当然可以！我来帮你写一个**清晰、结构合理、从原始数据处理到 Group Lasso 特征选择**的完整流程，专门适配你的数据结构（**长格式 + 存在缺失 Stage**），并实现原文所需的分组设备选择。

✅ 目标流程：

1. 原始数据（长格式） → 宽表（每个 Sample 一行）

2. 用 __MISSING__ 填补缺失的 Stage → 保留所有样本

3. One-hot 编码所有 stage 下的设备

4. 构建 group 信息（每个 Stage 作为一个 group）

5. 用 Group Lasso 选择关键 Stage

✅🔁 完整可运行代码（结构良好）

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from group_lasso import GroupLasso
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# ========== Step 1: Load long-format data ==========
# 假设你有一个 DataFrame df，结构如下：
# | SampleID | Stage     | Equipment | Yield |
# |----------|-----------|-----------|-------|

# ========== Step 2: Pivot to wide format ==========
df_wide = df.pivot_table(index=['SampleID', 'Yield'], columns='Stage',
values='Equipment', aggfunc='first').reset_index()

# ========== Step 3: Fill missing stages with "__MISSING__" ==========
X_raw = df_wide.drop(columns=['SampleID', 'Yield'])
X_filled = X_raw.fillna('__MISSING__')  # 关键步骤！
y = df_wide['Yield'].values

# ========== Step 4: One-hot encode equipment ==========
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_encoded = encoder.fit_transform(X_filled)

feature_names = encoder.get_feature_names_out(X_filled.columns)

# ========== Step 5: Build group index per stage ==========
# 每个 one-hot feature 的名字形如 "stage_1_EQP_A"
group_names = [name.split('_')[0] for name in feature_names]
unique_stages = np.unique(group_names)
```

```
group_map = {stage: i for i, stage in enumerate(unique_stages)}
groups = np.array([group_map[g] for g in group_names])

# ========== Step 6: Fit Group Lasso ==========
gl = GroupLasso(
    groups=groups,
    group_reg=0.05,         # 可调：越大越稀疏
    l1_reg=0,
    frobenius_lipschitz=True,
    scale_reg="inverse_group_size",
    supress_warning=True,
    n_iter=1000,
    tol=1e-3,
    fit_intercept=True,
    random_state=42
)

gl.fit(X_encoded, y)

# ========== Step 7: 选中的 Stage ==========
selected_feature_mask = gl.sparsity_mask_
selected_group_indices = np.unique(groups[selected_feature_mask])
selected_stages = [unique_stages[i] for i in selected_group_indices]

print("✅ 选中的关键 Stage 如下：")
print(selected_stages)
```

✅ **输出说明：**

• selected_stages：就是 Group Lasso 自动选择出来的关键阶段；

• 即这些 Stage 下的设备最能解释良率的变化。

✅ **扩展：查看哪些 Stage 的 "__MISSING__" 特征被选中**

这有助于你判断 **是否“跳过该 Stage”本身就是一个影响因素**。

```
missing_feature_indices = [
    i for i, name in enumerate(feature_names)
    if '__MISSING__' in name and selected_feature_mask[i]
]

print("\n✅ 被选中的 '跳过 Stage' (__MISSING__) 特征：")
for i in missing_feature_indices:
    print(f"特征：{feature_names[i]}")
```

✅ **下一步（可选）：**

• 用 selected_stages 去生成路径组合

• 进行三阶建模 + AIC 选择 + Reliance Index 验证

如果你要，我可以马上将这些模块和这一步整合起来，变成一个完整 pipeline。要整合进来吗？

太好了，你提到了两个非常关键的细节，完全对齐原文思路，我们现在来 **精准修正实现**：

✅ **核心修正点**

1. **预测路径良率**的方式改为：

```
best_model_yield = model.coef_.sum()
```

这确实是原文的定义（路径得分是模型系数之和，反映设备组合对良率的正向驱动力）。

2. 对于每一条路径，比如：

```
['stage_1': 'EQP_A', 'stage_2': None, 'stage_3': 'EQP_C']
```

只用那些同时经过 stage_1=EQP_A 且 stage_3=EQP_C 的样本来建模，**忽略 NA 的 stage**，这点你说得非常清楚。

✅ **修正后的代码实现：**

```
from itertools import product
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# 已选的关键 stage（来自 Group Lasso 或 GOGA）
selected_stages = [...]  # 例如 ['stage_1', 'stage_2', 'stage_3']

# 构建每个 stage 的有效设备列表（去除 NaN）
equipments_per_stage = {
    stage: df_pivot[stage].dropna().unique() for stage in selected_stages
}

# 构建所有路径组合
all_paths = list(product(*equipments_per_stage.values()))
print(f"共生成路径数：{len(all_paths)}")

# AIC 函数
def compute_aic(n, mse, k):
    return n * np.log(mse) + 2 * k

# 存储结果
path_results = []

for path in all_paths:
    stage_equip_map = dict(zip(selected_stages, path))

    # 构造有效样本掩码（所有参与 stage 的设备都匹配）
    valid_mask = np.ones(len(df_pivot), dtype=bool)
    active_stages = []  # 只记录当前路径有设备的 stage
    for stage, equip in stage_equip_map.items():
        if pd.isna(equip):  # 跳过 NA 设备（不作为建模变量）
            continue
        stage_match = (df_pivot[stage] == equip)
        valid_mask &= stage_match
        active_stages.append(stage)

    # 样本数量太少就跳过
    if valid_mask.sum() < 10 or len(active_stages) == 0:
        continue

    # 构建二元变量特征（只对 active_stages 做建模）
    X_bin = pd.DataFrame()
    for stage in active_stages:
        equip = stage_equip_map[stage]
        X_bin[f"{stage}={equip}"] = (df_pivot.loc[valid_mask, stage] ==
equip).astype(int)

    y_local = df_pivot.loc[valid_mask, 'Yield']
    best_model_score = None
    best_model_aic = np.inf

    # 尝试不同阶数交互项
    for degree in [1, 2, 3]:
```

```
        poly = PolynomialFeatures(degree=degree, interaction_only=True,
include_bias=False)
        X_poly = poly.fit_transform(X_bin)

        if X_poly.shape[1] >= valid_mask.sum():
            continue  # 特征比样本多，跳过过拟合情形

        model = LinearRegression().fit(X_poly, y_local)
        y_pred = model.predict(X_poly)
        mse = mean_squared_error(y_local, y_pred)
        aic = compute_aic(len(y_local), mse, X_poly.shape[1])

        if aic < best_model_aic:
            best_model_aic = aic
            # ✅ 预测路径得分用 coef_.sum()
            best_model_score = model.coef_.sum()

    path_results.append((stage_equip_map, best_model_score,
best_model_aic))

# 排序
sorted_paths = sorted(path_results, key=lambda x: x[1], reverse=True)

# 输出前 10 路径
print("\n 推荐 Top 10 黄金路径:")
for i, (path_dict, score, aic) in enumerate(sorted_paths[:10], 1):
    print(f"{i}. 路径: {path_dict}, 路径得分: {score:.2f}, AIC: {aic:.2f}")
```

✅ **最终行为特征总结：**

行为	是否实现	说明
路径建模只使用完整匹配样本	✅	所有包含所需设备的样本保留，其余不参与
NA stage 不参与建模	✅	自动跳过
使用三阶交互模型 + AIC 筛选	✅	最优阶数选择
使用 coef_.sum() 作为路径得分	✅	与原文一致的定义

如果你接下来还想添加：

• Reliance Index 验证

• 路径在生产系统中可行性约束（设备状态、可切换性等）

我也可以继续帮你加上去 ✌️

要不要现在就做 Reliance Index？

好，我们现在来实现 **Reliance Index（RIGP）**，这是文章中用于**评估每条路径的可信度**的指标。

✅ 什么是 Reliance Index？

原文中定义的 **RIGP (Reliance Index of Golden Path)**：

衡量路径预测结果是否“可靠”，也就是路径中每个设备和良率的相关性与预测结果是否一致。

✅ Reliance Index 计算公式（来自原文）：

对每条路径 p：

$$\text{RIGP}(p) = 1 - \frac{|\hat{Y}_p - \bar{r}_p|}{\max(Y) - \min(Y)}$$

其中：

- $\hat{Y}_p$：路径预测得分（我们已计算为 model.coef_.sum()）
- $\bar{r}_p$：路径中所有设备与良率之间的平均 Pearson 相关系数
- Y：所有样本的良率（用于归一化）

✅ 实现步骤：

🔢 Reliance Index 核心函数：

```
import numpy as np

def compute_reliance_index(path_dict, path_score, df_pivot,
all_yield_col='Yield'):
    corr_list = []
    for stage, equip in path_dict.items():
        if pd.isna(equip):
            continue  # NA 不计入

        # 构造设备对应的 binary indicator
        indicator = (df_pivot[stage] == equip).astype(int)

        if indicator.sum() < 5:
            continue  # 太稀疏的变量不计算相关

        y = df_pivot[all_yield_col]
        corr = np.corrcoef(indicator, y)[0, 1]

        if np.isnan(corr):
            continue
        corr_list.append(abs(corr))  # 取绝对值

    if not corr_list:
        return np.nan

    avg_corr = np.mean(corr_list)
    y_max = df_pivot[all_yield_col].max()
    y_min = df_pivot[all_yield_col].min()

    rigp = 1 - (abs(path_score - avg_corr)) / (y_max - y_min)
    return rigp
```

📊 给之前 Top 路径加上 RIGP：

```
print("\nTop 10 路径的 Reliance Index (RIGP):")
for i, (path_dict, score, aic) in enumerate(sorted_paths[:10], 1):
    rigp = compute_reliance_index(path_dict, score, df_pivot)
    print(f"{i}. 路径: {path_dict}")
    print(f"   路径得分: {score:.3f}, AIC: {aic:.1f}, RIGP: {rigp:.4f}")
```

✅ **可选：加入排序权重（路径得分 × RIGP）提升筛选精度**

```
ranked_by_combo = sorted(
    [(p, score, aic, compute_reliance_index(p, score, df_pivot)) for p,
score, aic in path_results],
    key=lambda x: x[1] * (x[3] if x[3] is not None else 0),
    reverse=True
)

print("\n按 路径得分 × RIGP 排序的前 10 黄金路径:")
for i, (path_dict, score, aic, rigp) in enumerate(ranked_by_combo[:10], 1):
    print(f"{i}. 路径: {path_dict}")
    print(f"    路径得分: {score:.2f}, RIGP: {rigp:.4f}, AIC: {aic:.1f}")
```

✅ **总结一下我们已经完成的内容：**

模块	是否完成	内容说明
Group Lasso/GOGA	✅	筛选关键 Stage
路径枚举 + 三阶建模	✅	拟合最优良率模型
使用 coef_.sum() 表示路径得分	✅	与原文定义一致
AIC 模型选择	✅	自动选最优阶数
Reliance Index (RIGP)	✅	量化路径可信度
综合排序	✅	可选按得分 × RIGP 排序

接下来你如果想要：

• 做结果可视化（热力图、路径图、分布图等）

• 或者部署成自动推荐系统/应用

我也可以继续帮你衔接 👇

你需要现在保存结果为 DataFrame 或继续分析吗？

太好了！我们现在把整个流程串起来，做一个**完整的 Golden Path Search Pipeline**，从：

原始长格式数据 → Group Lasso 选出关键 stage → 构建路径组合 → 三阶交互建模 + AIC 筛选 → Reliance Index 计算 → 排序推荐黄金路径

✅ 输入数据格式（你已有的）：

SampleID Stage Equipment Yield

✅🧠 整体步骤结构：

1. 数据转换（长 → 宽）

2. Group Lasso 选出关键 Stage

3. 构建路径组合（设备 × stage）

4. 每条路径建模（仅对可用 stage，1~3 阶交互）

5. 用 AIC 选最优模型，得出路径得分（系数和）

6. 计算每条路径的 Reliance Index (RIGP)

7. 输出排序结果（路径、得分、AIC、RIGP）

✅ 完整 Pipeline 代码实现

```
import pandas as pd
import numpy as np
from itertools import product
from sklearn.preprocessing import OneHotEncoder, PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from group_lasso import GroupLasso

# ========================== Step 1: 数据转换 ==========================
# 输入 df: 长格式
# ['SampleID', 'Stage', 'Equipment', 'Yield']
```

```
# Pivot 宽表
df_wide = df.pivot_table(index=['SampleID', 'Yield'], columns='Stage',
values='Equipment', aggfunc='first').reset_index()
X_raw = df_wide.drop(columns=['SampleID', 'Yield'])
X_filled = X_raw.fillna('__MISSING__')
y = df_wide['Yield'].values

# ========================== Step 2: Group Lasso 筛选关键 Stage
==========================
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_encoded = encoder.fit_transform(X_filled)
feature_names = encoder.get_feature_names_out(X_filled.columns)
group_names = [name.split('_')[0] for name in feature_names]
unique_stages = np.unique(group_names)
group_map = {stage: i for i, stage in enumerate(unique_stages)}
groups = np.array([group_map[g] for g in group_names])

gl = GroupLasso(
    groups=groups,
    group_reg=0.05,
    l1_reg=0,
    frobenius_lipschitz=True,
    scale_reg="inverse_group_size",
    supress_warning=True,
    n_iter=1000,
    tol=1e-3,
    fit_intercept=True,
    random_state=42
)
gl.fit(X_encoded, y)

selected_mask = gl.sparsity_mask_
selected_group_indices = np.unique(groups[selected_mask])
selected_stages = [unique_stages[i] for i in selected_group_indices]
print("✅ 选中的关键 Stage:", selected_stages)

# ========================== Step 3: 构建路径组合 ==========================
equipments_per_stage = {
    stage: df_wide[stage].dropna().unique() for stage in selected_stages
}
all_paths = list(product(*equipments_per_stage.values()))
print(f"路径组合数: {len(all_paths)}")

# ========================== Step 4: 建模 + AIC 选择
==========================
def compute_aic(n, mse, k):
    return n * np.log(mse) + 2 * k

path_results = []

for path in all_paths:
    path_map = dict(zip(selected_stages, path))

    # 获取样本匹配此路径（忽略 NA 的 stage）
    active_stages = []
    mask = np.ones(len(df_wide), dtype=bool)
    for stage, equip in path_map.items():
        if pd.isna(equip):
```

```
            continue
        match = df_wide[stage] == equip
        mask &= match
        active_stages.append(stage)

    if mask.sum() < 10 or not active_stages:
        continue

    # 二元特征
    X_bin = pd.DataFrame()
    for stage in active_stages:
        equip = path_map[stage]
        X_bin[f"{stage}={equip}"] = (df_wide.loc[mask, stage] ==
equip).astype(int)

    y_sub = df_wide.loc[mask, 'Yield']
    best_score = None
    best_aic = np.inf

    for deg in [1, 2, 3]:
        poly = PolynomialFeatures(degree=deg, interaction_only=True,
include_bias=False)
        X_poly = poly.fit_transform(X_bin)

        if X_poly.shape[1] >= len(y_sub): continue  # 防过拟合

        model = LinearRegression().fit(X_poly, y_sub)
        mse = mean_squared_error(y_sub, model.predict(X_poly))
        aic = compute_aic(len(y_sub), mse, X_poly.shape[1])

        if aic < best_aic:
            best_aic = aic
            best_score = model.coef_.sum()

    path_results.append((path_map, best_score, best_aic))

# 排序
sorted_paths = sorted(path_results, key=lambda x: x[1], reverse=True)

# ========================== Step 5: 计算 RIGP ==========================
def compute_reliance_index(path_map, path_score, df, yield_col='Yield'):
    corr_list = []
    for stage, equip in path_map.items():
        if pd.isna(equip): continue
        indicator = (df[stage] == equip).astype(int)
        if indicator.sum() < 5: continue
        corr = np.corrcoef(indicator, df[yield_col])[0, 1]
        if not np.isnan(corr):
            corr_list.append(abs(corr))
    if not corr_list:
        return np.nan
    avg_corr = np.mean(corr_list)
    y_max, y_min = df[yield_col].max(), df[yield_col].min()
    rigp = 1 - (abs(path_score - avg_corr)) / (y_max - y_min)
    return rigp

final_results = []
for path_map, score, aic in sorted_paths[:50]:  # 只取前 50 条做 RIGP
    rigp = compute_reliance_index(path_map, score, df_wide)
```

```
    final_results.append((path_map, score, aic, rigp))

# ========================== Step 6: 输出 Top 10 黄金路径
==========================
print("\n🏆 Top 10 黄金路径（按路径得分×RIGP）:")
ranked = sorted(final_results, key=lambda x: x[1] * (x[3] if x[3] is not
None else 0), reverse=True)
for i, (path, score, aic, rigp) in enumerate(ranked[:10], 1):
    print(f"{i}. 路径: {path}")
    print(f"   路径得分: {score:.2f}, AIC: {aic:.2f}, RIGP: {rigp:.4f}")
```

✅ **输出结果结构：**

每条路径包含：

• 所选设备（路径）

• 路径得分（sum of model coef）

• 模型 AIC

• Reliance Index (RIGP)

✅ **你可以做的扩展：**

• 把 final_results 转为 DataFrame 保存

• 可视化 top 路径热力图 / 交互图

• 接入生产排程系统推荐高良率路径

如果你想，我也可以帮你画个路径可视化图，或者导出 CSV 表格格式。要继续做这些吗？

好的！你提供的是一个完整的算法思想，叫 **Ensemble-SISPLS（集成式确定独立筛选 + 偏最小二乘）变量选择方法**，用于高维小样本近红外光谱数据的建模。

下面我给出一个基于 Python 的代码框架实现，使用的是：

• PLSRegression 来实现偏最小二乘回归

• BIC 用于模型优选

• SIS 用边际相关性（比如皮尔逊相关）排序变量

• 集成思想通过 Bootstrap + 投票 + 网格搜索

✅ 主要依赖：

```
pip install numpy pandas scikit-learn
```

🧠 Ensemble-SISPLS 实现代码（简化框架版）

```
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.cross_decomposition import PLSRegression
from sklearn.utils import resample
from sklearn.metrics import mean_squared_error
from scipy.stats import pearsonr
import warnings

warnings.filterwarnings("ignore")

# -------------------------
# Step 1: SIS Variable Pre-selection
# -------------------------
def SIS(X, y, top_k=None):
    corrs = np.array([abs(pearsonr(X[:, i], y)[0]) for i in
range(X.shape[1])])
    if top_k is None:
        top_k = X.shape[0]  # default to number of samples
    selected_idx = np.argsort(corrs)[-top_k:]
    return selected_idx

# -------------------------
# Step 2: Forward selection with BIC (SPLS)
# -------------------------
def compute_bic(y_true, y_pred, num_params):
    n = len(y_true)
```

```
    resid = y_true - y_pred
    rss = np.sum(resid ** 2)
    bic = n * np.log(rss / n) + num_params * np.log(n)
    return bic

def SIS_SPLS(X, y, max_components=10):
    selected_features = []
    candidate_features = list(range(X.shape[1]))
    best_bic = float('inf')
    best_model = None
    best_features = []

    for _ in range(len(candidate_features)):
        bic_list = []
        temp_models = []
        for feat in candidate_features:
            temp_feats = selected_features + [feat]
            pls = PLSRegression(n_components=min(max_components,
len(temp_feats)))
            pls.fit(X[:, temp_feats], y)
            y_pred = pls.predict(X[:, temp_feats]).ravel()
            bic = compute_bic(y, y_pred, len(temp_feats))
            bic_list.append(bic)
            temp_models.append((pls, temp_feats))

        best_idx = np.argmin(bic_list)
        if bic_list[best_idx] < best_bic:
            best_bic = bic_list[best_idx]
            best_model = temp_models[best_idx][0]
            selected_features = temp_models[best_idx][1]
            candidate_features.remove(selected_features[-1])
            best_features = selected_features.copy()
        else:
            break

    return best_features, best_model

# ------------------------
# Step 3: Ensemble SIS-SPLS with Voting
# ------------------------
def ensemble_SIS_SPLS(X, y, n_iter=100, threshold=0.5, max_components=10):
    n_samples = X.shape[0]
    feature_votes = np.zeros(X.shape[1])

    for i in range(n_iter):
        X_resampled, y_resampled = resample(X, y)
        sis_idx = SIS(X_resampled, y_resampled, top_k=n_samples)
        X_sis = X_resampled[:, sis_idx]

        selected_idx, _ = SIS_SPLS(X_sis, y_resampled,
max_components=max_components)
        selected_original_idx = sis_idx[selected_idx]
        feature_votes[selected_original_idx] += 1

    # 通过投票确定最终变量
    selected_features = np.where(feature_votes / n_iter >= threshold)[0]
    return selected_features

# ------------------------
# Step 4: Evaluate model via cross-validation
# ------------------------
```

```
def evaluate_model(X, y, features, n_components=5, cv=5):
    kf = KFold(n_splits=cv, shuffle=True, random_state=42)
    rmses = []

    for train_index, test_index in kf.split(X):
        pls = PLSRegression(n_components=n_components)
        pls.fit(X[train_index][:, features], y[train_index])
        y_pred = pls.predict(X[test_index][:, features])
        rmse = mean_squared_error(y[test_index], y_pred, squared=False)
        rmses.append(rmse)

    return np.mean(rmses)

# ------------------------
# Step 5: Grid Search for Optimal Threshold and n_components
# ------------------------
def grid_search(X, y, thresholds=[0.3, 0.4, 0.5, 0.6],
n_components_list=[2, 5, 10]):
    best_score = float('inf')
    best_params = None
    best_features = None

    for t in thresholds:
        for n_comp in n_components_list:
            features = ensemble_SIS_SPLS(X, y, threshold=t,
max_components=n_comp)
            if len(features) == 0:
                continue
            score = evaluate_model(X, y, features, n_components=n_comp)
            if score < best_score:
                best_score = score
                best_params = {'threshold': t, 'n_components': n_comp}
                best_features = features

    return best_params, best_features, best_score

# ------------------------
# Example usage:
# ------------------------
# 假设你有预处理后的数据 X（样本数 × 变量数），y 是目标变量
# X = np.loadtxt('nir_X.csv', delimiter=',')
# y = np.loadtxt('nir_y.csv', delimiter=',')

# params, final_features, rmse = grid_search(X, y)
# print("Best Params:", params)
# print("Selected Features:", final_features)
# print("Cross-Validated RMSE:", rmse)
```

🧪 **示例说明：**

• 你可以把 Corn 数据或 当归 数据代入 X, y，看哪个变量最终被选择。

• 支持调整：

• 阈值（threshold）

• 潜变量个数（n_components）

• 重采样次数（n_iter）

如果你需要我：

• 加载并预处理具体的 CSV 数据

• 增加可视化（比如变量重要性、RMSEP 对比图）

• 加入 UVE-PLS 以做三种方法对比

• 或者部署成脚本/网页小工具

都可以继续跟我说！

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
from scipy.signal import savgol_filter
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from skorch import NeuralNetRegressor
from skorch.callbacks import EarlyStopping
```

```
from skopt import BayesSearchCV
from skopt.space import Integer, Real
import joblib

# --------------------
# Data Preprocessing
# --------------------
def msc(input_data):
    mean_spectrum = np.mean(input_data, axis=0)
    corrected = np.zeros_like(input_data)
    for i in range(input_data.shape[0]):
        fit = np.polyfit(mean_spectrum, input_data[i, :], 1, full=True)
        corrected[i, :] = (input_data[i, :] - fit[0][1]) / fit[0][0]
    return corrected

def snv(input_data):
    return (input_data - np.mean(input_data, axis=1, keepdims=True)) / \
           np.std(input_data, axis=1, keepdims=True)

def first_derivative(input_data, window_length=11, polyorder=2):
    return savgol_filter(input_data, window_length=window_length, polyorder=polyorder,
deriv=1)

def preprocess_combined(data):
    data_msc = msc(data)
    data_snv = snv(data_msc)
    data_1d = first_derivative(data_snv)
    return data_1d
```

```
# --------------------
# BEST-1DConvNet Model
# --------------------
class Best1DCNN(nn.Module):
  def __init__(self, num_conv_layers=1, num_filters=16, kernel_size=11, stride=1, num_fc_layers=1):
    super(Best1DCNN, self).__init__()
    layers = []
    in_channels = 1
    for _ in range(num_conv_layers):
      layers.append(nn.Conv1d(in_channels, num_filters, kernel_size, stride=stride))
      layers.append(nn.ReLU())
      layers.append(nn.AdaptiveMaxPool1d(1))
      in_channels = num_filters
    self.conv = nn.Sequential(*layers)

    fc_layers = []
    for _ in range(num_fc_layers - 1):
      fc_layers.append(nn.Linear(num_filters, num_filters))
      fc_layers.append(nn.ReLU())
    fc_layers.append(nn.Linear(num_filters, 1))
    self.fc = nn.Sequential(*fc_layers)

  def forward(self, x):
    x = self.conv(x)
    x = x.view(x.size(0), -1)
    x = self.fc(x)
```

```
        return x


# --------------------
# Main Training Flow
# --------------------
# Load your dataset (replace this with real data loading)
# X_raw: shape (samples, features); y: shape (samples, )
# Example:
# X_raw = np.loadtxt('nir_X.csv', delimiter=',')
# y = np.loadtxt('nir_y.csv', delimiter=',')

# For demonstration, we use dummy data
np.random.seed(42)
X_raw = np.random.rand(100, 401)  # 100 samples, 401 wavelengths
y = np.random.rand(100)

X_processed = preprocess_combined(X_raw)
X_processed = X_processed[:, np.newaxis, :]  # Add channel dimension for CNN

X_train, X_test, y_train, y_test = train_test_split(X_processed, y, test_size=0.3,
random_state=42)

# Define the skorch model
net = NeuralNetRegressor(
    Best1DCNN,
    max_epochs=200,
    optimizer=torch.optim.Adam,
    callbacks=[EarlyStopping(patience=20)],
```

```
    device='cuda' if torch.cuda.is_available() else 'cpu'
)

# Define Bayesian search space
search_space = {
    'module__num_conv_layers': Integer(1, 5),
    'module__num_filters': Integer(1, 28),
    'module__kernel_size': Integer(1, 51),
    'module__stride': Integer(1, 10),
    'lr': Real(1e-6, 1e-1, prior='log-uniform'),
    'module__num_fc_layers': Integer(1, 5),
}

opt = BayesSearchCV(
    net,
    search_spaces=search_space,
    n_iter=25,
    cv=3,
    scoring='neg_root_mean_squared_error',
    n_jobs=-1,
    random_state=42
)

opt.fit(X_train.astype(np.float32), y_train.astype(np.float32))

# --------------------
# Evaluation
# --------------------
```

```
y_pred = opt.predict(X_test.astype(np.float32))
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print("Best Parameters:", opt.best_params_)
print("Test RMSE:", rmse)
print("Test R2:", r2)

# --------------------
# Save the best model
# --------------------
joblib.dump(opt.best_estimator_, 'best_1dcnn_model.pkl')

# To load the model later:
# model = joblib.load('best_1dcnn_model.pkl')
```