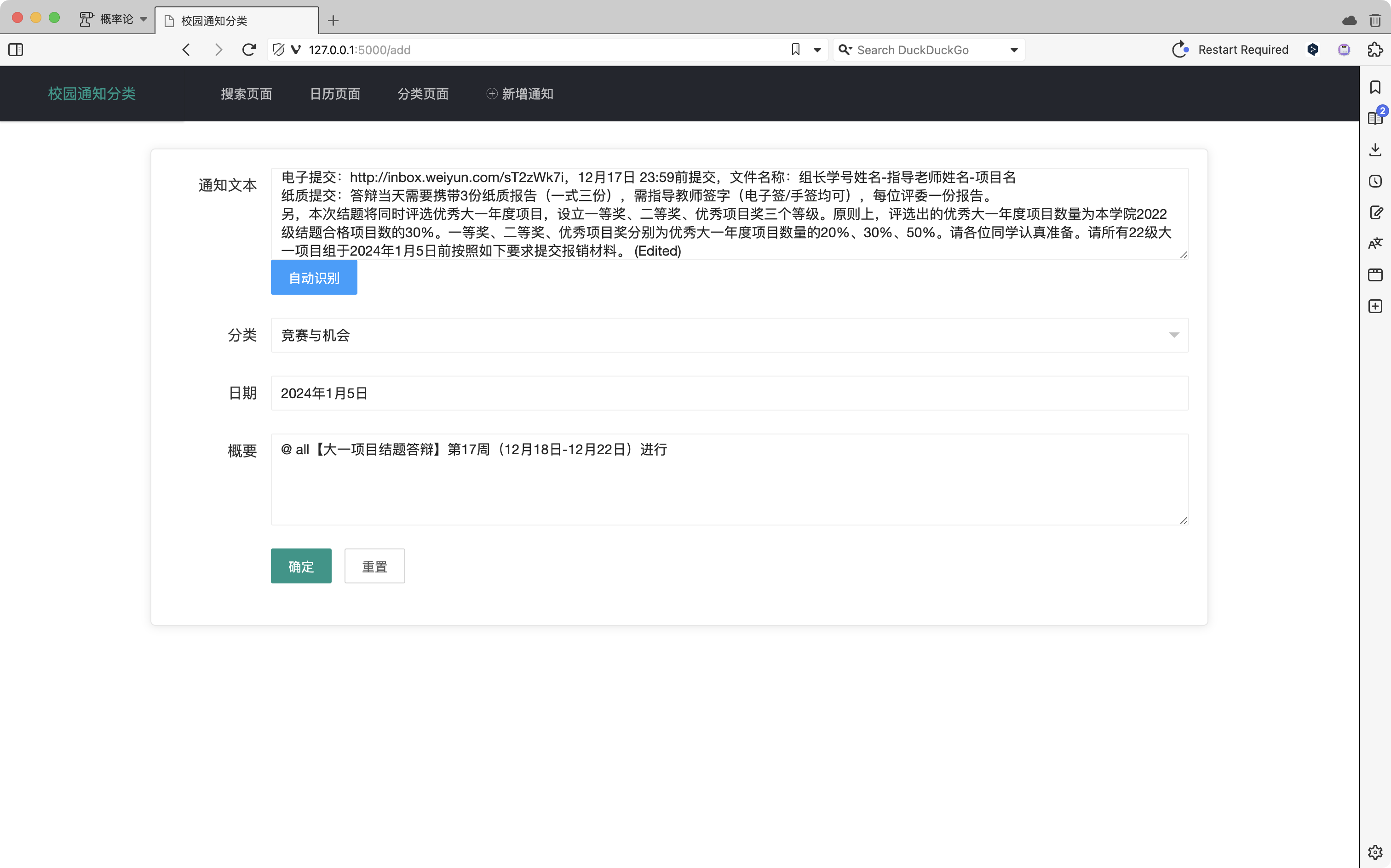
[图1 - “自动识别”的调用关系图]

* summarize\_text方法用于“文本摘要”
* predict\_label方法用于“文本分类”
* find\_date方法用于正则的寻找日期
* strptime方法用于将我们找到的日期，处理成"%Y年%m月%d日"的格式



[图2 - “自动识别”的流程图]

当点击“自动识别”后，AutoAddNotice会调用data.get("content")得到前端输入的文本。然后规范化文本-->文本摘要-->文本分类-->正则化匹配日期-->将数据封装成json-->返回到前端。



[图3 - “自动识别”效果展示]

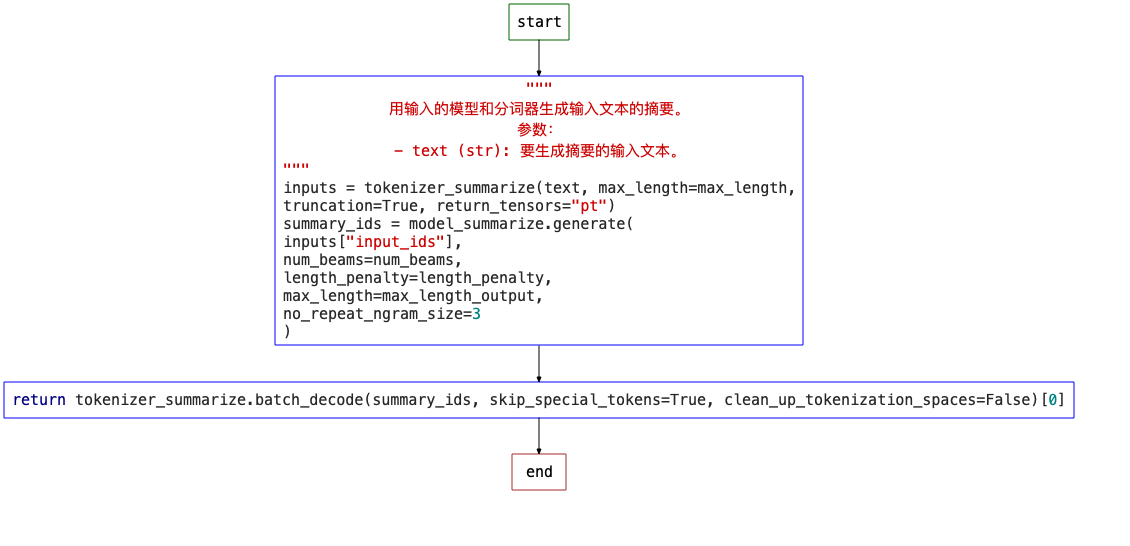
（这里的“分类”、“日期”、“摘要”是点击“自动识别”生成的）

（2）“文本摘要”的实现，下面是该功能的调用关系



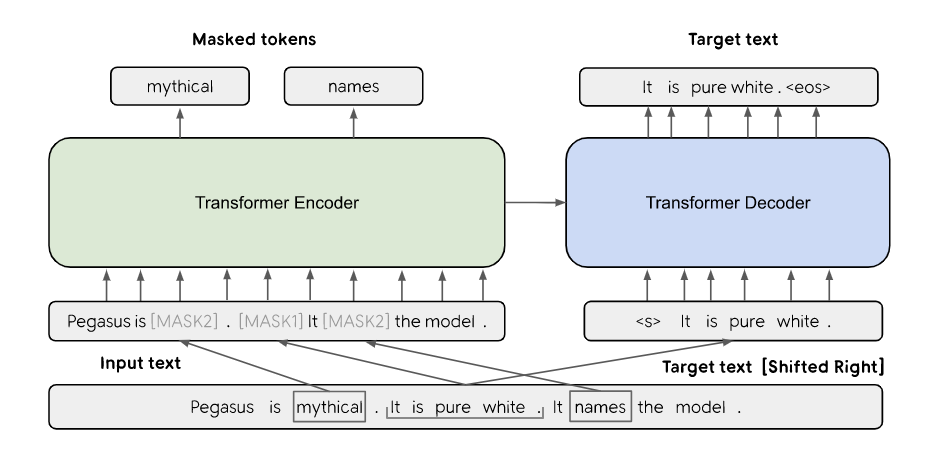
[图4 - “文本摘要”的调用关系]

* tokenizers\_summarize方法由IDEA-CCNL/Randeng-Pegasus-523M-Summary-Chinese-V1模型的配套文件tokenizers\_pegasus .py提供



[图5 - “文本摘要”的流程图]

PEGASUS模型介绍：



[图6 - Pegasus模型]

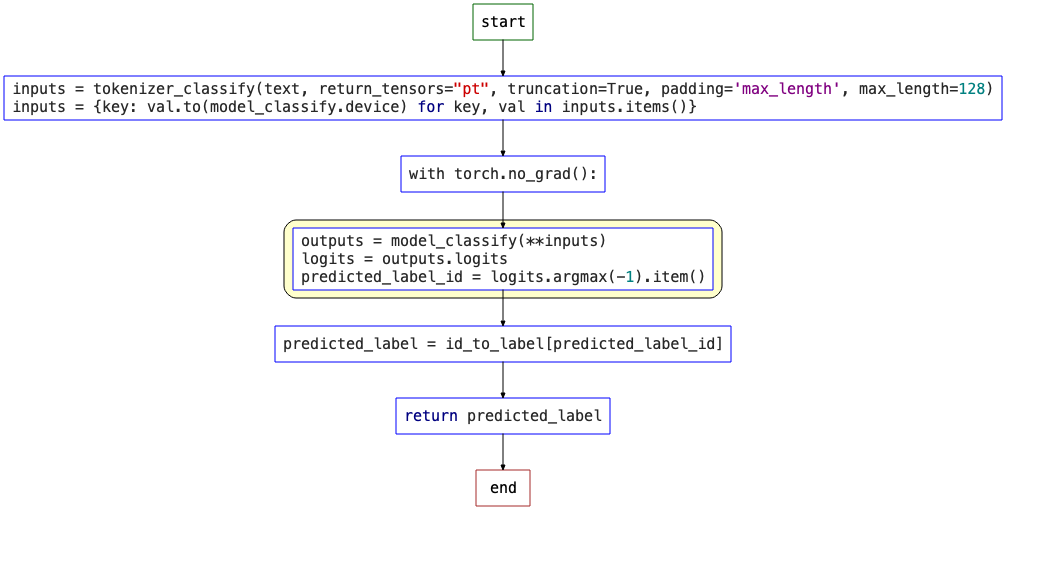
Pegasus模型是一种seq2seq模型，改进了T5与BART模型。T5模型将不同掩码程度的文本进行训练。BART在T5的基础上，引入了denoising的autoencoder层。Pegasus在BART的基础上，不只是对词语进行掩码，还对整个句子剪切到decoder层的输入中。

（3）“文本分类”的实现，下面是该功能的调用关系



[图7 - “文本分类”的调用关系]

* tokenizer\_classify用于将文本向量化，调用model/vocab.txt和tokenizer\_config.json
* model\_classify用于文本分类，调用我们训练好的模型



[图8 - “文本分类”的流程图]

首先对输入的文本进行分词处理，然后将向量转移到适配的设备上。如果CUDA可用，将数据转移到显存上，否则将在内存中进行处理。接下来，使用上下文管理器with，在预测过程中关闭梯度计算。然后进行预测，并最终返回预测ID对应的文本标签。

（4）“文本分类”之“迁移学习”

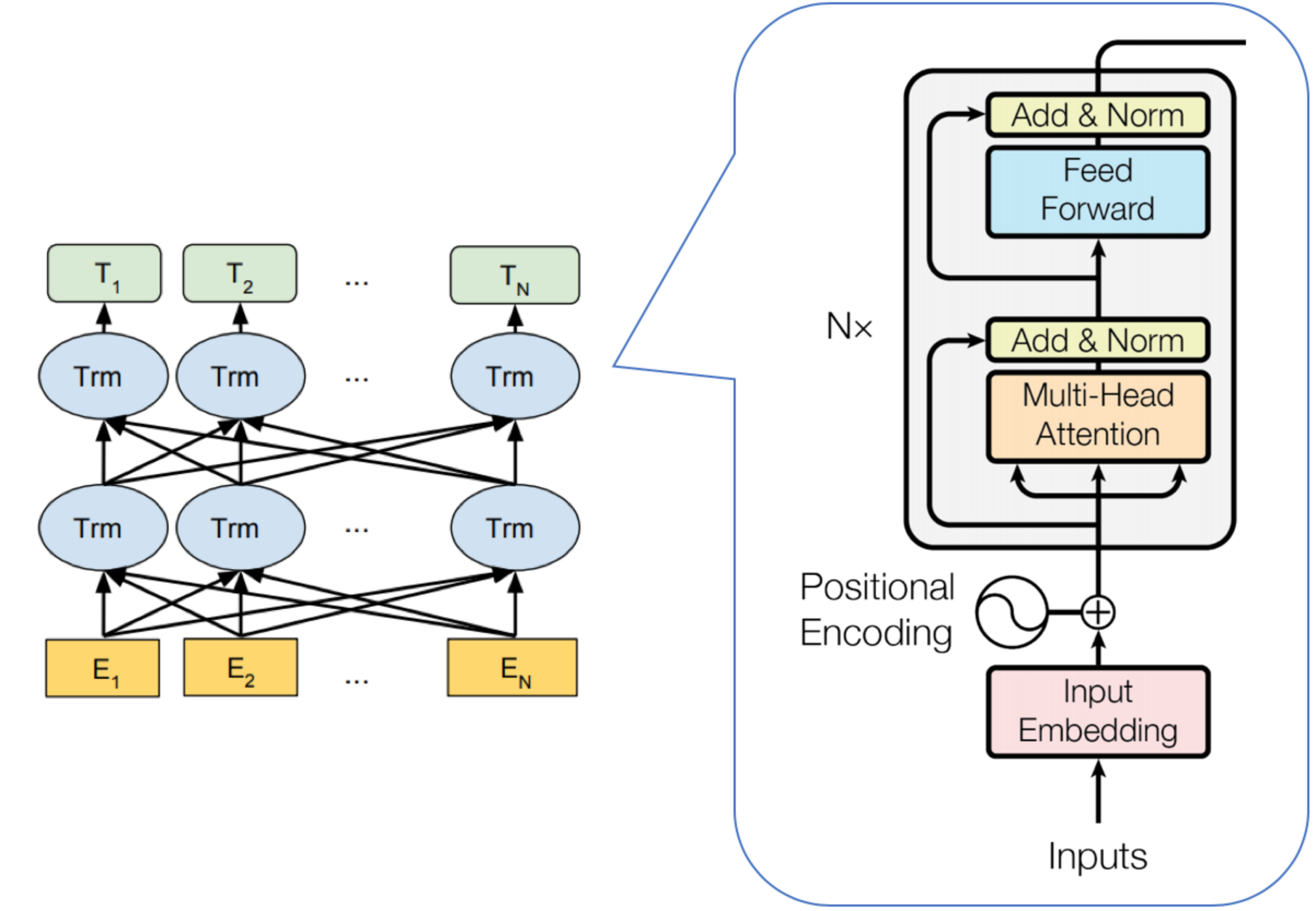
“文本分类”利用了RoBERTa预训练模型。RoBERTa模型是Bert模型的改进版:

1. 动态掩码与静态掩码结合

2. 使用了“没有NSP loss的FULL-SENTENCES”的训练策略

3. Training with large batches

因为RoBERTa是对BERT模型的改进，下面介绍一下BERT模型。



[图9 - BERT模型]

我们可以看见，BERT实际上是一个encoder层的叠加。最后如果我们想要输出一个分类结果，那么添加一个线性层和一个softmax激活层即可。本次项目的RoBERTa模型,层次结构如下：

1. BertForSequenceClassification(
2. (bert): BertModel(
3. (embeddings): BertEmbeddings(
4. (word\_embeddings): Embedding(21128, 768, padding\_idx=1)
5. (position\_embeddings): Embedding(512, 768)
6. (token\_type\_embeddings): Embedding(2, 768)
7. (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise\_affine=True)
8. (dropout): Dropout(p=0.1, inplace=False)
9. )
10. (encoder): BertEncoder(
11. (layer): ModuleList(
12. (0-11): 12 x BertLayer(
13. (attention): BertAttention(
14. (self): BertSelfAttention(
15. (query): Linear(in\_features=768, out\_features=768, bias=True)
16. (key): Linear(in\_features=768, out\_features=768, bias=True)
17. (value): Linear(in\_features=768, out\_features=768, bias=True)
18. (dropout): Dropout(p=0.1, inplace=False)
19. )
20. (output): BertSelfOutput(
21. (dense): Linear(in\_features=768, out\_features=768, bias=True)
22. (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise\_affine=True)
23. (dropout): Dropout(p=0.1, inplace=False)
24. )
25. )
26. (intermediate): BertIntermediate(
27. (dense): Linear(in\_features=768, out\_features=3072, bias=True)
28. (intermediate\_act\_fn): GELUActivation()
29. )
30. (output): BertOutput(
31. (dense): Linear(in\_features=3072, out\_features=768, bias=True)
32. (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise\_affine=True)
33. (dropout): Dropout(p=0.1, inplace=False)
34. )
35. )
36. )
37. )
38. (pooler): BertPooler(
39. (dense): Linear(in\_features=768, out\_features=768, bias=True)
40. (activation): Tanh()
41. )
42. )
43. (dropout): Dropout(p=0.1, inplace=False)
44. (classifier): Linear(in\_features=768, out\_features=12, bias=True)
45. )

（4.a）测试预训练模型

1. *# 使用预训练模型*
2. model\_name = 'IDEA-CCNL/Erlangshen-Roberta-110M-Sentiment'
3. tokenizer = BertTokenizer.from\_pretrained(model\_name)
4. model = BertForSequenceClassification.from\_pretrained(model\_name)  *# 预训练模型*
5. 2.2.2.2、测试 二分类 结果
6. texta = '鲸鱼是哺乳动物，所有哺乳动物都是恒温动物'
7. textb = '鲸鱼也是恒温动物'
8. output = model(torch.tensor([tokenizer.encode(texta, textb)]))
9. print(torch.nn.functional.softmax(output.logits, dim=-1))  *# 测试一下*

OUTPUT:

1. tensor([[0.0645, 0.9355]], grad\_fn=<SoftmaxBackward0>)

可以看到，第一句话模型认为的并不是很正确（但是实际上应该是对的）；第二句话模型认为是对的。

（4.b）数据预处理

加载数据，将文本 tokenize，将数据集划分为：训练集，测试集。因为预训练模型本身就比较大，就没有进行网格搜索，交叉验证，因此没有验证集。

1. from datasets import load\_dataset, Features, Value
2. label\_to\_id = {  *# 分类*
3. "升学": 0,
4. "志愿": 1,
5. "教务": 2,
6. "思政": 3,
7. ... // 等等
8. }
9. *# 将label\_to\_id进行反转*
10. id\_to\_label = {value: key for key, value in label\_to\_id.items()}
11. print(id\_to\_label)
12. def preprocess\_function(batch):
13. *# 对通知内容进行分词，并返回结果*
14. encoding = tokenizer(
15. batch["通知内容"], truncation=True, padding="max\_length", max\_length=128
16. )  *# 分词，截断，填充*
17. encoding["labels"] = [
18. label\_to\_id[label] for label in batch["类别"]
19. ]  *# 使用label\_to\_id将类别名转换为ID*
20. return encoding
21. *# 明确地定义CSV数据的特征描述*
22. features = Features({"类别": Value("string"), "通知内容": Value("string")})
23. *# 使用提供的特征描述加载数据集*
24. dataset = load\_dataset("csv", data\_files="../data/combined\_data.csv", features=features)
25. *# 使用map函数进行预处理*
26. encoded\_dataset = (
27. dataset["train"]
28. .map(preprocess\_function, batched=True)
29. .train\_test\_split(test\_size=0.05)
30. )
31. *# 数据集的分割*
32. train\_dataset = encoded\_dataset["train"]
33. test\_dataset = encoded\_dataset["test"]

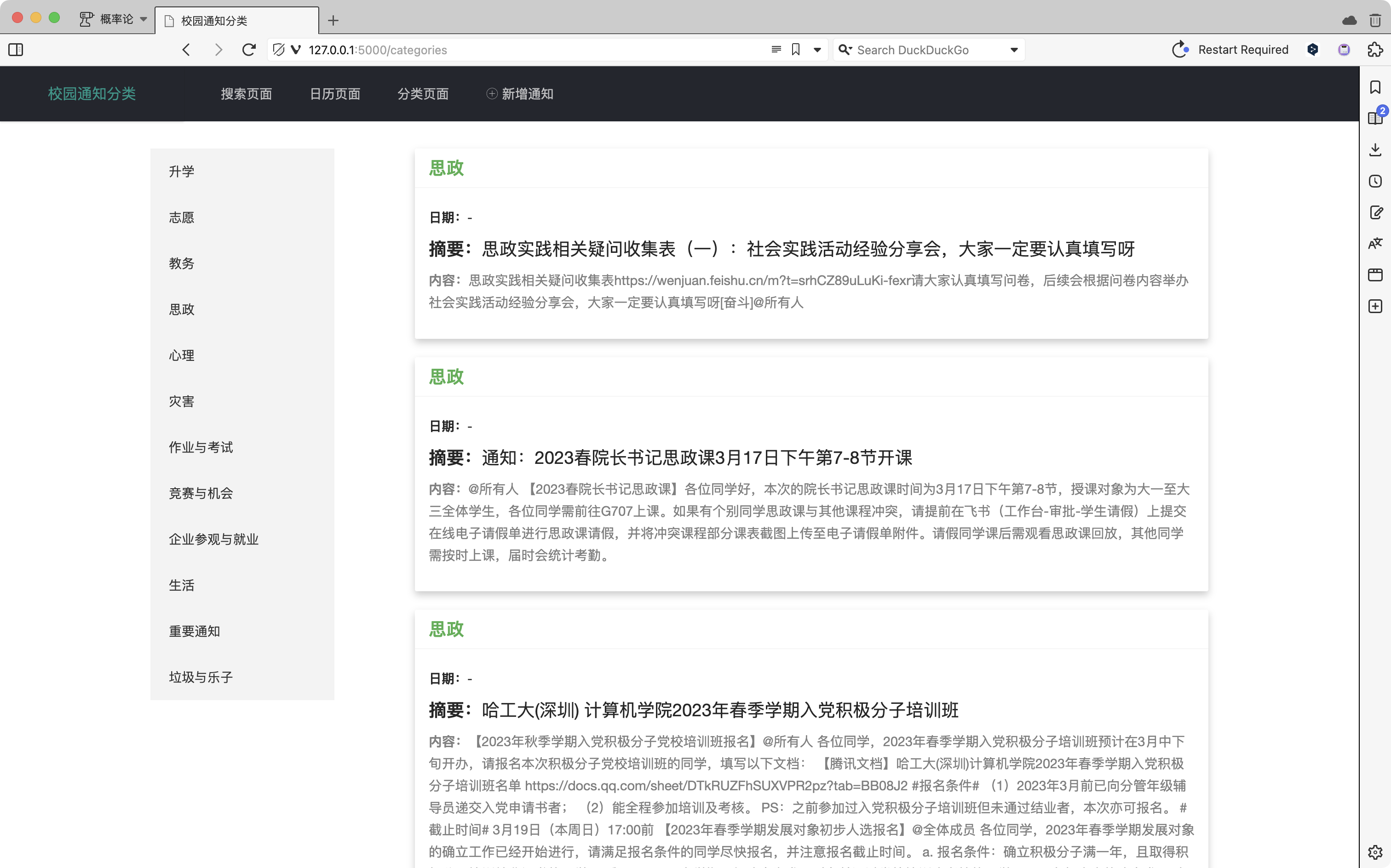
（4.c）修改输出层

1. *# 修改模型输出*
2. num\_labels = len(label\_to\_id)
3. model = BertForSequenceClassification.from\_pretrained(
4. model\_name, num\_labels=num\_labels, ignore\_mismatched\_sizes=True
5. )
6. *# 打印模型的最后一层，验证是12分类*
7. print(model.classifier)

OUTPUT：Linear(in\_features=768, out\_features=12, bias=True)

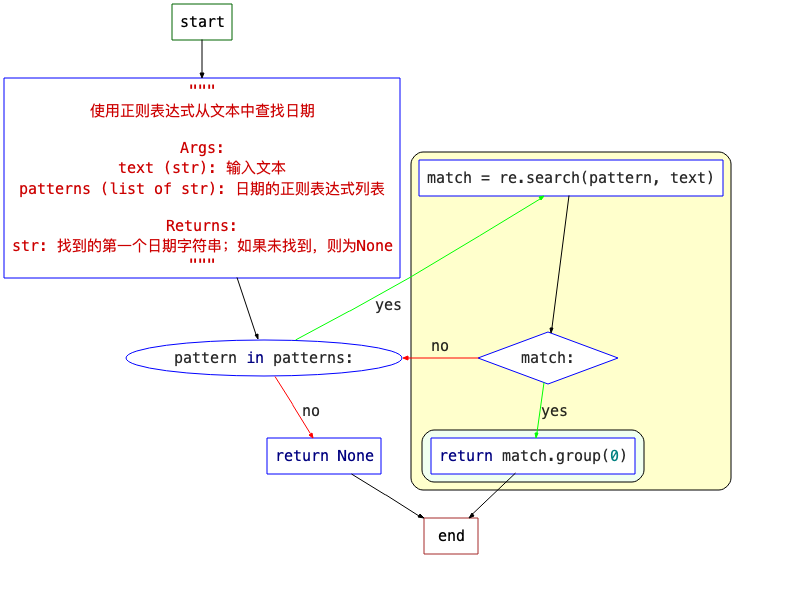
（4.d）激动人心的trainer.train()

1. from transformers import Trainer, TrainingArguments
2. *# 定义训练参数*
3. training\_args = TrainingArguments(
4. output\_dir="./results",
5. evaluation\_strategy="steps",
6. eval\_steps=10,
7. per\_device\_train\_batch\_size=64,
8. per\_device\_eval\_batch\_size=128,
9. num\_train\_epochs=3,
10. save\_steps=50,
11. logging\_steps=20,
12. learning\_rate=2e-5,
13. weight\_decay=0.01,
14. logging\_dir="./logs",
15. load\_best\_model\_at\_end=True,
16. )
17. *# 创建Trainer对象*
18. trainer = Trainer(
19. model=model,
20. args=training\_args,
21. train\_dataset=train\_dataset,
22. eval\_dataset=test\_dataset,
23. compute\_metrics=None,  *# 如果你需要在验证时计算评估指标，请提供一个compute\_metrics函数*
24. )
25. *# 启动！*
26. trainer.train()



[图10 - “文本分类”效果展示]

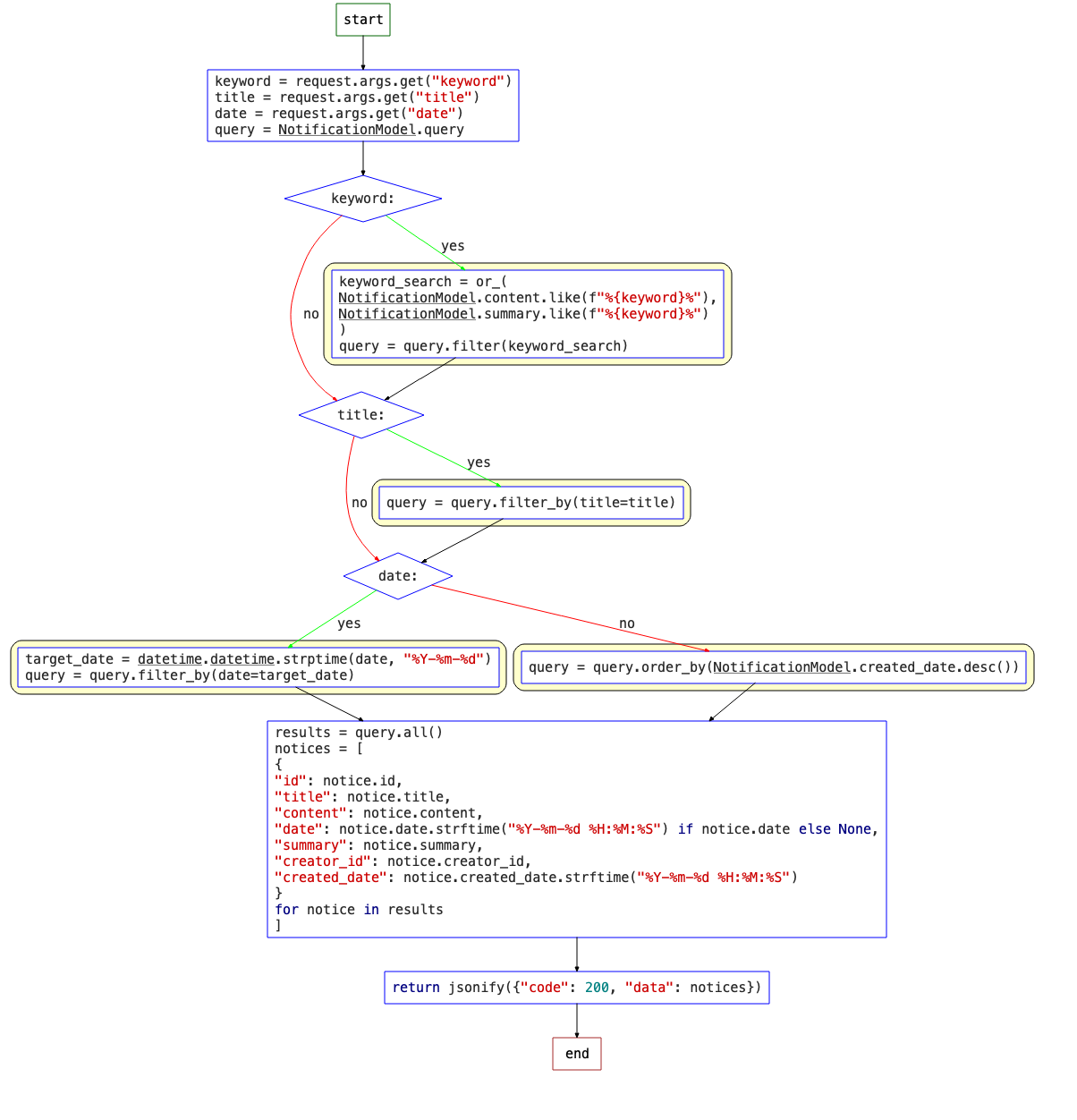
（5）正则化搜索日期，下面是“正则化搜索日期”的流程图



[图11 - “正则化搜索日期”的流程图]

流程说明：find\_date函数接收text和patterns两个参数。patterns是日期格式的列表，因为学院所推送的通知，其日期格式并非是一成不变的。在函数内部遍历所有的模式串，让其与传入的文本进行匹配。如果有匹配成功，返回第一个找到的日期；如果匹配失败，返回NULL。

（6）模糊搜索，下面是“模糊搜素”的流程图



[图12 - “模糊搜素”的流程图]

流程：先对输入的文本进行搜索，再是摘要的搜索，再是日期的搜索，这些搜索都是借助数据库完成。最后打包成json，传到前端。

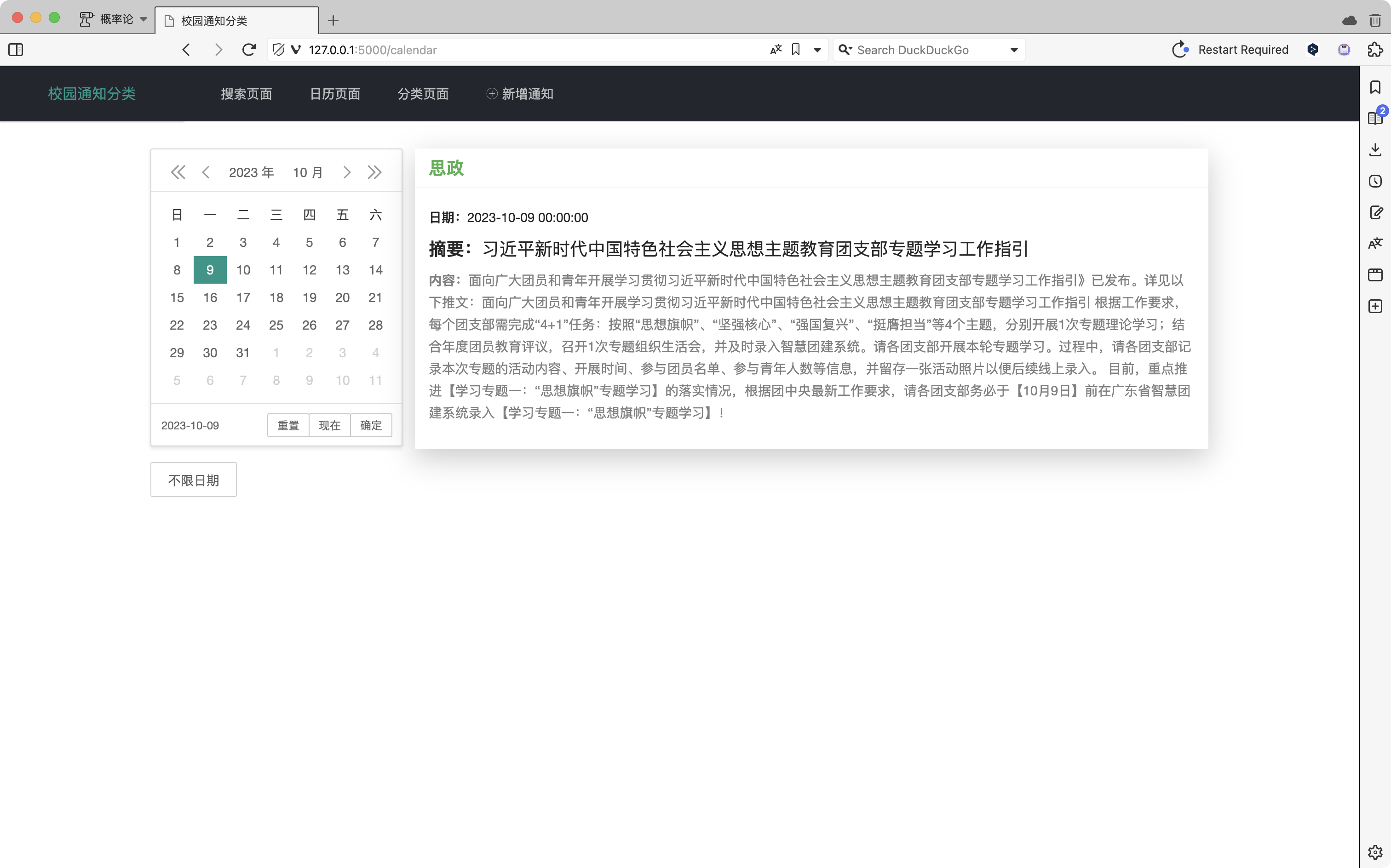
（7）生成日程表的前后端代码

（7.a）生成日程表的后端代码

1. @app.route('/calendar')
2. def calendar():
3. *# 使用模板插件，引入index.html。此处会自动Flask模板文件目录寻找index.html文件。*
4. return render\_template('calendar.html', name='calendar')

（7.b）生成日程表的前端代码

1. {% extends "base.html" %}
2. {% block content %}
3. <div class="layui-container">
4. <div class="layui-row">
5. *<!-- 左侧日期选择栏 -->*
6. <div class="layui-col-md3">
7. <div>
8. *<!-- Layui 日历组件 -->*
9. <div class="layui-inline" id="date-picker"></div>
10. <br>
11. <br>
12. *<!-- 无日期选项 -->*
13. <div>
14. <button class="layui-btn layui-btn-primary" id="no-date">不限日期</button>
15. </div>
16. </div>
17. </div>
18. *<!-- 通知卡片显示区域 -->*
19. <div class="layui-col-md9">
20. <div id="notice-cards"></div>
21. </div>
22. </div>
23. </div>
24. <script>
25. layui.use(['jquery', 'element', 'laytpl', 'laydate'], function () {
26. var $ = layui.jquery;
27. var element = layui.element;
28. var laytpl = layui.laytpl;
29. var laydate = layui.laydate;
30. *// 初始化Layui日历组件*
31. laydate.render({
32. elem: '#date-picker',
33. position: 'static',
34. done: function(value) {
35. console.log(value);
36. fetchNoticesByDate(value);
37. }
38. });
39. *// 无日期按钮点击事件*
40. $('#no-date').on('click', function () {
41. fetchNoticesByDate(null);
42. });
43. function fetchNoticesByDate(date) {
44. var url = date ? 'http://127.0.0.1:5000/Notice/Search?date=' + date : 'http://127.0.0.1:5000/Notice/Search';
45. $.ajax({
46. url: url,
47. method: 'GET',
48. success: function (data) {
49. renderNotices(data.data); *// 假设返回的数据中notices字段包含所有通知*
50. }
51. });
52. }
53. });
54. </script>
55. <style>
56. </style>
57. {% endblock %}



[图12 - “日历”的前端代码]

3 研究结果

小组成员通过深入学习和掌握基于pytorch的模型搭建与调试、flask框架、ajax开发前端等相关知识，成功实现通知内容管理App，支持通知信息分类、信息精简、重点关注、模糊搜索、生成日程安排五项功能。

通知信息分类功能能够准确判别、分类通知，并打上相应的分类标签，使得用户可以更加有序地管理信息流。信息精简功能通过自动筛选和整理，精简提炼通知中重要的信息（如时间、地点）。为了满足用户的个性化需求，小组成员设计并成功实现了重点关注功能。用户可以根据个人需求设置重点关注标签，使得用户能够更为高效地获取相关信息。此外，模糊搜索功能使用户可以通过关键词模糊匹配的方式找到目标通知，从而更快捷地定位所需信息，极大地提高了信息检索的效率。最后，小组成员为App添加了生成日程安排的功能，用户可以根据通知信息生成日程安排，帮助用户更好地规划和安排自己的时间。

在实现App五项功能的基础上，开发过程注重前端设计的简洁美观，打造了一个用户友好且功能丰富的通知内容管理App，为用户提供了一体化、高效率的信息管理解决方案。

4 创新点

通知内容管理App的信息分类、重点关注等功能，集成了飞书App“pin消息”功能（将消息固定置顶位置防止消息被淹没）和分类文档功能，在此基础上创新了信息精简、模糊搜索等功能，极大提升了对通知内有效信息等理解转化效率。

此外，学长学姐开发的《HITA课表》、《HITsz助手》等App仅支持查看每日课程安排等功能，通知内容管理App创新开发了生成日程安排的功能，解决了只能在日历上手动添加截止日期提醒的问题，提供了更为全面的信息管理服务，加强了它在生活中的实用性。

5 结束语

通过本次大一立项的研究与实践，小组成功地实现了通知内容管理App的设计与开发，为解决通知传达方面存在的问题提供了一种全新方案。在完成整个项目的过程中，我们深感信息化时代背景下对于高效通知管理的迫切需求，也认识到通过技术手段可以为用户提供更为便捷、个性化的服务。

作为计算机科学与技术学院和理学院的学生，小组通过深入学习和应用相关知识，成功实现了通知内容管理App的五项核心功能，即通知信息分类、信息精简、重点关注、模糊搜索和生成日程安排，为推动学院通知发布工作更为高效便利贡献了自己的力量。

此外，小组借鉴了飞书App的“pin消息”和分类文档功能，引入信息精简、模糊搜索等创新功能，使得通知内容管理App在信息整理和检索方面表现更为出色。同时，小组也致力于解决其他校内App的局限性，创新性地添加生成日程安排的功能。这些创新结果为项目增色不少，也为今后进一步的优化和拓展提供了方向和思路。

在这一年的时间中，小组不仅学习掌握了技术知识，还培养了团队协作、问题解决和创新思维能力。在未来小组将不断学习和成长，继续关注并参与技术的发展，努力为社会提供更多创新且实用的解决方案。

感谢指导老师吴宇琳老师。她的耐心指导和深厚的学科知识提供了宝贵的学术引导，使小组在整个研究过程中受益匪浅。感谢开题答辩和中期答辩的老师们。老师们的严谨态度和指导性意见为项目的改进提供了有力的支持。之后小组将继续努力完善和优化此通知内容管理App，希望能为广大师生提供更为便捷、高效的通知管理服务！

6 参考文献

[1] https://huggingface.co/IDEA-CCNL/Randeng-Pegasus-523M-Summary-Chinese 文本摘要 （pegasus）

[2] https://huggingface.co/IDEA-CCNL/Erlangshen-Roberta-110M-Sentiment 文本分类（二分类）

[3] https://flask.palletsprojects.com/en/3.0.x/ flask 文档

[4] https://arxiv.org/abs/1912.08777 PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization（pegasus 文本摘要预训练模型）

[5] https://arxiv.org/abs/1907.11692 RoBERTa: A Robustly Optimized BERT Pretraining Approach （预训练的 BERT 模型，用于分类）

[6] https://github.com/IDEA-CCNL/Fengshenbang-LM 封神榜大模型

[7] https://en.wikipedia.org/wiki/Ajax\_(programming) ajax wiki

[8] https://api.jquery.com/category/ajax/ ajax jquery

[9] https://revealjs.com reveal.js 通过html做ppt的node.js库

[10] https://www.sqlalchemy.org sqlalchemy

[11] https://pytorch.org pytorch

[12] https://layui.dev LayUI

[13] https://www.python.org python

[14] https://developer.mozilla.org/en-US/docs/Web/CSS css

[15] https://developer.mozilla.org/en-US/docs/Web/javascript javascript

[16] https://nodejs.org/en/download node.js（npm）

[17] https://www.opensuse.org opensuse（本项目在 opensuse 上训练的）

[18] https://arxiv.org/abs/1706.03762 attention is all you need

[19] https://zh-v2.d2l.ai 动手学深度学习

[20] https://arxiv.org/abs/1910.13461 BART

[21] https://arxiv.org/abs/1910.10683 T5