```
//-------------------------------------------------------界面模块
------------------------------------------------------------
import os
from PySide2.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout,
QPushButton, QLabel, QStackedWidget, \
    QTextEdit, QFileDialog, QFrame, QLineEdit, QScrollArea
from PySide2.QtGui import QPalette, QColor, QFont, QLinearGradient, QBrush, QPixmap
from PySide2.QtCore import Qt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
import matplotlib.pyplot as plt

from predict import predict_defects
from model import load_model

class IntroWindow(QWidget):
    """
    主界面类，用于展示软件的主界面。
    包括背景设置、标题显示、以及三个功能按钮：操作指南、详情介绍和开始体验。
    """

    def __init__(self):
        """
        初始化 IntroWindow 类，设置窗口的基础属性和 UI 界面。
        """
        super().__init__()
        self.init_ui()

    def init_ui(self):
        """
        设置主界面的用户界面，包括背景颜色、窗口属性、标题和功能按钮。
        """
        # 设置背景渐变颜色
        palette = QPalette()
        # 设置渐变颜色，从白色到浅蓝色
        gradient = QLinearGradient(0, 0, 1500, 1000)
        gradient.setColorAt(0.0, QColor(255, 255, 255))  # 渐变起点为白色
        gradient.setColorAt(1.0, QColor(173, 216, 230))  # 渐变终点为浅蓝色
        # 将渐变应用到背景
        palette.setBrush(QPalette.Window, QBrush(gradient))
        self.setPalette(palette)

        # 设置窗口标题和尺寸
        self.setWindowTitle('智能软件缺陷预测与排序系统')
        self.resize(1500, 1000)
        self.setFixedSize(1500, 1000)

        # 添加标题标签并居中
        title_label = QLabel('智能软件缺陷预测与排序系统', self)
        title_label.setAlignment(Qt.AlignCenter)
        # 设置标题的字体为楷体并加粗，字号为 48
```

```
51          font = QFont("KaiTi", 48)
52          title_label.setFont(font)
53          title_label.setStyleSheet("font-weight: bold;")
54          # 设置标题标签的大小和位置
55          title_label.setFixedSize(1500, 100)
56          title_label.move(0, 200)
57
58          # 创建操作指南按钮
59          guide_button = QPushButton('操作指南', self)
60          # 设置按钮字体为楷体，字号为 20
61          font = QFont("KaiTi", 20)
62          guide_button.setFont(font)
63          # 设置按钮的大小
64          guide_button.setFixedSize(300, 100)
65          # 连接按钮点击事件，点击后显示操作指南界面
66          guide_button.clicked.connect(self.show_guide)
67          # 设置按钮的位置
68          guide_button.move(200, 700)
69
70          # 创建详情介绍按钮
71          detail_button = QPushButton('详情介绍', self)
72          # 设置详情按钮字体及大小
73          font = QFont("KaiTi", 20)
74          detail_button.setFont(font)
75          detail_button.setFixedSize(300, 100)
76          # 连接按钮点击事件，点击后显示详情介绍界面
77          detail_button.clicked.connect(self.show_details)
78          # 设置按钮的位置
79          detail_button.move(600, 700)
80
81          # 创建开始体验按钮
82          predict_button = QPushButton('开始体验', self)
83          # 设置按钮字体为楷体，字号为 20
84          font = QFont("KaiTi", 20)
85          predict_button.setFont(font)
86          # 设置按钮大小
87          predict_button.setFixedSize(300, 100)
88          # 连接按钮点击事件，点击后显示登录体验界面
89          predict_button.clicked.connect(self.show_login)
90          # 设置按钮位置
91          predict_button.move(1000, 700)
92
93      def show_guide(self):
94          """
95          打开操作指南界面。
96          创建一个 GuideWindow 类对象并显示
97          """
98          self.guide_window = GuideWindow()
99          self.guide_window.show()
100
```

```
101     def show_details(self):
102         """
103         打开详情介绍界面。
104         创建一个 DetailsWindow 类对象并显示
105         """
106         self.details_window = DetailsWindow()
107         self.details_window.show()
108 
109     def show_login(self):
110         """
111         打开登录体验界面。
112         创建一个 LoginWindow 类对象并显示
113         """
114         self.login_window = LoginWindow()
115         self.login_window.show()
116 
117 
118 class GuideWindow(QWidget):
119     """
120     使用手册界面类，用于展示用户操作指南和系统的使用说明。
121     包括窗口背景设置、标题显示、以及详细的操作步骤和注意事项。
122     """
123 
124     def __init__(self):
125         """
126         初始化 GuideWindow 类，设置窗口的基本属性和界面布局。
127         """
128         super().__init__()
129 
130         # 设置窗口标题
131         self.setWindowTitle("智能软件缺陷预测与排序系统")
132         # 设置窗口大小
133         self.resize(1500, 1000)
134         # 设置窗口背景为蓝白渐变
135         palette = QPalette()
136 
137         # 创建渐变对象，从浅蓝色到白色，垂直渐变
138         gradient = QLinearGradient(0, 0, 0, self.height())
139         gradient.setColorAt(0.0, QColor(173, 216, 230))  # 起始颜色：浅蓝色
140         gradient.setColorAt(1.0, QColor(255, 255, 255))  # 终止颜色：白色
141         # 将渐变应用到窗口背景
142         palette.setBrush(QPalette.Window, QBrush(gradient))
143         self.setPalette(palette)
144 
145         # 创建主布局，采用垂直布局
146         main_layout = QVBoxLayout()
147 
148         # 在顶部和标题之间添加空白区域
149         main_layout.addStretch(1)
150         # 创建标题标签
```

```
151         title_label = QLabel("智能软件缺陷预测与排序系统  使用手册")
152         # 设置标题的字体为楷体，字号为 24，加粗
153         title_label.setFont(QFont("KaiTi", 24, QFont.Bold))
154         # 设置标题居中对齐
155         title_label.setAlignment(Qt.AlignCenter)
156         # 将标题添加到主布局中
157         main_layout.addWidget(title_label)
158
159         # 在标题和文本内容之间添加空白区域
160         main_layout.addStretch(1)
161         # 创建内容标签，显示操作步骤
162         content_label = QLabel(
163             "1.登录界面操作"
164             "\n    步骤 1: 在登录界面输入您的手机号。"
165             "\n    步骤 2: 输入对应的密码。"
166             "\n    步骤 3: 点击“登录 / 注册”按钮以进入系统。"
167             "\n2.上传代码文件"
168             "\n    步骤 1: 成功登录后，您将进入预测界面。"
169             "\n    步骤 2: 在预测界面上，点击“上传代码”按钮。"
170             "\n    步骤3: 选择并上传您需要检测的代码文件。支持的文件格式包括C, C
171 + +, Java, Python 等。"
172             "\n3.执行代码漏洞检测"
173             "\n    步骤 1: 在代码文件上传完成后，点击“预测”按钮。"
174             "\n    步骤 2: 等待几秒钟，系统将自动分析每个文件的代码缺陷。"
175             "\n    步骤 3: 分析完成后，右侧的输出栏中将显示每个文件预测的缺陷个
176 数。"
177             "\n    步骤 4: 同时，系统会生成一个柱状图，直观展示各文件的缺陷数量
178 排序。"
179             "\n注意事项"
180             "\n    确保在上传代码前已正确登录系统。"
181             "\n    上传的代码文件应为有效的编程语言文件（如 C, C + +, Java, Python
182 等）。"
183             "\n    请耐心等待预测过程完成，避免中途关闭或刷新页面。"
184         )
185         # 设置内容标签的字体为楷体，字号为 16
186         content_label.setFont(QFont("KaiTi", 16))
187         # 允许内容自动换行
188         content_label.setWordWrap(True)
189         # 设置文本顶部对齐
190         content_label.setAlignment(Qt.AlignTop)
191         # 为内容添加边框和内陷效果
192         content_label.setFrameShape(QFrame.Panel)
193         content_label.setFrameShadow(QFrame.Sunken)
194         content_label.setLineWidth(2)  # 设置边框线宽
195         content_label.setMargin(10)  # 设置内容的边距
196         # 创建内容布局并添加内容标签
197         content_layout = QHBoxLayout()
198         content_layout.addWidget(content_label)
199         # 设置内容的外边距
200         content_layout.setContentsMargins(20, 20, 20, 20)
```

```
        # 将内容布局添加到主布局
        main_layout.addLayout(content_layout)
        # 在内容和窗口底部添加空白区域
        main_layout.addStretch(1)

        # 将主布局设置为窗口的布局
        self.setLayout(main_layout)


class DetailsWindow(QWidget):
    """
    详情介绍界面类，用于展示详细的项目介绍和说明。
    包括窗口背景设置、标题、以及通过滚动区域显示的详细文本内容。
    """

    def __init__(self):
        """
        初始化 DetailsWindow 类，设置窗口的基本属性、背景、标题和详细内容显示区域。
        """
        super().__init__()

        # 设置窗口标题
        self.setWindowTitle("智能软件缺陷预测与排序系统")
        # 设置窗口大小
        self.resize(1500, 1000)
        # 设置窗口背景为蓝白渐变
        palette = QPalette()
        # 创建渐变对象，从浅蓝色到白色，垂直渐变
        gradient = QLinearGradient(0, 0, 0, self.height())
        gradient.setColorAt(0.0, QColor(173, 216, 230))  # 起始颜色：浅蓝色
        gradient.setColorAt(1.0, QColor(255, 255, 255))  # 终止颜色：白色
        # 将渐变应用到窗口背景
        palette.setBrush(QPalette.Window, QBrush(gradient))
        self.setPalette(palette)
        # 设置窗口为固定大小
        self.setFixedSize(1500, 1000)

        # 添加标题标签，并通过绝对布局设置位置和大小
        self.title_label = QLabel('智能软件缺陷预测与排序系统：基于深度学习的软件缺陷分析师', self)
        # 设置标题的字体为楷体，字号为 24，加粗
        self.title_label.setFont(QFont("KaiTi", 24, QFont.Bold))
        # 设置标题居中对齐
        self.title_label.setAlignment(Qt.AlignCenter)
        # 设置标题的位置和大小
        self.title_label.setGeometry(50, 100, 1400, 50)

        # 创建滚动区域，用于显示超出可视范围的内容
        self.scroll_area = QScrollArea(self)
```

```
        # 设置滚动区域的位置和大小
        self.scroll_area.setGeometry(100, 200, 1300, 700)
        # 允许滚动区域内的内容大小根据窗口自动调整
        self.scroll_area.setWidgetResizable(True)
        # 创建内容显示的 QWidget，用于容纳所有文本
        content_widget = QWidget()
        # 设置内容区域的大小
        content_widget.setFixedSize(1280, 1500)
        # 创建 QTextEdit，用于显示项目的详细内容
        content_text = QTextEdit(content_widget)
        # 设置文本框为只读模式，避免用户修改内容
        content_text.setReadOnly(True)
        # 设置文本框的大小和位置
        content_text.setGeometry(0, 0, 1280, 1500)
        # 设置文本框字体为楷体，字号为 16
        content_text.setFont(QFont("KaiTi", 16))
        # 设置文本框的边框样式和内陷效果
        content_text.setFrameShape(QFrame.Panel)
        content_text.setFrameShadow(QFrame.Sunken)
        # 设置边框线宽
        content_text.setLineWidth(2)
        # 添加显示的内容文本
        content_text.setText(
            "    软件缺陷是指在软件开发过程中或软件已经投入使用后发现的，与预期功能或性"
            "能不符的问题或错误，并且导致软件不能按照预期的方式运行或不能满足用户需求。"
            "已有的软件缺陷预测算法无法充分提取代码的语义信息，并且排序过程中并未考虑到"
            "代价敏感问题。因此，本项目旨在采用预训练模型及基于代价敏感学习的缺陷排序算"
            "法来解决上述问题，以对软件模块进行合理的排序，从而提升软件质量保障的效率和效果。"
            "\n    针对已有方法在对软件模块进行缺陷预测时代码语义理解不充分和未考虑对存在"
            "缺陷的软件模块测试优先级的困境，尝试以代码预训练模型，对代码的多语义特征进行"
            "提取。并基于混合注意力机制的特征学习方法，同时利用单头注意力编码器和多头注意"
            "力编码器对代码 token 进行编码，提高软件缺陷预测模型的性能。且基于代价敏感学习"
            "的学习排序方法，修改损失函数，让软件模块按照缺陷个数，缺陷严重程度更大或者缺"
            "陷密度更高进行正确排序，有效解决当前现有软件缺陷预测方法技术准确率低以及未考"
            "虑对软件模块的测试优先级的问题。"
            "\n    本项目是在现有研究成果的基础上，考虑到软件缺陷会对用户造成巨额经济损失的这一痛点问题，立志在理论、技术和方法上创新，提出对软件缺陷排序方法的研究，其"
            "特色和创新之处在于：本项目提出了基于 CodeBERT 的软件代码多语义提取
```

```
方法、基"
                "于混合注意力机制的特征学习方法、基于代价敏感学习的学习排序方法。提
出的基于"
                "CodeBERT 的软件代码多语义提取方法，首先将软件代码表示为抽象语法树，
然后利用"
                "CodeBERT 从抽象语法树中提取代码词汇、语法和结构信息，实现了对软件代
码多语义"
                "信息的提取，为后续缺陷预测模型的建立提供了缺陷代码信息基础。"
                "\n    本项目提出了一种"
                "基于混合注意力机制的特征学习方法，为了对代码 token 进行编码，团队结
合了基于"
                "Bi-GRU 的单头注意力编码器和基于 Transformer 的多头注意力编码器。后
续通过捕获"
                "代码 token 之间的依赖关系，提高了软件缺陷预测模型的性能，使得并能通
过混合注意"
                "力机制对代码 token 赋予的不同权重解释预测结果。"
                "\n    通过代价敏感学习算法，可以计算"
                "错误排序的代价并相应地修改学习排序算法的损失函数。这样能够更好地排
序缺陷严重"
                "程度更大、缺陷个数更多或缺陷密度更高的软件模块，从而减少相关问题的
代价，有效"
                "地提升了软件测试的效率和质量。"
                "\n    这些 CodeBERT、注意力机制、代价敏感学习、学习"
                "排序技术的深度运用，既体现了大规模软件代码环境下缺陷预测研究的独特
技术特征，"
                "也为今后这些技术在软件理论技术研究中的常态化运用作出了示范。"
        )

        # 将内容 widget 设置为滚动区域的 widget
        self.scroll_area.setWidget(content_widget)


class PredictWindow(QWidget):
    """
    预测窗口类，用于用户上传代码文件、执行缺陷预测并显示结果。
    包括文件上传、预测按钮、显示输入代码和预测结果的文本框，以及缺陷数量的柱状图。
    """

    def __init__(self):
        """
        初始化 PredictWindow 类，设置主界面的布局和组件。
        """
        super().__init__()
        self.init_ui()

    def init_ui(self):
        """
        初始化主界面的布局和用户界面，包括文件上传按钮、预测按钮、文本框和柱状图
显示区域。
        """
```

```
351         # 设置主布局为水平布局，左侧为文件操作，右侧为结果显示
352         main_layout = QHBoxLayout()
353         # 左侧布局，包含文件上传和输入框
354         left_layout = QVBoxLayout()
355
356         # 创建上传按钮，用于上传代码文件
357         self.upload_button = QPushButton('上传代码文件')
358         self.upload_button.setFixedHeight(50)  # 设置按钮高度
359         self.upload_button.clicked.connect(self.upload_files)  # 连接文件上传事
360 件
361         left_layout.addWidget(self.upload_button)
362
363         # 创建文本框，用于显示上传的文件内容
364         self.input_box = QTextEdit()
365         self.input_box.setReadOnly(True)  # 设置为只读模式
366         left_layout.addWidget(self.input_box)
367
368         # 创建预测按钮，用于执行代码缺陷预测
369         self.predict_button = QPushButton('预测')
370         self.predict_button.setFixedHeight(50)  # 设置按钮高度
371         self.predict_button.clicked.connect(self.on_predict)  # 连接预测事件
372         left_layout.addWidget(self.predict_button)
373
374         # 右侧布局，包含结果文本框和柱状图显示区域
375         right_layout = QVBoxLayout()
376
377         # 创建文本框，用于显示预测结果
378         self.output_box = QTextEdit()
379         self.output_box.setReadOnly(True)  # 设置为只读模式
380         right_layout.addWidget(self.output_box)
381
382         # 创建柱状图区域，用于显示缺陷预测结果
383         self.figure = plt.figure()
384         self.canvas = FigureCanvas(self.figure)
385         right_layout.addWidget(self.canvas)
386
387         # 将左侧和右侧布局添加到主布局中
388         main_layout.addLayout(left_layout)
389         main_layout.addLayout(right_layout)
390
391         # 设置左右布局的比例
392         main_layout.setStretchFactor(left_layout, 3)  # 左侧占 60%
393         main_layout.setStretchFactor(right_layout, 2)  # 右侧占 40%
394
395         # 设置窗口主布局
396         self.setLayout(main_layout)
397         self.setWindowTitle('智能软件缺陷预测与排序系统')
398         # 设置窗口大小
399         self.resize(1800, 1200)
400
```

```
401         # 初始化文件内容的列表，用于存储上传的文件内容
402         self.file_contents = []
403 
404         # 加载缺陷预测模型
405         self.model = load_model("linear_regression_model.pkl")
406 
407     def upload_files(self):
408         """
409         文件上传功能，允许用户选择多个代码文件并展示其内容。
410         """
411         # 打开文件对话框，选择代码文件
412         file_paths, _ = QFileDialog.getOpenFileNames(self, "选择代码文件", "",
413                                                      "代码文件 (*.cpp *.java
414 *.py *.txt);;所有文件 (*)")
415 
416         if file_paths:
417             # 清空之前的文件内容
418             self.file_contents.clear()
419             combined_text = ""
420 
421             # 逐个读取文件内容并展示
422             for file_path in file_paths:
423                 try:
424                     # 打开文件并读取内容，使用 UTF-8 编码
425                     with open(file_path, 'r', encoding='utf-8') as file:
426                         content = file.read()
427 
428                     # 获取文件名并展示文件内容
429                     file_name = os.path.basename(file_path)
430                     combined_text += f"文件: {file_name}\n" + content + "\n" + "-"
431 * 40 + "\n\n"
432 
433                     # 保存文件名和内容到文件内容列表中
434                     self.file_contents.append((file_name, content))
435 
436                 except Exception as e:
437                     # 如果文件读取失败，输出错误信息
438                     print(f"文件读取错误: {e}")
439 
440             # 将读取到的文件内容显示在输入框中
441             self.input_box.setText(combined_text)
442 
443     def on_predict(self):
444         """
445         预测功能，使用加载的模型对上传的代码文件进行缺陷预测，并显示结果。
446         """
447         # 如果没有上传文件，则显示提示
448         if not self.file_contents:
449             self.output_box.setText("No files uploaded or content is empty")
450             return
```

```
451
452         # 提取文件名和文件内容
453         file_names = [name for name, _ in self.file_contents]
454         code_snippets = [content for _, content in self.file_contents]
455
456         # 使用模型进行缺陷预测
457         predicted_defects = predict_defects(self.model, code_snippets)
458
459         # 生成文件编号（a, b, c, d...）
460         file_identifiers = [chr(97 + i) for i in range(len(file_names))]
461         # 生成文件名和预测结果的组合
462         labeled_file_names = [f"{file_names[i]}" for i in range(len(file_names))]
463         results      =      list(zip(file_identifiers,      labeled_file_names,
464 predicted_defects))
465         # 按照预测的缺陷数量降序排列
466         sorted_results = sorted(results, key=lambda x: x[2], reverse=True)
467         # 构建结果文本
468         output_text = (
469             "Below is the number of defects predicted for each input file using the
470 linear regression model, "
471             "sorted in descending order of predicted defect counts:\n\n"
472         )
473
474         # 输出排序后的预测结果
475         for identifier, file_name, defects in sorted_results:
476             output_text  +=  f"{identifier}:  {file_name},  Predicted  defects:
477 {defects}\n"
478         # 将结果显示在输出框中
479         self.output_box.setText(output_text)
480         # 绘制柱状图显示预测结果
481         self.plot_defects_bar_chart([x[0] for x in sorted_results], [x[2] for x in
482 sorted_results])
483
484     def plot_defects_bar_chart(self, file_ids, defects):
485         """
486         绘制柱状图，展示预测的缺陷数量。
487         """
488         # 清除旧的图表内容
489         self.figure.clear()
490
491         # 创建新的柱状图
492         ax = self.figure.add_subplot(111)
493         ax.bar(file_ids, defects)
494         # 设置图表标题和坐标轴标签
495         ax.set_title("Predicted Defects by File ID")
496         ax.set_xlabel("File ID")
497         ax.set_ylabel("Number of Defects")
498         # 刷新画布，显示新图表
499         self.canvas.draw()
500
```

```
501
502 class LoginWindow(QWidget):
503     """
504     登录窗口类，提供用户登录功能。
505     包括手机号输入框、密码输入框以及登录/注册按钮。
506     """
507
508     def __init__(self):
509         """
510         初始化 LoginWindow 类，设置窗口基本属性和用户界面布局。
511         """
512         super().__init__()
513
514         # 设置窗口标题
515         self.setWindowTitle("智能软件缺陷预测与排序系统")
516
517         # 设置窗口的固定大小为 1200x800
518         self.setFixedSize(1200, 800)
519
520         # 设置背景为渐变颜色，起点为白色，终点为浅蓝色
521         palette = QPalette()
522         gradient = QLinearGradient(0, 0, 1200, 800)
523         # 起点颜色：白色
524         gradient.setColorAt(0.0, QColor(255, 255, 255))
525         # 终点颜色：浅蓝色
526         gradient.setColorAt(1.0, QColor(173, 216, 230))
527         palette.setBrush(QPalette.Window, QBrush(gradient))
528         self.setPalette(palette)
529
530         # 使用绝对布局，通过 move() 方法设置部件的位置
531
532         # 添加左上角的 Logo
533         logo_label = QLabel(self)
534
535         # 加载并缩放 Logo 图片
536         pixmap = QPixmap("logo.png")
537         # 按比例缩放图片
538         scaled_pixmap = pixmap.scaled(450, 450, Qt.KeepAspectRatio)
539         logo_label.setPixmap(scaled_pixmap)
540         # 设置 Logo 的位置
541         logo_label.move(100, 100)
542         # 创建手机号标签，并设置字体
543         phone_label = QLabel("手机号", self)
544         # 楷体字体，字号为 20
545         phone_label.setFont(QFont("KaiTi", 20, QFont.Bold))
546         # 设置手机号标签的位置
547         phone_label.move(600, 300)
548
549         # 创建手机号输入框，并设置字体和大小
550         self.phone_input = QLineEdit(self)
```

```
551             # 楷体字体，字号为 17
552             font = QFont("KaiTi", 17)
553             self.phone_input.setFont(font)
554             # 设置输入框高度
555             self.phone_input.setFixedHeight(50)
556             # 设置输入框宽度
557             self.phone_input.setFixedWidth(350)
558             # 设置手机号输入框的位置
559             self.phone_input.move(750, 300)
560
561             # 创建密码标签，并设置字体
562             password_label = QLabel("密码", self)
563             # 楷体字体，字号为 20
564             password_label.setFont(QFont("KaiTi", 20, QFont.Bold))
565             # 设置密码标签的位置
566             password_label.move(600, 400)
567
568             # 创建密码输入框，并设置字体和大小
569             self.password_input = QLineEdit(self)
570             # 楷体字体，字号为 16
571             font = QFont("KaiTi", 16)
572             self.password_input.setFont(font)
573             # 设置输入框高度
574             self.password_input.setFixedHeight(50)
575             # 设置输入框宽度
576             self.password_input.setFixedWidth(350)
577             # 设置密码输入框为密码模式，隐藏输入字符
578             self.password_input.setEchoMode(QLineEdit.Password)
579             # 设置密码输入框的位置
580             self.password_input.move(750, 400)
581
582             # 创建登录/注册按钮，并设置样式
583             login_button = QPushButton("登录/注册", self)
584             # 楷体字体，字号为 20
585             login_button.setFont(QFont("KaiTi", 20, QFont.Bold))
586             # 设置按钮的固定大小
587             login_button.setFixedSize(250, 50)
588             # 设置按钮的背景颜色、文本颜色和圆角效果
589             login_button.setStyleSheet("background-color:  #1E90FF;  color:  white;
590 border-radius: 10px;")
591             # 设置按钮的位置
592             login_button.move(600, 500)
593             # 连接按钮的点击事件，点击按钮后显示预测界面
594             login_button.clicked.connect(self.show_predict)
595
596         def show_predict(self):
597             """
598             打开预测界面。
599             """
600             self.predict_window = PredictWindow()
```

```
        self.predict_window.show()


if __name__ == '__main__':
    app = QApplication([])
    intro_window = IntroWindow()
    intro_window.show()
    app.exec_()

//-----------------------------------------------------训练模块
-------------------------------------------------------------
import pandas as pd
from transformers import AutoTokenizer, AutoModel
import torch
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from model import save_model
import numpy as np


def read_data(file_path):
    """
    读取 CSV 文件中的数据，将代码片段和对应的缺陷数量作为输出返回。

    参数:
    file_path (str): 文件路径

    返回:
    tuple: 包含代码片段的列表和缺陷数量的列表
    """
    data = pd.read_csv(file_path)
    code_snippets = data['code'].tolist()
    defect_counts = data['defects'].tolist()
    return code_snippets, defect_counts


def extract_features(code_snippets):
    """
    使用 CodeBERT 模型提取代码片段的特征。每个代码片段被转化为嵌入向量。

    参数:
    code_snippets (list): 包含代码片段的列表

    返回:
    list: 包含每个代码片段的特征向量
    """
    tokenizer = AutoTokenizer.from_pretrained("microsoft/codebert-base")
    model = AutoModel.from_pretrained("microsoft/codebert-base")
    features = []

```

```
    # 遍历每个代码片段，提取其特征向量
    for code in code_snippets:
        inputs = tokenizer(code, return_tensors="pt", padding=True,
truncation=True)
        with torch.no_grad():
            outputs = model(**inputs)

        # 获取最后一层隐藏状态的均值作为特征

features.append(outputs.last_hidden_state.mean(dim=1).squeeze().numpy())

    return features


def train_model(features, defect_counts):
    """
    训练线性回归模型，根据代码特征预测缺陷数量。使用对数变换处理目标变量，确保模
型输出正值。

    参数:
    features (list): 包含代码特征的列表
    defect_counts (list): 包含代码缺陷数量的列表

    返回:
    tuple: 训练后的模型，测试集特征，测试集真实值，测试集预测值
    """
    # 对缺陷数量进行对数变换，避免对数计算中的零问题
    log_defects = np.log(np.array(defect_counts) + 1)

    # 将数据分割为训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(features, log_defects,
test_size=0.2, random_state=42)

    # 初始化线性回归模型并进行训练
    model = LinearRegression()
    model.fit(X_train, y_train)

    # 进行预测，并对预测值进行反变换
    y_pred_log = model.predict(X_test)
    y_pred = np.exp(y_pred_log) - 1

    return model, X_test, y_test, y_pred


def evaluate_model(y_test, y_pred):
    """
    评估模型性能，计算测试集的均方误差 (MSE)。

    参数:
    y_test (list): 测试集的真实值
```

```
    y_pred (list): 测试集的预测值

    返回:
    float: 均方误差 (MSE)
    """
    from sklearn.metrics import mean_squared_error

    # 计算 MSE，首先对测试集的真实值进行反变换
    mse = mean_squared_error(np.exp(y_test) - 1, y_pred)
    print(f"Mean Squared Error: {mse}")
    return mse


def main():
    """
    主函数，负责读取数据、提取特征、训练模型、评估模型并保存模型。
    """
    # 读取数据
    file_path = "../File/Code.csv"
    code_snippets, defect_counts = read_data(file_path)

    # 提取代码特征
    features = extract_features(code_snippets)

    # 训练模型并返回测试集和预测结果
    model, X_test, y_test, y_pred = train_model(features, defect_counts)

    # 评估模型性能
    evaluate_model(y_test, y_pred)

    # 保存训练好的模型
    model_file_path = "linear_regression_model.pkl"
    save_model(model, model_file_path)
    print(f"Model saved to {model_file_path}")


if __name__ == "__main__":
    main()

//-------------------------------------------------------模型模块
------------------------------------------------------------
# model.py
import joblib


def save_model(model, file_path):
    """
    将训练好的模型保存到文件。

    参数:
```

```
    - model: 训练好的模型
    - file_path: 模型保存的文件路径
    """
    joblib.dump(model, file_path)


def load_model(file_path):
    """
    从文件加载已保存的模型。

    参数:
    - file_path: 模型保存的文件路径

    返回:
    - model: 加载的模型
    """
    return joblib.load(file_path)

//-----------------------------------------------------------创建虚拟环境模块
-----------------------------------------------------------------
chcp 65001
@echo off

set start_time=%time%
echo =: 开始时间: %start_time%

echo =: 正在创建虚拟环境...
env\python -m venv workenv

echo =: 正在进入虚拟环境...
call workenv\Scripts\activate.bat
python.exe -m pip install --upgrade pip

echo =: 正在安装依赖...
REM 定义 requirements.txt 文件路径
set REQUIREMENTS=requirements.txt

REM 检查 requirements.txt 文件是否存在
if not exist %REQUIREMENTS% (
    echo =: requirements.txt 文件不存在
    exit /b 1
)

REM 读取 requirements.txt 文件中的所有库名称
for /f "tokens=1,* delims==" %%i in (%REQUIREMENTS%) do (
    REM 检查当前库是否已安装
    pip show %%i >nul 2>&1
    if errorlevel 1 (
        REM 如果未安装，则使用 pip 安装该库
        echo =: 安装 %%i... %time%
```

```
        pip install %%i
    ) else (
        echo =: %%i 已安装
    )
)

echo =: 所有依赖都已安装

set end_time=%time%
echo =: 结束时间: %end_time%

python -c "from datetime import datetime as dt; start_time = dt.strptime('%start_time%', '%%H:%%M:%%S.%%f'); end_time = dt.strptime('%end_time%', '%%H:%%M:%%S.%%f'); time_diff = end_time - start_time; print('环境配置完成，耗时:', time_diff)"
Deactivate

//-------------------------------------------------------运行模块
------------------------------------------------------------
chcp 65001
@echo off

echo 注意：程序运行过程中切勿关闭本窗口，否则会导致程序中断运行！！！
echo 如果您是第一次启动本程序，可能您需要等待一会……

REM 定义 requirements.txt 文件路径
set WORKENV=workenv

REM 检查环境是否初始化
if not exist %WORKENV% (
    echo =: 环境未初始化，正在初始化环境，可能需要 1-2 分钟，请耐心等待...
    call init_workenv.bat
)
.\workenv\Scripts\python.exe newMain.py

pause
```