

```
import sys
import os
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import rcParams

matplotlib.use('Qt5Agg')
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
import scipy.signal as signal
import scipy.fftpack as fftpack
from scipy.optimize import curve_fit, least_squares
from PyQt5.QtWidgets import (QApplication, QMainWindow, QTabWidget, QWidget,
                             QVBoxLayout,
                             QHBoxLayout, QPushButton, QFileDialog, QLabel,
                             QComboBox,
                             QSpinBox, QDoubleSpinBox, QGroupBox, QFormLayout,
                             QTextEdit,
                             QTableWidget, QTableWidgetItem, QSplitter, QMessageBox,
                             QCheckBox, QTabBar, QFrame, QProgressDialog)
from PyQt5.QtGui import QFont, QIcon, QPixmap, QColor, QPalette
from PyQt5.QtCore import Qt, QThread, pyqtSignal, QUrl
import matplotlib.font_manager as fm
from matplotlib.ticker import MaxNLocator
import seaborn as sns
import datetime
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas
from docx import Document
from docx.shared import Inches

# 设置中文字体和特殊字符支持
# rcParams["font.family"] = ["KaiTi", "SimHei", "WenQuanYi Micro Hei", "Heiti TC", "DejaVu Sans"]
# rcParams["mathtext.fontset"] = "custom"
# rcParams["mathtext.rm"] = "KaiTi"
# rcParams["mathtext.it"] = "KaiTi:italic"
# rcParams["mathtext.bf"] = "KaiTi:bold"
# matplotlib.rcParams["axes.unicode_minus"] = False # 解决负号显示问题

# 直接使用系统中确定存在的微软雅黑          -----> 解决找不到字体 使用全局的雅黑
plt.rcParams["font.family"] = ["Microsoft YaHei"]
plt.rcParams["axes.unicode_minus"] = False
plt.rcParams["figure.dpi"] = 100

class MplCanvas(FigureCanvas):
```

```
""" matplotlib 画布类 """
def __init__(self, parent=None, width=5, height=4, dpi=100):
    self.fig = Figure(figsize=(width, height), dpi=dpi)
    self.axes = self.fig.add_subplot(111)
    super(MplCanvas, self).__init__(self.fig)
    self.fig.tight_layout()
    self.setParent(parent)
    self.axes.grid(True, linestyle='--', alpha=0.7)

def clear_plot(self):
    """ 清除绘图 """
    self.axes.clear()
    self.axes.grid(True, linestyle='--', alpha=0.7)
    self.draw()

class DataProcessor:
    """ 数据处理工具类 """

    @staticmethod
    def import_data(file_path):
        """ 导入数据 """
        try:
            if file_path.endswith('.xlsx') or file_path.endswith('.xls'):
                df = pd.read_excel(file_path)
            elif file_path.endswith('.csv'):
                df = pd.read_csv(file_path)
            else:
                return None, "不支持的文件格式"

            # 检查数据格式
            if len(df.columns) < 2:
                return None, "数据格式不正确, 至少需要两列数据"

            # 尝试转换为数值类型
            try:
                wavenumber = pd.to_numeric(df.iloc[:, 0])
                reflectance = pd.to_numeric(df.iloc[:, 1])
            except ValueError:
                return None, "数据包含非数值内容, 无法转换"

            # 检查是否有 NaN 值
            if wavenumber.isna().any() or reflectance.isna().any():
                print("数据中包含缺失值, 将进行修复")

            # 创建数据框
            return df, "成功读取数据"
        except Exception as e:
            return None, str(e)
```

```
data = pd.DataFrame({  
    'wavenumber': wavenumber,  
    'reflectance': reflectance  
})  
  
# 去除 NaN 值  
data = data.dropna()  
  
return data, "数据导入成功"  
  
except Exception as e:  
    return None, f"导入失败: {str(e)}"  
  
@staticmethod  
def detect_outliers(data, column='reflectance', threshold=3):  
    """ 异常值检测 """  
    mean = np.mean(data[column])  
    std = np.std(data[column])  
    z_scores = np.abs((data[column] - mean) / std)  
    outliers = z_scores > threshold  
    return outliers  
  
@staticmethod  
def remove_outliers(data, column='reflectance', threshold=3):  
    """ 移除异常值 """  
    outliers = DataProcessor.detect_outliers(data, column, threshold)  
    cleaned_data = data[~outliers].copy()  
    return cleaned_data, sum(outliers)  
  
@staticmethod  
def smooth_data(data, window_size=5):  
    """ 数据平滑处理 """  
    if window_size % 2 == 0:  
        window_size += 1 # 确保窗口大小为奇数  
  
    smoothed = signal.savgol_filter(data['reflectance'], window_size, 3)  
    result = data.copy()  
    result['smoothed_reflectance'] = smoothed  
    return result  
  
@staticmethod  
def repair_missing(data):  
    """ 修复缺失值 """  
    # 首先确保数据按波数排序  
    data_sorted = data.sort_values('wavenumber').copy()  
  
    # 使用线性插值填充可能的缺失值
```

```

data_sorted['reflectance'] = data_sorted['reflectance'].interpolate(method='linear')

return data_sorted


class RefractiveIndexModel:
    """ 折射率模型类 """

    @staticmethod
    def calculate_refractive_index(wavenumber, material_type, carrier_concentration=1e16):
        """ 计算折射率 """
        # 转换波数(cm^-1)到波长(μm)
        wavelength = 10000 / wavenumber # 1 cm = 10000 μm

        # 不同材料的折射率模型参数
        if material_type == "4H-SiC":
            # 4H-SiC 的折射率模型
            n0 = 2.65
            n1 = 0.01
            n2 = 0.001
            n = n0 + n1 * wavelength - n2 * wavelength ** 2
        elif material_type == "6H-SiC":
            # 6H-SiC 的折射率模型
            n0 = 2.68
            n1 = 0.008
            n2 = 0.0009
            n = n0 + n1 * wavelength - n2 * wavelength ** 2
        else:
            # 默认模型
            n0 = 2.66
            n1 = 0.009
            n2 = 0.00095
            n = n0 + n1 * wavelength - n2 * wavelength ** 2

        # 载流子浓度对折射率的影响 (简化模型)
        plasma_freq = np.sqrt(carrier_concentration * 1.6e-19 ** 2 / (9.1e-31 * 8.85e-12))
        light_speed = 3e8 # m/s
        wavelength_m = wavelength * 1e-6 # 转换为米
        plasma_wavelength = 2 * np.pi * light_speed / plasma_freq

        # 载流子引起的折射率变化
        delta_n = - (plasma_wavelength ** 2) / (4 * np.pi ** 2 * wavelength_m ** 2)

        return n + delta_n

    @staticmethod
    def simulate_interference_spectrum(wavenumber, thickness, n, angle_of_incidence):

```

```

""" 模拟干涉光谱 """
# 转换角度到弧度
theta = np.radians(angle_of_incidence)

# 转换波数到波长(μm)
wavelength = 10000 / wavenumber # 单位: μm

# 计算光程差
optical_path_difference = 2 * n * thickness * np.cos(theta)

# 计算相位差
phase_difference = (4 * np.pi / wavelength) * optical_path_difference

# 计算反射率 (简化模型)
r0 = 0.3 # 基础反射率
reflectance = r0 * (1 + np.cos(phase_difference))

# 添加一些噪声使模拟更真实
noise = np.random.normal(0, 0.02, len(wavenumber))
reflectance += noise

# 确保反射率在 0-1 之间
reflectance = np.clip(reflectance, 0, 1)

# 转换为百分比
return reflectance * 100

class ThicknessCalculator:
    """ 厚度计算类 """

    @staticmethod
    def calculate_using_fft(data, n, angle):
        """ 使用 FFT 方法计算厚度 """
        # 确保数据排序
        data_sorted = data.sort_values('wavenumber').copy()

        # 提取波数和反射率
        wavenumber = data_sorted['wavenumber'].values
        reflectance = data_sorted['smoothed_reflectance'].values if 'smoothed_reflectance' in data_sorted.columns else \
            data_sorted['reflectance'].values

        # 计算 FFT
        fft_result = fftpack.fft(reflectance)
        fft_freq = fftpack.fftfreq(len(reflectance), d=(wavenumber[1] - wavenumber[0]) if len(wavenumber) > 1 else 1)

```

```

# 取绝对值并找到主要频率成分
fft_amplitude = np.abs(fft_result)
dominant_freq = np.abs(fft_freq[np.argmax(fft_amplitude[1:]) + 1]) # 排除 DC 分量

# 计算厚度
theta = np.radians(angle)
thickness = (dominant_freq * 10000) / (4 * n * np.cos(theta)) # 转换为 nm

return thickness, fft_freq, fft_amplitude

@staticmethod
def calculate_using_fitting(data, n, angle):
    """ 使用拟合方法计算厚度 """
    # 确保数据排序
    data_sorted = data.sort_values('wavenumber').copy()

    # 提取波数和反射率
    wavenumber = data_sorted['wavenumber'].values
    reflectance = data_sorted['smoothed_reflectance'].values if 'smoothed_reflectance' in data_sorted.columns else \
        data_sorted['reflectance'].values

    # 定义拟合函数
    def fit_func(k, d, r0, a, b):
        theta = np.radians(angle)
        wavelength = 10000 / k # 转换为μm
        optical_path = 2 * n * d * np.cos(theta)
        phase = (4 * np.pi / wavelength) * optical_path
        return r0 + a * np.cos(phase + b)

    # 初始猜测值
    initial_guess = [1000, np.mean(reflectance), 0.5 * np.std(reflectance), 0]

    try:
        # 曲线拟合
        params, params_cov = curve_fit(fit_func, wavenumber, reflectance,
                                        p0=initial_guess, maxfev=10000)

        # 计算误差
        residuals = reflectance - fit_func(wavenumber, *params)
        rmse = np.sqrt(np.mean(residuals ** 2))

        # 计算 R2
        ss_total = np.sum((reflectance - np.mean(reflectance)) ** 2)
        ss_residual = np.sum(residuals ** 2)
        r_squared = 1 - (ss_residual / ss_total)

        return thickness, r_squared
    except:
        return None

```

```

thickness = params[0] # 厚度参数

return thickness, rmse, r_squared, params, residuals

except Exception as e:
    print(f"拟合失败: {str(e)}")
    return None, None, None, None, None

@staticmethod
def calculate_using_peaks(data, n, angle):
    """ 使用峰值检测方法计算厚度 """
    # 确保数据排序
    data_sorted = data.sort_values('wavenumber').copy()

    # 提取波数和反射率
    wavenumber = data_sorted['wavenumber'].values
    reflectance = data_sorted['smoothed_reflectance'].values if 'smoothed_reflectance' in
    data_sorted.columns else \
        data_sorted['reflectance'].values

    # 检测峰值
    peaks, _ = signal.find_peaks(reflectance, prominence=1)

    if len(peaks) < 2:
        return None, "峰值数量不足, 无法计算厚度"

    # 计算峰值之间的平均波数差
    peak_wavenumbers = wavenumber[peaks]
    delta_k = np.mean(np.diff(peak_wavenumbers))

    # 计算厚度
    theta = np.radians(angle)
    thickness = (10000) / (2 * n * delta_k * np.cos(theta)) # 转换为 nm

    return thickness, peak_wavenumbers

class MultiAngleFitter:
    """ 多角度联合拟合类 """

    @staticmethod
    def joint_fit(data_list, angles, material_type, carrier_concentration=1e16):
        """ 多角度联合拟合 """
        try:
            # 确保所有数据按波数排序
            sorted_data = [data.sort_values('wavenumber').copy() for data in data_list]

```

```

# 创建一个合并的波数数组, 用于统一计算
all_wavenumbers = np.unique(np.concatenate([data['wavenumber'].values for data
in sorted_data]))


# 计算各角度下的折射率
n_values = [RefractiveIndexModel.calculate_refractive_index(
    all_wavenumbers, material_type, carrier_concentration) for _ in angles]


# 准备用于拟合的数据
combined_wavenumbers = []
combined_reflectance = []
combined_weights = []


for i, data in enumerate(sorted_data):
    # 为每个数据集创建权重
    weights = np.ones(len(data))
    combined_wavenumbers.extend(data['wavenumber'].values)
    combined_reflectance.extend(
        data['smoothed_reflectance'].values if 'smoothed_reflectance' in
data.columns else data[
            'reflectance'].values)
    combined_weights.extend(weights)


# 转换为 numpy 数组
combined_wavenumbers = np.array(combined_wavenumbers)
combined_reflectance = np.array(combined_reflectance)
combined_weights = np.array(combined_weights)


# 定义联合拟合函数
def fit_func(params, k_list, angles, n_list):
    thickness = params[0]
    r0 = params[1]
    a = params[2]

    result = []
    data_idx = 0

    for i, angle in enumerate(angles):
        data_len = len(sorted_data[i])
        k_subset = k_list[data_idx:data_idx + data_len]
        data_idx += data_len

        theta = np.radians(angle)
        wavelength = 10000 / k_subset # 转换为μm
        n = n_list[i][np.searchsorted(all_wavenumbers, k_subset)]

```

```

optical_path = 2 * n * thickness * np.cos(theta)
phase = (4 * np.pi / wavelength) * optical_path
result.extend(r0 + a * np.cos(phase))

return np.array(result)

# 定义误差函数（用于最小化）
def error_func(params):
    predicted = fit_func(params, combined_wavenumbers, angles, n_values)
    return combined_weights * (predicted - combined_reflectance)

# 初始猜测值
initial_guess = [1000, np.mean(combined_reflectance), 0.5 *
np.std(combined_reflectance)]

# 使用最小二乘法进行拟合
result = least_squares(error_func, initial_guess, max_nfev=10000)

if not result.success:
    return None, f"拟合失败: {result.message}"

thickness = result.x[0]

# 计算拟合误差
predicted = fit_func(result.x, combined_wavenumbers, angles, n_values)
residuals = combined_reflectance - predicted
rmse = np.sqrt(np.mean(residuals ** 2))

# 计算 R^2
ss_total = np.sum((combined_reflectance - np.mean(combined_reflectance)) ** 2)
ss_residual = np.sum(residuals ** 2)
r_squared = 1 - (ss_residual / ss_total)

# 为每个角度计算单独的厚度
individual_thicknesses = []
for i, angle in enumerate(angles):
    data = sorted_data[i]
    avg_n = np.mean(n_values[i])
    thickness_i, _, _, _ = ThicknessCalculator.calculate_using_fitting(data, avg_n,
angle)
    individual_thicknesses.append(thickness_i if thickness_i is not None else
thickness)

return {
    'joint_thickness': thickness,
    'individual_thicknesses': individual_thicknesses,
    'rmse': rmse,
}

```

```
'r_squared': r_squared,
'success': True
}, None

except Exception as e:
    return None, f"拟合过程出错: {str(e)}"

class ReportGenerator:
    """ 报告生成器类 """

    @staticmethod
    def generate_pdf_report(output_path, data, results, plots=None):
        """ 生成 PDF 报告, 直接指定字体文件路径 """
        try:
            from reportlab.pdfbase import pdfmetrics
            from reportlab.pdfbase.ttfonts import TTFont

            # 手动指定 SimHei 字体路径 (Windows 系统默认路径)
            simhei_paths = [
                "C:/Windows/Fonts/simhei.ttf",  # 系统字体目录
                "C:/Windows/Fonts/黑体.ttf",  # 可能的中文名称路径
                "C:/Users/你的用户名/AppData/Local/Microsoft/Windows/Fonts/simhei.ttf"
            ]
            # 用户字体目录
            ]

            # 查找可用的 SimHei 字体文件
            font_path = None
            for path in simhei_paths:
                if os.path.exists(path):
                    font_path = path
                    break

            # 如果找到字体, 注册字体
            if font_path:
                pdfmetrics.registerFont(TTFont('SimHei', font_path))
                selected_font = 'SimHei'
            else:
                # 如果没找到, 尝试系统默认字体
                selected_font = "Helvetica"
                print("警告: 未找到 SimHei 字体, 将使用默认字体")

            c = canvas.Canvas(output_path, pagesize=A4)
            width, height = A4

            # 设置标题字体
            c.setFont(selected_font, 16)
```

```
c.drawCentredString(width / 2, height - 40, "碳化硅晶圆厚度分析报告")
```

```
# 设置正文字体
c.setFont(selected_font, 10)
c.drawString(50, height - 70, f"报 告 生 成 日 期 : {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")


# 以下内容保持不变...
# 添加数据集信息
c.setFont(selected_font, 12)
c.drawString(50, height - 100, "1. 数据集信息")
c.line(50, height - 105, width - 50, height - 105)

y_pos = height - 130
for i, (angle, data_item) in enumerate(data.items()):
    c.setFont(selected_font, 10)
    c.drawString(70, y_pos, f"入射角 {angle}° 数据:")
    c.drawString(90, y_pos - 20, f"数据点数量: {len(data_item)}")
    c.drawString(90, y_pos - 40,
               f"波 数 范 围 : {min(data_item['wavenumber']):.2f} - {max(data_item['wavenumber']):.2f} cm⁻¹")
    y_pos -= 60
    if y_pos < 100:
        c.showPage()
        y_pos = height - 100
    c.setFont(selected_font, 10)

# 添加分析结果
c.setFont(selected_font, 12)
c.drawString(50, y_pos, "2. 分析结果")
c.line(50, y_pos - 5, width - 50, y_pos - 5)
y_pos -= 30

if 'individual_results' in results:
    c.setFont(selected_font, 10)
    c.drawString(70, y_pos, "各角度单独分析结果:")
    y_pos -= 20

    for angle, res in results['individual_results'].items():
        c.drawString(90, y_pos, f"入射角 {angle}°:")
        c.drawString(110, y_pos - 20, f"厚度: {res['thickness']:.2f} nm")
        c.drawString(110, y_pos - 40, f"RMSE: {res['rmse']:.4f}")
        c.drawString(110, y_pos - 60, f"R²: {res['r_squared']:.4f}")
        y_pos -= 80
        if y_pos < 100:
            c.showPage()
            y_pos = height - 100
```

```
c.setFont(selected_font, 10)
```

```

if 'joint_result' in results:
    c.setFont(selected_font, 10)
    c.drawString(70, y_pos, "多角度联合拟合结果:")
    y_pos -= 20
    c.drawString(90, y_pos, f"最 优 厚 度 : {results['joint_result']['joint_thickness']:.2f} nm")
    y_pos -= 20
    c.drawString(90, y_pos, f"RMSE: {results['joint_result']['rmse']:.4f}")
    y_pos -= 20
    c.drawString(90, y_pos, f"R2: {results['joint_result']['r_squared']:.4f}")
    y_pos -= 40

# 添加图表
if plots and len(plots) > 0:
    c.setFont(selected_font, 12)
    c.drawString(50, y_pos, "3. 分析图表")
    c.line(50, y_pos - 5, width - 50, y_pos - 5)
    y_pos -= 30

for plot_path in plots:
    if os.path.exists(plot_path):
        c.drawImage(plot_path, 70, y_pos - 200, width=400, height=200)
        y_pos -= 220
    if y_pos < 250:
        c.showPage()
        y_pos = height - 100
    c.setFont(selected_font, 12)

c.save()
return True, "PDF 报告生成成功"
except Exception as e:
    return False, f"PDF 报告生成失败: {str(e)}"

@staticmethod
def generate_word_report(output_path, data, results, plots=None):
    """ 生成 Word 报告 """
    try:
        doc = Document()

        # 添加标题
        doc.add_heading('碳化硅晶圆厚度分析报告', 0)

        # 添加日期
        p = doc.add_paragraph()
        p.add_run(f"报 告 生 成 日 期 : ")
    
```

```

{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"}.italic = True

    # 添加数据集信息
    doc.add_heading('1. 数据集信息', level=1)
    for angle, data_item in data.items():
        doc.add_heading(f'入射角 {angle}° 数据', level=2)
        table = doc.add_table(rows=3, cols=2)
        table.cell(0, 0).text = '数据点数量'
        table.cell(0, 1).text = f'{len(data_item)}'
        table.cell(1, 0).text = '波数范围'
        table.cell(1, 1).text = f'{min(data_item["wavenumber"]):.2f} - {max(data_item["wavenumber"]):.2f} cm-1'
        table.cell(2, 0).text = '预处理情况'
        table.cell(2, 1).text = '已进行异常值检测和数据平滑'
        doc.add_paragraph()

    # 添加分析结果
    doc.add_heading('2. 分析结果', level=1)

    if 'individual_results' in results:
        doc.add_heading('各角度单独分析结果', level=2)
        table = doc.add_table(rows=1, cols=4)
        hdr_cells = table.rows[0].cells
        hdr_cells[0].text = '入射角'
        hdr_cells[1].text = '厚度 (nm)'
        hdr_cells[2].text = 'RMSE'
        hdr_cells[3].text = 'R2'

        for angle, res in results['individual_results'].items():
            row_cells = table.add_row().cells
            row_cells[0].text = f'{angle}°'
            row_cells[1].text = f'{res["thickness"]:.2f}' if res["thickness"] else 'N/A'
            row_cells[2].text = f'{res["rmse"]:.4f}' if res["rmse"] else 'N/A'
            row_cells[3].text = f'{res["r_squared"]:.4f}' if res["r_squared"] else 'N/A'

        doc.add_paragraph()

    if 'joint_result' in results:
        doc.add_heading('多角度联合拟合结果', level=2)
        table = doc.add_table(rows=3, cols=2)
        table.cell(0, 0).text = '最优厚度'
        table.cell(0, 1).text = f'{results["joint_result"]["joint_thickness"]:.2f} nm'
        table.cell(1, 0).text = 'RMSE'
        table.cell(1, 1).text = f'{results["joint_result"]["rmse"]:.4f}'
        table.cell(2, 0).text = 'R2'
        table.cell(2, 1).text = f'{results["joint_result"]["r_squared"]:.4f}'
        doc.add_paragraph()

```

```
# 添加图表
if plots and len(plots) > 0:
    doc.add_heading('3. 分析图表', level=1)
    for plot_path in plots:
        if os.path.exists(plot_path):
            doc.add_picture(plot_path, width=Inches(6))
            doc.add_paragraph()

# 保存文档
doc.save(output_path)
return True, "Word 报告生成成功"
except Exception as e:
    return False, f"Word 报告生成失败: {str(e)}"
```

```
class FirstTab(QWidget):
    """ 第一个功能框：数据预处理与光谱可视化功能 """
    data_updated = pyqtSignal(dict) # 用于传递处理后的数据

    def __init__(self, parent=None):
        super().__init__(parent)
        self.data = {} # 存储导入的数据，格式: {入射角: DataFrame}
        self.processed_data = {} # 存储处理后的数据
        self.init_ui()

    def init_ui(self):
        # 设置全局字体
        font = QFont("KaiTi", 10)
        self.setFont(font)

        # 主布局
        main_layout = QVBoxLayout(self)

        # 标题
        title_label = QLabel("数据预处理与光谱可视化")
        title_font = QFont("KaiTi", 14, QFont.Bold)
        title_label.setFont(title_font)
        title_label.setAlignment(Qt.AlignCenter)
        main_layout.addWidget(title_label)

        # 创建分割器
        splitter = QSplitter(Qt.Vertical)

        # 上半部分：控制面板
        control_panel = QWidget()
```

```
control_layout = QVBoxLayout(control_panel)

# 数据导入区域
import_group = QGroupBox("数据导入")
import_layout = QHBoxLayout()

self.import_btn = QPushButton("导入数据文件")
self.import_btn.clicked.connect(self.import_data)
import_layout.addWidget(self.import_btn)

self.angle_input = QDoubleSpinBox()
self.angle_input.setRange(0, 90)
self.angle_input.setValue(10)
self.angle_input.setSuffix("°")
import_layout.addWidget(QLabel("入射角:"))
import_layout.addWidget(self.angle_input)

self.file_path_label = QLabel("未选择文件")
import_layout.addWidget(self.file_path_label, 1)

import_group.setLayout(import_layout)
control_layout.addWidget(import_group)

# 数据预处理区域
preprocess_group = QGroupBox("数据预处理")
preprocess_layout = QHBoxLayout()

self.outlier_threshold = QDoubleSpinBox()
self.outlier_threshold.setRange(1, 5)
self.outlier_threshold.setValue(3)
self.outlier_threshold.setSuffix("σ")

self.smooth_window = QSpinBox()
self.smooth_window.setRange(3, 21)
self.smooth_window.setValue(5)
self.smooth_window.setSingleStep(2)

preprocess_layout.addWidget(QLabel("异常值阈值:"))
preprocess_layout.addWidget(self.outlier_threshold)
preprocess_layout.addSpacing(20)
preprocess_layout.addWidget(QLabel("平滑窗口大小:"))
preprocess_layout.addWidget(self.smooth_window)
preprocess_layout.addSpacing(20)

self.process_btn = QPushButton("Savitzky-Golay 平滑处理数据")
self.process_btn.clicked.connect(self.process_data)
preprocess_layout.addWidget(self.process_btn)
```

```
self.clear_btn = QPushButton("清除数据")
self.clear_btn.clicked.connect(self.clear_data)
preprocess_layout.addWidget(self.clear_btn)

preprocess_group.setLayout(preprocess_layout)
control_layout.addWidget(preprocess_group)

# 数据可视化控制区域
visualize_group = QGroupBox("光谱可视化")
visualize_layout = QHBoxLayout()

self.plot_raw_check = QCheckBox("原始光谱")
self.plot_raw_check.setChecked(True)

self.plot_processed_check = QCheckBox("处理后光谱")
self.plot_processed_check.setChecked(True)

self.plot_overlay_check = QCheckBox("叠加显示")
self.plot_overlay_check.setChecked(True)

self.plot_btn = QPushButton("绘制光谱")
self.plot_btn.clicked.connect(self.plot_spectra)

visualize_layout.addWidget(self.plot_raw_check)
visualize_layout.addWidget(self.plot_processed_check)
visualize_layout.addWidget(self.plot_overlay_check)
visualize_layout.addWidget(self.plot_btn)

visualize_group.setLayout(visualize_layout)
control_layout.addWidget(visualize_group)

# 信息显示区域
self.info_text = QTextEdit()
self.info_text.setReadOnly(True)
self.info_text.setMinimumHeight(100)
control_layout.addWidget(self.info_text)

splitter.addWidget(control_panel)

# 下半部分：绘图区域
plot_widget = QWidget()
plot_layout = QVBoxLayout(plot_widget)

self.canvas = MplCanvas(self, width=8, height=6, dpi=100)
plot_layout.addWidget(self.canvas)
```

```
splitter.addWidget(plot_widget)

# 设置分割器比例
splitter.setSizes([300, 500])

main_layout.addWidget(splitter)

def import_data(self):
    """ 导入数据文件 """
    file_paths, _ = QFileDialog.getOpenFileNames(
        self, "选择数据文件", "", "Excel files (*.xlsx *.xls);;CSV files (*.csv);;All files (*)"
    )

    if not file_paths:
        return

    angle = self.angle_input.value()
    angle_key = f"{angle:.1f}"

    # 显示导入进度
    progress = QProgressDialog("正在导入数据...", "取消", 0, len(file_paths), self)
    progress.setWindowTitle("导入数据")
    progress.setWindowModality(Qt.WindowModal)

    for i, file_path in enumerate(file_paths):
        progress.setValue(i)
        if progress.wasCanceled():
            break

        self.file_path_label.setText(os.path.basename(file_path))

        # 导入数据
        data, msg = DataProcessor.import_data(file_path)
        self.info_text.append(f"导入 {os.path.basename(file_path)}: {msg}")

        if data is not None:
            self.data[angle_key] = data
            self.info_text.append(f"成功导入入射角 {angle}° 的数据, 共 {len(data)} 个数据点")

        # 自动进行初步处理
        self.process_data(angle_key)

    progress.setValue(len(file_paths))

def process_data(self, angle_key=None):
    """ 处理数据 """

```

```

if not self.data:
    QMessageBox.warning(self, "警告", "请先导入数据")
    return

angles_to_process = [angle_key] if angle_key else self.data.keys()

for angle in angles_to_process:
    if angle not in self.data:
        continue

    data = self.data[angle].copy()

    # 修复缺失值
    data_repaired = DataProcessor.repair_missing(data)
    self.info_text.append(f"修复缺失值: {len(data) - len(data_repaired)} 个缺失值被
修复")

    # 检测并移除异常值
    data_cleaned, outlier_count = DataProcessor.remove_outliers(
        data_repaired, threshold=self.outlier_threshold.value())
    self.info_text.append(f"移除异常值: 共检测到 {outlier_count} 个异常值并移除
")

    # 数据平滑
    window_size = self.smooth_window.value()
    data_smoothed = DataProcessor.smooth_data(data_cleaned, window_size)
    self.info_text.append(f"数据平滑: 使用窗口大小为 {window_size} 的
Savitzky-Golay 滤波器")

    # 保存处理后的数据
    self.processed_data[angle] = data_smoothed

    # 发送数据更新信号
    self.data_updated.emit(self.processed_data)

    # 自动绘图
    self.plot_spectra()

def clear_data(self):
    """ 清除数据 """
    self.data = {}
    self.processed_data = {}
    self.file_path_label.setText("未选择文件")
    self.info_text.clear()
    self.canvas.clear_plot()
    self.data_updated.emit(self.processed_data)

```

```

def plot_spectra(self):
    """ 绘制光谱图 """
    if not self.processed_data and not self.data:
        QMessageBox.warning(self, "警告", "没有可绘制的数据")
        return

    self.canvas.clear_plot()

    # 检查是否需要叠加显示
    overlay = self.plot_overlay_check.isChecked()
    plot_raw = self.plot_raw_check.isChecked()
    plot_processed = self.plot_processed_check.isChecked()

    # 颜色列表, 用于区分不同入射角的数据
    colors = ['blue', 'red', 'green', 'purple', 'orange']
    color_idx = 0

    for angle in (self.processed_data.keys() if self.processed_data else self.data.keys()):
        # 获取数据
        if angle in self.processed_data:
            data = self.processed_data[angle]
        else:
            data = self.data[angle]

        color = colors[color_idx % len(colors)]
        color_idx += 1

        # 绘制原始光谱
        if plot_raw:
            self.canvas.axes.plot(
                data['wavenumber'],
                data['reflectance'],
                label=f'入射角 {angle}° (原始)',
                color=color,
                alpha=0.5,
                linestyle='-' )

        # 绘制处理后光谱
        if plot_processed and 'smoothed_reflectance' in data.columns:
            label = f'入射角 {angle}° (处理后)' if not overlay else f'入射角 {angle}°'
            self.canvas.axes.plot(
                data['wavenumber'],
                data['smoothed_reflectance'],
                label=label,
                color=color,
                linewidth=2)

```

```
)\n\n# 设置图表属性\nself.canvas.axes.set_title('红外干涉光谱', fontproperties='KaiTi', fontsize=12)\nself.canvas.axes.set_xlabel(r'波数 ($\mathrm{cm}^{-1}$)', fontsize=10)\nself.canvas.axes.set_ylabel('反射率 (%)', fontsize=10)\nself.canvas.axes.legend(prop={'family': ['KaiTi', 'SimHei']})\nself.canvas.axes.grid(True, linestyle='--', alpha=0.7)\n\n# 调整布局并绘制\nself.canvas.fig.tight_layout()\nself.canvas.draw()\n\n\nclass SecondTab(QWidget):\n    """ 第二个功能框： 折射率模型与干涉模拟功能 """\n\n    def __init__(self, parent=None):\n        super().__init__(parent)\n        self.data = {} # 存储从第一个功能框获取的数据\n        self.init_ui()\n\n    def init_ui(self):\n        # 设置全局字体\n        font = QFont("KaiTi", 10)\n        self.setFont(font)\n\n        # 主布局\n        main_layout = QVBoxLayout(self)\n\n        # 标题\n        title_label = QLabel("折射率模型与干涉模拟功能")\n        title_font = QFont("KaiTi", 14, QFont.Bold)\n        title_label.setFont(title_font)\n        title_label.setAlignment(Qt.AlignCenter)\n        main_layout.addWidget(title_label)\n\n        # 创建分割器\n        splitter = QSplitter(Qt.Vertical)\n\n        # 上半部分： 控制面板\n        control_panel = QWidget()\n        control_layout = QVBoxLayout(control_panel)\n\n        # 材料参数设置区域\n        material_group = QGroupBox("材料与光学参数")\n        material_layout = QFormLayout()
```

```
self.material_combo = QComboBox()
self.material_combo.addItems(["4H-SiC", "6H-SiC", "其他"])

self.carrier_spin = QDoubleSpinBox()
self.carrier_spin.setRange(1e15, 1e19)
self.carrier_spin.setValue(1e16)
self.carrier_spin.setSuffix(" cm-3")
self.carrier_spin.setDecimals(0)

self.angle_spin = QDoubleSpinBox()
self.angle_spin.setRange(0, 90)
self.angle_spin.setValue(10)
self.angle_spin.setSuffix("°")

self.thickness_spin = QDoubleSpinBox()
self.thickness_spin.setRange(100, 10000)
self.thickness_spin.setValue(1000)
self.thickness_spin.setSuffix(" nm")

material_layout.addRow("材料类型:", self.material_combo)
material_layout.addRow("载流子浓度:", self.carrier_spin)
material_layout.addRow("入射角:", self.angle_spin)
material_layout.addRow("外延层厚度:", self.thickness_spin)

material_group.setLayout(material_layout)
control_layout.addWidget(material_group)

# 模拟控制区域
simulation_group = QGroupBox("模拟控制")
simulation_layout = QHBoxLayout()

self.wavenumber_start = QDoubleSpinBox()
self.wavenumber_start.setRange(400, 4000)
self.wavenumber_start.setValue(400)
self.wavenumber_start.setSuffix(" cm-1")

self.wavenumber_end = QDoubleSpinBox()
self.wavenumber_end.setRange(400, 4000)
self.wavenumber_end.setValue(1000)
self.wavenumber_end.setSuffix(" cm-1")

self.points_spin = QSpinBox()
self.points_spin.setRange(100, 10000)
self.points_spin.setValue(1000)
self.points_spin.setSuffix(" 点")
```

```
self.plot_refractive_check = QCheckBox("折射率曲线")
self.plot_refractive_check.setChecked(True)

self.plot_simulation_check = QCheckBox("模拟光谱")
self.plot_simulation_check.setChecked(True)

self.plot_measured_check = QCheckBox("实测光谱")
self.plot_measured_check.setChecked(True)

self.simulate_btn = QPushButton("运行模拟")
self.simulate_btn.clicked.connect(self.run_simulation)

simulation_layout.addWidget(QLabel("波数范围:"))
simulation_layout.addWidget(self.wavenumber_start)
simulation_layout.addWidget(QLabel("至"))
simulation_layout.addWidget(self.wavenumber_end)
simulation_layout.addSpacing(10)
simulation_layout.addWidget(QLabel("数据点:"))
simulation_layout.addWidget(self.points_spin)
simulation_layout.addSpacing(10)
simulation_layout.addWidget(self.plot_refractive_check)
simulation_layout.addWidget(self.plot_simulation_check)
simulation_layout.addWidget(self.plot_measured_check)
simulation_layout.addWidget(self.simulate_btn)

simulation_group.setLayout(simulation_layout)
control_layout.addWidget(simulation_group)

# 信息显示区域
self.info_text = QTextEdit()
self.info_text.setReadOnly(True)
self.info_text.setMinimumHeight(100)
control_layout.addWidget(self.info_text)

splitter.addWidget(control_panel)

# 下半部分：绘图区域
plot_widget = QWidget()
plot_layout = QVBoxLayout(plot_widget)

# 创建两个画布，一个用于折射率，一个用于光谱
self.fig, (self.ax1, self.ax2) = plt.subplots(2, 1, figsize=(8, 8), dpi=100)
self.canvas = FigureCanvas(self.fig)

plot_layout.addWidget(self.canvas)

splitter.addWidget(plot_widget)
```

```
# 设置分割器比例
splitter.setSizes([300, 600])

main_layout.addWidget(splitter)

# 初始化图表
self.init_plots()

def init_plots(self):
    """ 初始化图表 """
    self.ax1.clear()
    self.ax2.clear()

    self.ax1.set_title('折射率随波数变化', fontproperties='KaiTi', fontsize=12)
    self.ax1.set_xlabel('波数 ($\mathrm{cm}^{-1}$)', fontproperties='KaiTi', fontsize=10)
    self.ax1.set_ylabel('折射率', fontproperties='KaiTi', fontsize=10)
    self.ax1.grid(True, linestyle='--', alpha=0.7)

    self.ax2.set_title('干涉光谱模拟', fontproperties='KaiTi', fontsize=12)
    self.ax2.set_xlabel('波数 ($\mathrm{cm}^{-1}$)', fontproperties='KaiTi', fontsize=10)
    self.ax2.set_ylabel('反射率 (%)', fontproperties='KaiTi', fontsize=10)
    self.ax2.grid(True, linestyle='--', alpha=0.7)

    # 确保初始状态不显示任何数据曲线
    self.fig.tight_layout()
    self.canvas.draw()

def update_data(self, data):
    """ 更新数据 """
    self.data = data
    if data:
        angles = list(data.keys())
        self.angle_spin.setValue(float(angles[0]))
        self.info_text.append(f"已更新数据，包含 {len(angles)} 个入射角的光谱数据")

def run_simulation(self):
    """ 运行模拟 """
    # 清除之前的绘图
    self.ax1.clear()
    self.ax2.clear()

    # 获取参数
    material_type = self.material_combo.currentText()
    carrier_concentration = self.carrier_spin.value()
    angle = self.angle_spin.value()
    thickness = self.thickness_spin.value()
```

```

wavenumber_start = self.wavenumber_start.value()
wavenumber_end = self.wavenumber_end.value()
num_points = self.points_spin.value()

# 验证参数
if wavenumber_start >= wavenumber_end:
    QMessageBox.warning(self, "参数错误", "波数起始值必须小于结束值")
    return

# 生成波数数组
wavenumber = np.linspace(wavenumber_start, wavenumber_end, num_points)

# 计算折射率
n = RefractiveIndexModel.calculate_refractive_index(
    wavenumber, material_type, carrier_concentration)

# 模拟干涉光谱
simulated_reflectance = RefractiveIndexModel.simulate_interference_spectrum(
    wavenumber, thickness, np.mean(n), angle)

# 显示信息
self.info_text.clear()
self.info_text.append(f"模拟参数:")
self.info_text.append(f"材料类型: {material_type}")
self.info_text.append(f"载流子浓度: {carrier_concentration:.2e} cm-3")
self.info_text.append(f"入射角: {angle}°")
self.info_text.append(f"外延层厚度: {thickness} nm")
self.info_text.append(f"波数范围: {wavenumber_start} - {wavenumber_end} cm-1")
self.info_text.append(f"平均折射率: {np.mean(n):.4f}")

# 绘制折射率曲线
if self.plot_refractive_check.isChecked():
    self.ax1.plot(wavenumber, n, 'b-', linewidth=2)
    self.ax1.set_title('折射率随波数变化', fontproperties='KaiTi', fontsize=12)
    self.ax1.set_xlabel(' 波 数     ($\mathrm{cm^{-1}}$)', fontproperties='KaiTi',
    fontsize=10)
    self.ax1.set_ylabel('折射率', fontproperties='KaiTi', fontsize=10)
    self.ax1.grid(True, linestyle='--', alpha=0.7)

# 绘制模拟光谱
if self.plot_simulation_check.isChecked():
    self.ax2.plot(wavenumber, simulated_reflectance, 'r-', linewidth=2, label='模拟光谱')

# 绘制实测光谱 (如果有)
angle_key = f"{angle:.1f}"
if self.plot_measured_check.isChecked() and angle_key in self.data:

```

```

        data = self.data[angle_key]
        # 筛选在当前波数范围内的数据
        mask = (data['wavenumber'] >= wavenumber_start) & (data['wavenumber'] <=
wavenumber_end)
        filtered_data = data[mask]

        if not filtered_data.empty:
            self.ax2.plot(
                filtered_data['wavenumber'],
                filtered_data['smoothed_reflectance'] if 'smoothed_reflectance' in
filtered_data.columns else
                filtered_data['reflectance'],
                'b--',
                linewidth=1.5,
                label='实测光谱'
            )
        else:
            self.info_text.append("警告: 没有在指定波数范围内的实测数据")

        # 设置光谱图属性
        self.ax2.set_title('干涉光谱', fontproperties='KaiTi', fontsize=12)
        self.ax2.set_xlabel('波数 ($\mathbf{cm}^{-1}$)', fontproperties='KaiTi', fontsize=10)
        self.ax2.set_ylabel('反射率 (%)', fontproperties='KaiTi', fontsize=10)
        self.ax2.legend(prop={'family': ['KaiTi', 'SimHei']})
        self.ax2.grid(True, linestyle='--', alpha=0.7)

        # 调整布局并绘制
        self.fig.tight_layout()
        self.canvas.draw()

class ThirdTab(QWidget):
    """ 第三个功能框: 厚度确定功能 """
    thickness_results_updated = pyqtSignal(dict) # 用于传递厚度计算结果

    def __init__(self, parent=None):
        super().__init__(parent)
        self.data = {} # 存储从第一个功能框获取的数据
        self.results = {} # 存储计算结果
        self.init_ui()

    def init_ui(self):
        # 设置全局字体
        font = QFont("KaiTi", 10)
        self.setFont(font)

        # 主布局

```

```
main_layout = QVBoxLayout(self)

# 标题
title_label = QLabel("厚度确定功能")
title_font = QFont("KaiTi", 14, QFont.Bold)
title_label.setFont(title_font)
title_label.setAlignment(Qt.AlignCenter)
main_layout.addWidget(title_label)

# 创建分割器
splitter = QSplitter(Qt.Vertical)

# 左侧：控制面板和结果显示
left_panel = QWidget()
left_layout = QVBoxLayout(left_panel)

# 参数设置区域
param_group = QGroupBox("计算参数")
param_layout = QFormLayout()

self.angle_combo = QComboBox()

self.material_combo = QComboBox()
self.material_combo.addItems(["4H-SiC", "6H-SiC", "其他"])

self.carrier_spin = QDoubleSpinBox()
self.carrier_spin.setRange(1e15, 1e19)
self.carrier_spin.setValue(1e16)
self.carrier_spin.setSuffix(" cm-3")

self.method_combo = QComboBox()
self.method_combo.addItems(["快速傅里叶变换(FFT)", "曲线拟合", "峰值检测"])

param_layout.addRow("入射角:", self.angle_combo)
param_layout.addRow("材料类型:", self.material_combo)
param_layout.addRow("载流子浓度:", self.carrier_spin)
param_layout.addRow("计算方法:", self.method_combo)

param_group.setLayout(param_layout)
left_layout.addWidget(param_group)

# 计算按钮
calc_btn = QPushButton("计算厚度")
calc_btn.clicked.connect(self.calculate_thickness)
left_layout.addWidget(calc_btn)

# 结果显示区域
```

```
result_group = QGroupBox("计算结果")
result_layout = QFormLayout()

self.thickness_label = QLabel("N/A")
self.rmse_label = QLabel("N/A")
self.r_squared_label = QLabel("N/A")

result_layout.addRow("外延层厚度 (nm):", self.thickness_label)
result_layout.addRow("均方根误差 (RMSE):", self.rmse_label)
result_layout.addRow("决定系数 (R2):", self.r_squared_label)

result_group.setLayout(result_layout)
left_layout.addWidget(result_group)

# 信息显示区域
self.info_text = QTextEdit()
self.info_text.setReadOnly(True)
self.info_text.setMinimumHeight(100)
left_layout.addWidget(self.info_text)

# 右侧: 绘图区域
right_panel = QWidget()
right_layout = QVBoxLayout(right_panel)

# 创建两个画布, 一个用于光谱和拟合结果, 一个用于 FFT 结果
self.fig, (self.ax1, self.ax2) = plt.subplots(2, 1, figsize=(8, 8), dpi=100)
self.canvas = FigureCanvas(self.fig)

right_layout.addWidget(self.canvas)

# 水平分割器
h_splitter = QSplitter(Qt.Horizontal)
h_splitter.addWidget(left_panel)
h_splitter.addWidget(right_panel)
h_splitter.setSizes([300, 600])

splitter.addWidget(h_splitter)

main_layout.addWidget(splitter)

# 初始化图表
self.init_plots()

def init_plots(self):
    """ 初始化图表 """
    self.ax1.clear()
    self.ax2.clear()
```

```
self.ax1.set_title('光谱与拟合结果', fontproperties='KaiTi', fontsize=12)
self.ax1.set_xlabel('波数 ($\mathrm{cm}^{-1}$)', fontproperties='KaiTi', fontsize=10)
self.ax1.set_ylabel('反射率 (%)', fontproperties='KaiTi', fontsize=10)
self.ax1.grid(True, linestyle='--', alpha=0.7)

self.ax2.set_title('FFT 分析结果', fontproperties='KaiTi', fontsize=12)
self.ax2.set_xlabel('频率', fontproperties='KaiTi', fontsize=10)
self.ax2.set_ylabel('振幅', fontproperties='KaiTi', fontsize=10)
self.ax2.grid(True, linestyle='--', alpha=0.7)

# 确保初始状态不显示任何数据曲线
self.fig.tight_layout()
self.canvas.draw()

def update_data(self, data):
    """ 更新数据 """
    self.data = data
    self.angle_combo.clear()
    self.angle_combo.addItems(list(data.keys()))
    if data:
        self.info_text.append(f"已更新数据，包含 {len(data)} 个入射角的光谱数据")

def calculate_thickness(self):
    """ 计算厚度 """
    if not self.data:
        QMessageBox.warning(self, "警告", "请先在第一个功能框中导入并处理数据")
        return

    # 清除之前的绘图
    self.ax1.clear()
    self.ax2.clear()

    # 获取参数
    angle_str = self.angle_combo.currentText()
    angle = float(angle_str)
    material_type = self.material_combo.currentText()
    carrier_concentration = self.carrier_spin.value()
    method = self.method_combo.currentIndex() # 0: FFT, 1: 曲线拟合, 2: 峰值检测

    # 获取数据
    if angle_str not in self.data:
        QMessageBox.warning(self, "警告", "所选入射角的数据不存在")
        return

    data = self.data[angle_str]
```

```

# 计算折射率
n = RefractiveIndexModel.calculate_refractive_index(
    data['wavenumber'].values, material_type, carrier_concentration)
avg_n = np.mean(n)

# 显示信息
self.info_text.clear()
self.info_text.append(f"计算参数:")
self.info_text.append(f"入射角: {angle}")
self.info_text.append(f"材料类型: {material_type}")
self.info_text.append(f"载流子浓度: {carrier_concentration:.2e} cm-3")
self.info_text.append(f"平均折射率: {avg_n:.4f}")
self.info_text.append(f"计算方法: {self.method_combo.currentText()}")


thickness = None
rmse = None
r_squared = None

# 根据选择的方法计算厚度
if method == 0: # FFT 方法
    thickness, fft_freq, fft_amplitude = ThicknessCalculator.calculate_using_fft(data,
avg_n, angle)

    # 绘制 FFT 结果
    self.ax2.plot(np.abs(fft_freq), np.abs(fft_amplitude), 'b-')
    self.ax2.set_title('FFT 分析结果', fontproperties='KaiTi', fontsize=12)
    self.ax2.set_xlabel('频率', fontproperties='KaiTi', fontsize=10)
    self.ax2.set_ylabel('振幅', fontproperties='KaiTi', fontsize=10)
    self.ax2.grid(True, linestyle='--', alpha=0.7)

    # 限制 x 轴范围以便更好地观察
    max_freq = np.max(np.abs(fft_freq)) * 0.2
    self.ax2.set_xlim(0, max_freq)

    self.info_text.append(f"FFT 方法计算完成")

elif method == 1: # 曲线拟合方法
    thickness, rmse, r_squared, params, residuals =
ThicknessCalculator.calculate_using_fitting(data, avg_n,
angle)

    if thickness is not None:
        # 绘制拟合曲线
        wavenumber = data['wavenumber'].values
        reflectance = data['smoothed_reflectance'].values if 'smoothed_reflectance' in
data.columns else data[

```

```
'reflectance'].values
```

```
# 计算拟合值
```

```
theta = np.radians(angle)
```

```
wavelength = 10000 / wavenumber # 转换为μm
```

```
optical_path = 2 * avg_n * thickness * np.cos(theta)
```

```
phase = (4 * np.pi / wavelength) * optical_path
```

```
fitted = params[1] + params[2] * np.cos(phase + params[3]))
```

```
self.ax1.plot(wavenumber, fitted, 'r-', linewidth=2, label='拟合曲线')
```

```
self.info_text.append(f"曲线拟合完成, RMSE: {rmse:.4f}, R2: {r_squared:.4f}")
```

```
else:
```

```
self.info_text.append("曲线拟合失败")
```

```
elif method == 2: # 峰值检测方法
```

```
result = ThicknessCalculator.calculate_using_peaks(data, avg_n, angle)
```

```
if isinstance(result, tuple) and len(result) == 2:
```

```
thickness, peak_wavenumbers = result
```

```
# 绘制峰值
```

```
wavenumber = data['wavenumber'].values
```

```
reflectance = data['smoothed_reflectance'].values if 'smoothed_reflectance' in data.columns else data[
```

```
'reflectance'].values
```

```
self.ax1.scatter(peak_wavenumbers, [np.interp(k, wavenumber, reflectance)
```

```
for k in peak_wavenumbers],
```

```
color='red', s=50, zorder=5, label='检测到的峰值')
```

```
self.info_text.append(f"峰值检测完成, 共检测到 {len(peak_wavenumbers)}
```

```
个峰值")
```

```
else:
```

```
thickness = None
```

```
self.info_text.append(f"峰值检测失败: {result}")
```

```
# 显示结果
```

```
if thickness is not None:
```

```
self.thickness_label.setText(f"{thickness:.2f}")
```

```
self.rmse_label.setText(f"{rmse:.4f}" if rmse is not None else "N/A")
```

```
self.r_squared_label.setText(f"{r_squared:.4f}" if r_squared is not None else "N/A")
```

```
# 保存结果
```

```
self.results[angle_str] = {
```

```
'thickness': thickness,
```

```
'rmse': rmse,
```

```
'r_squared': r_squared,
```

```
'method': self.method_combo.currentText(),
```

```
'material': material_type,
```

```

    'carrier_concentration': carrier_concentration
}

# 发送结果更新信号
self.thickness_results_updated.emit(self.results)
else:
    self.thickness_label.setText("计算失败")
    self.rmse_label.setText("N/A")
    self.r_squared_label.setText("N/A")

# 绘制原始光谱
wavenumber = data['wavenumber'].values
reflectance = data['smoothed_reflectance'].values if 'smoothed_reflectance' in
data.columns else data[
    'reflectance'].values
self.ax1.plot(wavenumber, reflectance, 'b-', linewidth=1.5, label='实测光谱')

# 设置光谱图属性
self.ax1.set_title('光谱与分析结果', fontproperties='KaiTi', fontsize=12)
self.ax1.set_xlabel('波数 ($\mathrm{cm}^{-1}$)', fontproperties='KaiTi', fontsize=10)
self.ax1.set_ylabel('反射率 (%)', fontproperties='KaiTi', fontsize=10)
self.ax1.legend(prop={'family': ['KaiTi', 'SimHei']})
self.ax1.grid(True, linestyle='--', alpha=0.7)

# 调整布局并绘制
self.fig.tight_layout()
self.canvas.draw()

class FourthTab(QWidget):
    """ 第四个功能框：多角度联合拟合与厚度优化功能 """
    joint_results_updated = pyqtSignal(dict) # 用于传递联合拟合结果

    def __init__(self, parent=None):
        super().__init__(parent)
        self.data = {} # 存储从第一个功能框获取的数据
        self.individual_results = {} # 存储单独计算结果
        self.joint_result = None # 存储联合拟合结果
        self.angle_checkboxes = {} # 存储角度选择框
        self.init_ui()

    def init_ui(self):
        # 设置全局字体
        font = QFont("KaiTi", 10)
        self.setFont(font)

        # 主布局

```

```
main_layout = QVBoxLayout(self)

# 标题
title_label = QLabel("多角度联合拟合与厚度优化功能")
title_font = QFont("KaiTi", 14, QFont.Bold)
title_label.setFont(title_font)
title_label.setAlignment(Qt.AlignCenter)
main_layout.addWidget(title_label)

# 创建分割器
splitter = QSplitter(Qt.Vertical)

# 上半部分：控制面板和结果显示
top_panel = QWidget()
top_layout = QHBoxLayout(top_panel)

# 左侧：参数设置
param_widget = QWidget()
param_layout = QVBoxLayout(param_widget)

param_group = QGroupBox("拟合参数")
param_form = QFormLayout()

self.material_combo = QComboBox()
self.material_combo.addItems(["4H-SiC", "6H-SiC", "其他"])

self.carrier_spin = QDoubleSpinBox()
self.carrier_spin.setRange(1e15, 1e19)
self.carrier_spin.setValue(1e16)
self.carrier_spin.setSuffix(" cm-3")

param_form.addRow("材料类型:", self.material_combo)
param_form.addRow("载流子浓度:", self.carrier_spin)

param_group.setLayout(param_form)
param_layout.addWidget(param_group)

# 角度选择
angle_group = QGroupBox("选择用于联合拟合的入射角")
self.angle_layout = QVBoxLayout()
angle_group.setLayout(self.angle_layout)
param_layout.addWidget(angle_group)

# 拟合按钮
fit_btn = QPushButton("执行联合拟合")
fit_btn.clicked.connect(self.perform_joint_fit)
param_layout.addWidget(fit_btn)
```

```
param_layout.addStretch()
top_layout.addWidget(param_widget, 1)

# 右侧: 结果显示
result_widget = QWidget()
result_layout = QVBoxLayout(result_widget)

# 联合拟合结果
joint_result_group = QGroupBox("联合拟合结果")
joint_result_form = QFormLayout()

self.joint_thickness_label = QLabel("N/A")
self.joint_rmse_label = QLabel("N/A")
self.joint_r_squared_label = QLabel("N/A")

joint_result_form.addRow("最优厚度 (nm):", self.joint_thickness_label)
joint_result_form.addRow("均方根误差 (RMSE):", self.joint_rmse_label)
joint_result_form.addRow("决定系数 (R2):", self.joint_r_squared_label)

joint_result_group.setLayout(joint_result_form)
result_layout.addWidget(joint_result_group)

# 厚度对比
comparison_group = QGroupBox("厚度对比 (nm)")
self.comparison_table = QTableWidget()
self.comparison_table.setColumnCount(3)
self.comparison_table.setHorizontalHeaderLabels(["入射角", "单独计算", "联合拟合"])
comparison_group.setLayout(QVBoxLayout())
comparison_group.layout().addWidget(self.comparison_table)
result_layout.addWidget(comparison_group)

# 信息显示
self.info_text = QTextEdit()
self.info_text.setReadOnly(True)
self.info_text.setMinimumHeight(100)
result_layout.addWidget(self.info_text)

top_layout.addWidget(result_widget, 2)

splitter.addWidget(top_panel)

# 下半部分: 绘图区域
plot_widget = QWidget()
plot_layout = QVBoxLayout(plot_widget)

# 创建两个画布, 一个用于光谱拟合对比, 一个用于厚度误差分析
```

```
self.fig, (self.ax1, self.ax2) = plt.subplots(2, 1, figsize=(10, 8), dpi=100)
self.canvas = FigureCanvas(self.fig)

plot_layout.addWidget(self.canvas)

splitter.addWidget(plot_widget)

# 设置分割器比例
splitter.setSizes([400, 500])

main_layout.addWidget(splitter)

# 初始化图表
self.init_plots()

def init_plots(self):
    """ 初始化图表 """
    self.ax1.clear()
    self.ax2.clear()

    self.ax1.set_title('多角度光谱与拟合结果', fontproperties='KaiTi', fontsize=12)
    self.ax1.set_xlabel('波数 ($\mathrm{cm}^{-1}$)', fontproperties='KaiTi', fontsize=10)
    self.ax1.set_ylabel('反射率 (%)', fontproperties='KaiTi', fontsize=10)
    self.ax1.grid(True, linestyle='--', alpha=0.7)

    self.ax2.set_title('厚度计算结果对比', fontproperties='KaiTi', fontsize=12)
    self.ax2.set_xlabel('入射角 ($^\circ$)', fontproperties='KaiTi', fontsize=10)
    self.ax2.set_ylabel('厚度 (nm)', fontproperties='KaiTi', fontsize=10)
    self.ax2.grid(True, linestyle='--', alpha=0.7)

    # 确保初始状态不显示任何数据曲线
    self.fig.tight_layout()
    self.canvas.draw()

def update_data(self, data):
    """ 更新数据 """
    self.data = data

    # 更新角度选择框
    # 先清除现有控件
    while self.angle_layout.count():
        item = self.angle_layout.takeAt(0)
        widget = item.widget()
        if widget:
            widget.deleteLater()

    # 添加新的角度选择框
```

```

self.angle_checkboxes = {}
for angle in data.keys():
    checkbox = QCheckBox(f"{angle}°")
    checkbox.setChecked(True)
    self.angle_checkboxes[angle] = checkbox
    self.angle_layout.addWidget(checkbox)

if data:
    self.info_text.append(f"已更新数据, 包含 {len(data)} 个入射角的光谱数据")

def update_individual_results(self, results):
    """ 更新单独计算结果 """
    self.individual_results = results
    self.info_text.append(f"已更新单独计算结果, 共 {len(results)} 个入射角的数据")
    self.update_comparison_table()

def update_comparison_table(self):
    """ 更新对比表格 """
    self.comparison_table.setRowCount(0)

    if not self.individual_results:
        return

    # 添加单独计算结果
    for row, (angle, result) in enumerate(self.individual_results.items()):
        self.comparison_table.insertRow(row)
        self.comparison_table.setItem(row, 0, QTableWidgetItem(angle))
        self.comparison_table.setItem(row, 1, QTableWidgetItem(
            f"{result['thickness']:.2f}" if result['thickness'] else "N/A"))
        self.comparison_table.setItem(row, 2, QTableWidgetItem("N/A"))

    # 如果有联合拟合结果, 更新表格
    if self.joint_result:
        joint_thickness = self.joint_result['joint_thickness']
        for row in range(self.comparison_table.rowCount()):
            angle_item = self.comparison_table.item(row, 0)
            if angle_item and angle_item.text() in self.individual_results:
                self.comparison_table.setItem(row,
                                              2,
                                              QTableWidgetItem(f"{joint_thickness:.2f}"))

def perform_joint_fit(self):
    """ 执行联合拟合 """
    if not self.data:
        QMessageBox.warning(self, "警告", "请先在第一个功能框中导入并处理数据")
        return

    if len(self.data) < 2:

```

```

    QMessageBox.warning(self, "警告", "至少需要两个不同入射角的数据才能进行
联合拟合")
    return

    # 获取选中的角度
    selected_angles = [angle for angle, checkbox in self.angle_checkboxes.items() if
checkbox.isChecked()]
    if len(selected_angles) < 2:
        QMessageBox.warning(self, "警告", "请至少选择两个不同入射角的数据")
        return

    # 获取参数
    material_type = self.material_combo.currentText()
    carrier_concentration = self.carrier_spin.value()

    # 准备数据
    data_list = [self.data[angle] for angle in selected_angles]
    angles = [float(angle) for angle in selected_angles]

    # 显示信息
    self.info_text.clear()
    self.info_text.append(f"联合拟合参数:")
    self.info_text.append(f"材料类型: {material_type}")
    self.info_text.append(f"载流子浓度: {carrier_concentration:.2e} cm-3")
    self.info_text.append(f"选中的入射角: {', '.join(selected_angles)}°")
    self.info_text.append("正在执行联合拟合, 请稍候...")

try:
    # 执行联合拟合
    result, error = MultiAngleFitter.joint_fit(data_list, angles, material_type,
carrier_concentration)

    if error:
        self.info_text.append(f"拟合失败: {error}")
        return

    if not result or not result.get('success', False):
        self.info_text.append("联合拟合失败, 未返回有效结果")
        return

    # 保存结果
    self.joint_result = result
    self.info_text.append(f"联合拟合完成, 最优厚度: {result['joint_thickness']:.2f}
nm")
    self.info_text.append(f"拟合优度: RMSE = {result['rmse']:.4f}, R2 =
{result['r_squared']:.4f}")

```

```

# 更新结果显示
self.joint_thickness_label.setText(f"{{result['joint_thickness']:.2f}}")
self.joint_rmse_label.setText(f"{{result['rmse']:.4f}}")
self.joint_r_squared_label.setText(f"{{result['r_squared']:.4f}}")

# 更新对比表格
self.update_comparison_table()

# 发送结果更新信号
self.joint_results_updated.emit({
    'joint_result': result,
    'individual_results': self.individual_results
})

# 绘制结果
self.plot_results(selected_angles, angles, data_list, result, material_type,
carrier_concentration)

except Exception as e:
    self.info_text.append(f"拟合过程出错: {str(e)}")
    import traceback
    traceback.print_exc()

def plot_results(self, angle_strings, angles, data_list, result, material_type,
carrier_concentration):
    """ 绘制拟合结果 """
    # 清除之前的绘图
    self.ax1.clear()
    self.ax2.clear()

    # 颜色列表
    colors = ['blue', 'red', 'green', 'purple', 'orange']
    line_styles = [':', '--', '-.', ':'] # 线型列表

    # 绘制光谱与拟合结果
    for i, (angle_str, angle, data) in enumerate(zip(angle_strings, angles, data_list)):
        color = colors[i % len(colors)]
        line_style = line_styles[i % len(line_styles)]

        # 绘制实测光谱 - 修复格式字符串
        wavenumber = data['wavenumber'].values
        reflectance = data['smoothed_reflectance'].values if 'smoothed_reflectance' in
data.columns else data[
            'reflectance'].values
        # 使用正确的格式字符串: 颜色+线型, 例如 'b-' 而不是 'blue-'
        self.ax1.plot(wavenumber, reflectance, f'{color[0]}{line_style}', linewidth=1.5,
alpha=0.7,

```

```
label=f'实测光谱 ({angle_str}°)'
```

```
# 计算拟合光谱
n = RefractiveIndexModel.calculate_refractive_index(wavenumber, material_type,
carrier_concentration)
theta = np.radians(angle)
wavelength = 10000 / wavenumber # 转换为μm
optical_path = 2 * n * result['joint_thickness'] * np.cos(theta)
phase = (4 * np.pi / wavelength) * optical_path

# 简单的拟合模型
r0 = np.mean(reflectance)
a = 0.5 * np.std(reflectance)
fitted = r0 + a * np.cos(phase)

# 绘制拟合光谱 - 修复格式字符串
self.ax1.plot(wavenumber, fitted, f'{color[0]}'{line_styles[(i + 1) % len(line_styles)]},
linewidth=2,
label=f'拟合光谱 ({angle_str}°)')

# 设置光谱图属性
self.ax1.set_title('多角度光谱与拟合结果', fontproperties='KaiTi', fontsize=12)
self.ax1.set_xlabel('波数 ($\mathrm{cm^{-1}}$)', fontproperties='KaiTi', fontsize=10)
self.ax1.set_ylabel('反射率 (%)', fontproperties='KaiTi', fontsize=10)
self.ax1.legend(prop={'family': ['KaiTi', 'SimHei']}, loc='upper right')
self.ax1.grid(True, linestyle='--', alpha=0.7)

# 绘制厚度对比 (保持不变)
angles_numeric = [float(angle) for angle in angle_strings]
individual_thicknesses = result['individual_thicknesses']
joint_thickness = [result['joint_thickness']] * len(angles_numeric)

x = np.arange(len(angles_numeric))
width = 0.35

self.ax2.bar(x - width / 2, individual_thicknesses, width, label='单独计算', color='blue')
self.ax2.bar(x + width / 2, joint_thickness, width, label='联合拟合', color='red')

self.ax2.set_xticks(x)
self.ax2.set_xticklabels(angle_strings)
self.ax2.set_title('厚度计算结果对比', fontproperties='KaiTi', fontsize=12)
self.ax2.set_xlabel('入射角 (°)', fontproperties='KaiTi', fontsize=10)
self.ax2.set_ylabel('厚度 (nm)', fontproperties='KaiTi', fontsize=10)
self.ax2.legend(prop={'family': ['KaiTi', 'SimHei']})
self.ax2.grid(True, linestyle='--', alpha=0.7)

# 调整布局并绘制
```

```
self.fig.tight_layout()
self.canvas.draw()

class FifthTab(QWidget):
    """ 第五个功能框：结果报告导出功能 """

    def __init__(self, parent=None):
        super().__init__(parent)
        self.data = {} # 存储数据
        self.results = {} # 存储分析结果
        self.temp_plot_paths = [] # 存储临时图表路径
        self.init_ui()

    def init_ui(self):
        # 设置全局字体
        font = QFont("KaiTi", 10)
        self.setFont(font)

        # 主布局
        main_layout = QVBoxLayout(self)

        # 标题
        title_label = QLabel("结果报告导出功能")
        title_font = QFont("KaiTi", 14, QFont.Bold)
        title_label.setFont(title_font)
        title_label.setAlignment(Qt.AlignCenter)
        main_layout.addWidget(title_label)

        # 创建分割器
        splitter = QSplitter(Qt.Vertical)

        # 上半部分：报告设置
        settings_panel = QWidget()
        settings_layout = QVBoxLayout(settings_panel)

        # 报告参数设置
        report_group = QGroupBox("报告设置")
        report_layout = QFormLayout()

        self.report_title = QTextEdit("碳化硅晶圆厚度分析报告")
        self.report_title.setMaximumHeight(50)

        self.report_author = QTextEdit("")
        self.report_author.setMaximumHeight(30)

        self.report_notes = QTextEdit("")
```

```
self.report_notes.setMaximumHeight(80)

self.format_combo = QComboBox()
self.format_combo.addItem("PDF", "Word (DOCX)")

report_layout.addRow("报告标题:", self.report_title)
report_layout.addRow("作者:", self.report_author)
report_layout.addRow("备注信息:", self.report_notes)
report_layout.addRow("导出格式:", self.format_combo)

report_group.setLayout(report_layout)
settings_layout.addWidget(report_group)

# 包含内容设置
content_group = QGroupBox("报告包含内容")
content_layout = QVBoxLayout()

self.include_spectra = QCheckBox("光谱图")
self.include_spectra.setChecked(True)

self.include_refractive = QCheckBox("折射率曲线")
self.include_refractive.setChecked(True)

self.include_thickness = QCheckBox("厚度计算结果")
self.include_thickness.setChecked(True)

self.include_comparison = QCheckBox("厚度对比图")
self.include_comparison.setChecked(True)

content_layout.addWidget(self.include_spectra)
content_layout.addWidget(self.include_refractive)
content_layout.addWidget(self.include_thickness)
content_layout.addWidget(self.include_comparison)

content_group.setLayout(content_layout)
settings_layout.addWidget(content_group)

# 导出按钮
export_btn = QPushButton("导出报告")
export_btn.clicked.connect(self.export_report)
settings_layout.addWidget(export_btn)

# 状态信息
self.status_text = QTextEdit()
self.status_text.setReadOnly(True)
self.status_text.setMinimumHeight(100)
settings_layout.addWidget(self.status_text)
```

```
splitter.addWidget(settings_panel)

# 下半部分：报告预览
preview_panel = QWidget()
preview_layout = QVBoxLayout(preview_panel)

preview_label = QLabel("报告内容预览")
preview_label.setFont(QFont("KaiTi", 12, QFont.Bold))
preview_layout.addWidget(preview_label)

self.preview_text = QTextEdit()
self.preview_text.setReadOnly(True)
preview_layout.addWidget(self.preview_text)

splitter.addWidget(preview_panel)

# 设置分割器比例
splitter.setSizes([400, 300])

main_layout.addWidget(splitter)

def update_data(self, data):
    """ 更新数据 """
    self.data = data
    self.update_preview()

def update_results(self, results):
    """ 更新结果 """
    self.results = results
    self.update_preview()

def update_preview(self):
    """ 更新预览 """
    preview = ""

    preview += f"标题: {self.report_title.toPlainText()}\n\n"

    if self.report_author.toPlainText():
        preview += f"作者: {self.report_author.toPlainText()}\n\n"

    preview += f"生       成       日       期       :\n{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n\n"

    if self.data:
        preview += "数据集信息:\n"
        for angle, data in self.data.items():
```

```

        preview += f"- 入射角 {angle}°: {len(data)} 个数据点，波数范围
{min(data['wavenumber']):.2f} - {max(data['wavenumber']):.2f} cm-1\n"
        preview += "\n"

    if self.results and 'individual_results' in self.results:
        preview += "各角度单独分析结果:\n"
        for angle, res in self.results['individual_results'].items():
            preview += f"- 入射角 {angle}°: 厚度 {res['thickness']:.2f} nm, RMSE
{res['rmse']:.4f}, R2 {res['r_squared']:.4f}\n"
        preview += "\n"

    if self.results and 'joint_result' in self.results:
        preview += "联合拟合结果:\n"
        res = self.results['joint_result']
        preview += f"- 最优厚度: {res['joint_thickness']:.2f} nm\n"
        preview += f"- RMSE: {res['rmse']:.4f}\n"
        preview += f"- R2: {res['r_squared']:.4f}\n"
        preview += "\n"

    if self.report_notes.toPlainText():
        preview += f"备注:\n{self.report_notes.toPlainText()}\n"

    self.preview_text.setPlainText(preview)

def generate_temp_plots(self):
    """ 生成临时图表 """
    # 清除之前的临时文件
    for path in self.temp_plot_paths:
        if os.path.exists(path):
            try:
                os.remove(path)
            except:
                pass
    self.temp_plot_paths = []

    temp_plots = []

    # 创建临时目录
    temp_dir = os.path.join(os.path.dirname(os.path.abspath(__file__)), "temp_plots")
    if not os.path.exists(temp_dir):
        os.makedirs(temp_dir)

    # 光谱图
    if self.include_spectra.isChecked() and self.data:
        fig, ax = plt.subplots(figsize=(8, 6), dpi=100)
        colors = ['blue', 'red', 'green', 'purple', 'orange']
        color_idx = 0

```

```

        for angle, data in self.data.items():
            color = colors[color_idx % len(colors)]
            color_idx += 1

            ax.plot(
                data['wavenumber'],
                data['smoothed_reflectance'] if 'smoothed_reflectance' in data.columns
                else data['reflectance'],
                label=f'入射角 {angle}°',
                color=color,
                linewidth=2
            )

        ax.set_title('红外干涉光谱', fontproperties='KaiTi', fontsize=12)
        ax.set_xlabel('波数 ($\mathrm{cm^{-1}}$)', fontproperties='KaiTi', fontsize=10)
        ax.set_ylabel('反射率 (%)', fontproperties='KaiTi', fontsize=10)
        ax.legend(prop={'family': ['KaiTi', 'SimHei']})
        ax.grid(True, linestyle='--', alpha=0.7)
        plt.tight_layout()

        plot_path = os.path.join(temp_dir,
f"spectra_{datetime.datetime.now().strftime('%Y%m%d%H%M%S')}.png")
        fig.savefig(plot_path)
        temp_plots.append(plot_path)
        self.temp_plot_paths.append(plot_path)
        plt.close(fig)

# 厚度对比图
if self.include_comparison.isChecked() and self.results and 'joint_result' in self.results:
    joint_result = self.results['joint_result']
    individual_results = self.results['individual_results']

    if individual_results and len(individual_results) >= 2:
        fig, ax = plt.subplots(figsize=(8, 6), dpi=100)

        angles = list(individual_results.keys())
        angles_numeric = [float(angle) for angle in angles]
        individual_thicknesses = [res['thickness'] for res in individual_results.values()]
        joint_thickness = [joint_result['joint_thickness']] * len(angles)

        x = np.arange(len(angles))
        width = 0.35

        ax.bar(x - width / 2, individual_thicknesses, width, label='单独计算',
color='blue')
        ax.bar(x + width / 2, joint_thickness, width, label='联合拟合', color='red')

```

```
        ax.set_xticks(x)
        ax.set_xticklabels(angles)
        ax.set_title('厚度计算结果对比', fontproperties='KaiTi', fontsize=12)
        ax.set_xlabel('入射角 (°)', fontproperties='KaiTi', fontsize=10)
        ax.set_ylabel('厚度 (nm)', fontproperties='KaiTi', fontsize=10)
        ax.legend(prop={'family': ['KaiTi', 'SimHei']})
        ax.grid(True, linestyle='--', alpha=0.7)
        plt.tight_layout()

    plot_path = os.path.join(temp_dir,
                            f"thickness_comparison_{datetime.datetime.now().strftime('%Y%m%d%H%M%S')}.png")
    fig.savefig(plot_path)
    temp_plots.append(plot_path)
    self.temp_plot_paths.append(plot_path)
    plt.close(fig)

    return temp_plots

def export_report(self):
    """ 导出报告 """
    if not self.data:
        QMessageBox.warning(self, "警告", "没有可导出的数据, 请先处理数据")
        return

    # 获取导出路径
    format = self.format_combo.currentText()
    filter = f"{format} files (*.{format.lower()})" if format == "PDF" else "Word files (*.docx)"
    file_path, _ = QFileDialog.getSaveFileName(self, "保存报告", f"碳化硅晶圆厚度分析报告.{format.lower()}", filter)

    if not file_path:
        return

    # 更新状态
    self.status_text.clear()
    self.status_text.append("正在准备报告...")

    # 生成临时图表
    self.status_text.append("生成图表...")
    temp_plots = self.generate_temp_plots()

    # 导出报告
    self.status_text.append(f"导出{format}报告...")

    success = False
```

```
msg = ""

if format == "PDF":
    success, msg = ReportGenerator.generate_pdf_report(file_path, self.data,
self.results, temp_plots)
else: # Word
    success, msg = ReportGenerator.generate_word_report(file_path, self.data,
self.results, temp_plots)

# 清理临时文件
for path in self.temp_plot_paths:
    if os.path.exists(path):
        try:
            os.remove(path)
        except:
            self.status_text.append(f"警告：无法删除临时文件 {path}")

self.temp_plot_paths = []

# 显示结果
if success:
    self.status_text.append(f"报告已成功导出至: {file_path}")
    QMessageBox.information(self, "成功", f"{format}报告已成功导出")
else:
    self.status_text.append(f"导出失败: {msg}")
    QMessageBox.error(self, "失败", f"导出报告时出错: {msg}")


class MainWindow(QMainWindow):
    """ 主窗口 """

    def __init__(self):
        super().__init__()
        self.init_ui()

    def init_ui(self):

        # ====== 添加图标代码 开始 ======
        from PyQt5.QtGui import QIcon
        import os

        # 图标文件路径
        icon_path = r"C:\Users\11579\Desktop\flame\碳化硅.png"

        # 检查文件是否存在，避免报错
        if os.path.exists(icon_path):
            self.setWindowIcon(QIcon(icon_path))
```

```
else:  
    print(f"警告：图标文件不存在 - {icon_path}")  
# ====== 添加图标代码 结束 ======  
  
# 设置窗口标题和大小  
self.setWindowTitle("碳化硅晶圆外延层厚度分析系统")  
self.setGeometry(100, 100, 1200, 800)  
  
# 设置全局字体  
font = QFont("KaiTi", 10)  
self.setFont(font)  
  
# 创建中心部件和标签页  
central_widget = QWidget()  
self.setCentralWidget(central_widget)  
  
main_layout = QVBoxLayout(central_widget)  
  
# 创建标签页  
self.tabs = QTabWidget()  
self.tabs.setTabBar(QTabBar(self))  
self.tabs.setTabPosition(QTabWidget.North)  
  
# 创建各个功能框  
self.first_tab = FirstTab()  
self.second_tab = SecondTab()  
self.third_tab = ThirdTab()  
self.fourth_tab = FourthTab()  
self.fifth_tab = FifthTab()  
  
# 添加标签页  
self.tabs.addTab(self.first_tab, "光谱预处理与可视化")  
self.tabs.addTab(self.second_tab, "折射率建模与干涉模拟")  
self.tabs.addTab(self.third_tab, "碳化硅外延层厚度计算")  
self.tabs.addTab(self.fourth_tab, "多角度联合拟合优化")  
self.tabs.addTab(self.fifth_tab, "分析结果与报告导出")  
  
# 设置标签页字体  
tab_font = QFont("KaiTi", 11)  
for i in range(self.tabs.count()):  
    self.tabs.tabBar().setFont(tab_font)  
  
main_layout.addWidget(self.tabs)  
  
# 连接信号和槽  
self.first_tab.data_updated.connect(self.second_tab.update_data)  
self.first_tab.data_updated.connect(self.third_tab.update_data)
```

```
self.first_tab.data_updated.connect(self.fourth_tab.update_data)
self.first_tab.data_updated.connect(self.fifth_tab.update_data)

self.third_tab.thickness_results_updated.connect(self.fourth_tab.update_individual_results)
self.third_tab.thickness_results_updated.connect(self.update_fifth_tab_results)

self.fourth_tab.joint_results_updated.connect(self.fifth_tab.update_results)

# 状态栏
self.statusBar().showMessage("就绪")

def update_fifth_tab_results(self, individual_results):
    """ 更新第五个标签页的结果 """
    current_results = self.fifth_tab.results
    current_results['individual_results'] = individual_results
    self.fifth_tab.update_results(current_results)

if __name__ == "__main__":
    app = QApplication(sys.argv)

    # 设置全局字体
    font = QFont("KaiTi", 10)
    app.setFont(font)

    window = MainWindow()
    window.show()

    sys.exit(app.exec_())
```