你这个任务非常有代表性：**少量真实数据 + 大量低质量仿真数据**，是现在物理建模结合机器学习中很典型的问题。

我来分两部分给你梳理解决方案和代码草稿：

🧩 Part 1：PLS 及其变种的代码实现（基于 10 条真实光谱）

你目前是基于 PLSRegression 做回归，R^2 达到 0.85，很不错了。要进一步尝试提升模型，我们可以考虑以下几种 PLS 的变种：

✅ 你可以尝试的 PLS 变种包括：

方法名	描述
PLSRegression（普通 PLS）	sklearn 提供，baseline
Kernel PLS	非线性扩展，映射到高维特征空间
Sparse PLS	引入稀疏性，做变量选择（适合你 550 维高维特征）
Robust PLS	对异常值更稳健
NIPALS PLS	原始 PLS 算法实现，用于小样本场景

✅ PLS / Kernel PLS 代码示例（真实光谱数据）

```
import numpy as np
from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler

# 假设你已有: X_real (10x550), y_real (10x1)
X_real = ...  # shape: (10, 550)
y_real = ...  # shape: (10, )

# 标准化
scaler_X = StandardScaler()
scaler_y = StandardScaler()
X_scaled = scaler_X.fit_transform(X_real)
y_scaled = scaler_y.fit_transform(y_real.reshape(-1, 1)).ravel()
```

```
# 1. 标准 PLS
pls = PLSRegression(n_components=5)
pls.fit(X_scaled, y_scaled)
y_pred = pls.predict(X_scaled)
print("R² (PLS):", r2_score(y_scaled, y_pred))

# 2. Kernel PLS (用 RBF 核)
from sklearn.kernel_ridge import KernelRidge
kr = KernelRidge(kernel='rbf', alpha=1.0)
kr.fit(X_scaled, y_scaled)
y_kr = kr.predict(X_scaled)
print("R² (KernelRidge):", r2_score(y_scaled, y_kr))

# 3. Sparse PLS (用 `sparsepls` 第三方库, optional)
# pip install sparse-pls
from sparse_pls import SPLS
spls = SPLS(n_components=5, eta=0.7)  # eta 控制稀疏程度
spls.fit(X_scaled, y_scaled)
y_spls = spls.predict(X_scaled)
print("R² (Sparse PLS):", r2_score(y_scaled, y_spls))
```

🧩 Part 2：波段筛选 + 仿真数据形态变化分析

你的想法已经很对路了：用真实数据做**重要波段筛选**，然后观察仿真数据在这些波段上的**形态-值-响应关系（spectrum to CG mapping）**。

✅ Step 1：从真实数据中选取重要波段

你可以用下面几种方式来做特征选择：

- **基于 PLS 权重的变量重要性**
- **基于树模型的特征重要性（比如 RandomForest）**
- **基于相关系数（简单）**

```
# 方式 1：PLS 回归系数的重要性排序
pls.fit(X_scaled, y_scaled)
importance = np.abs(pls.coef_[:, 0])  # 每个波段的重要性
top_idx = np.argsort(importance)[-50:]  # 选前 50 个重要波段
```

```
print("Top 50 bands:", top_idx)
```

✅ Step 2：观察仿真数据在这些波段的变化趋势

对于你这 6000 条仿真数据：

1. 从中取出上面这 50 个波段的光谱子集；

2. 观察这 50 维的变化对 CG 值的影响，可以用线性回归/SHAP/偏导等方式做 sensitivity 分析。

示例：偏导数（数值梯度）近似光谱形态对 CG 值的影响

```
# 假设 X_sim 是 shape (6000, 550), y_sim 是仿真的 CG 值
X_sim_selected = X_sim[:, top_idx]

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_sim_selected, y_sim)
coef = lr.coef_

# 可视化每个波段对 CG 值的影响程度
import matplotlib.pyplot as plt

plt.figure()
plt.plot(range(50), coef)
plt.title("Sensitivity of CG to spectral changes at 50 important bands")
plt.xlabel("Selected Band Index")
plt.ylabel("Partial Derivative ~ ∂CG/∂Band")
plt.grid(True)
plt.show()
```

你甚至可以做成一个小的热图，比较：

• 每个仿真样本 vs 每个波段变化时 CG 的响应趋势

✅ 总结

你现在要做的事	方法/建议
PLS 的增强版尝试	用 kernel PLS / sparse PLS 可以尝试突破 0.85 的瓶颈
波段选择	PLS 权重 + 再训练模型验证 + 可视化
仿真数据的借用	先在“高重要性波段”上做 sensitivity mapping，之后再挑选“波段形态类似”的仿真数据用于训练

我可以继续帮你：

1. 生成所有 PLS 变种的代码模板；

2. 可视化 + 动态分析光谱对 CG 影响；

3. 设计仿真数据的筛选策略（基于选择波段的空间距离、相关性等）。

你要我接下来先给哪个？或者你这边有 X/y 的例子我可以代跑分析也行。

太好了，聚焦到仿真数据和真实数据在**重要波段的变化趋势**，你其实是在尝试做一种：

✳️ **跨域光谱响应行为的对比建模**：想找到仿真与真实在相同波段下，**“形态变化 vs CG 响应”的规律是否一致**，从而提炼共性，甚至做校正。

我们来把你的目标拆解成三步，思路先理清：

✅ **总体任务目标：**

在选定的一组关键波段（比如 50 个）中，对比真实数据与仿真数据的变化趋势（波段值 vs CG 响应），寻找它们的共性规律或量化映射差异。

🧩 Step 1：选定重要波段（基于真实数据）

前面讲到，可以用 PLS 或 RandomForest 这种模型，找到前 k=50 个重要波段。我们不赘述，假设你已经得到了这些波段的索引：top_idx。

🧩 Step 2：对比“真实 vs 仿真”的每个波段的响应曲线

这一步是核心：**我们要理解每个波段在真实/仿真数据下，它的变化对 CG 值的影响有多大、有无趋势差异、是否可对齐？**

我们可以采用两种方式做这个量化：

✅ 方法一：单变量回归（每个波段 vs CG）

对于每个波段 b，在真实数据中回归：

CG_{\text{real}} = \beta_b^{real} \cdot X_b + \epsilon

CG_{\text{sim}} = \beta_b^{sim} \cdot X_b + \epsilon

再比较每个波段的 \beta_b^{real} 和 \beta_b^{sim}，比如计算：

```
slope_diff = np.abs(beta_real - beta_sim)
```

也可以可视化对比：

```
plt.plot(range(50), beta_real, label="Real")
plt.plot(range(50), beta_sim, label="Sim")
plt.legend()
plt.title("Band-wise CG sensitivity (slope)")
```

✅ 方法二：计算偏导近似（CG 对波段值的梯度）

可用 ∂CG/∂X_i 近似代表敏感度，方式如下：

```
from sklearn.linear_model import LinearRegression

def get_bandwise_slope(X, y):
    coefs = []
    for i in range(X.shape[1]):
        reg = LinearRegression().fit(X[:, [i]], y)
        coefs.append(reg.coef_[0])
    return np.array(coefs)

# 得到 slope 序列
beta_real = get_bandwise_slope(X_real[:, top_idx], y_real)
beta_sim = get_bandwise_slope(X_sim[:, top_idx], y_sim)
```

✅ 方法三（可选）：局部形态变化 vs CG 增减 的关系

你可以定义“光谱拐点”（如斜率、凹凸性）对 CG 的作用，这就涉及到**波段形态特征构造**，比如：

- 一阶导数（变化趋势）
- 二阶导数（拐点）
- 局部峰/谷/面积 等特征

这样做能更深入分析“仿真 vs 真实”的**形态-响应**差异。

🧩 Step 3：量化仿真 vs 真实 的共性程度

你希望找到一种**量化共性指标**，可以这样做：

方法	描述
✅ Pearson Corr	计算每个波段上：真实 vs 仿真 的 slope 的相关性（越高越一致）
✅ Cosine 相似度	\cos(\beta_{real}, \beta_{sim}) 越接近 1 越说明形态响应相似
✅ 均方误差	越小说明两者斜率（响应程度）越一致
✅ 可视化叠加曲线	查看真实 vs 仿真之间的趋势线是否一致

示例代码：

```
from sklearn.metrics import mean_squared_error
from scipy.spatial.distance import cosine
from scipy.stats import pearsonr

mse = mean_squared_error(beta_real, beta_sim)
cos_sim = 1 - cosine(beta_real, beta_sim)
corr, _ = pearsonr(beta_real, beta_sim)

print("MSE between slopes:", mse)
print("Cosine similarity:", cos_sim)
print("Pearson correlation:", corr)
```

📌 如果你想得更深一步

你可以尝试：

• 对每个仿真样本，计算其光谱形态是否和某个真实样本在这 50 个波段上“梯度响应模式”一致；

• 找出“仿真中哪些样本在这些波段的响应曲线和真实是高度相似的”，再筛选用于训练；

• 利用响应趋势相似性构建 sample-level weight，做“软领域适应”或者权重微调。

✅ 总结下一步建议

步骤	工具	目标
Step 1	PLS 变量重要性排序	找到最关键波段（top_idx）

步骤	工具	目标
Step 2	band-wise slope 对比	真实 vs 仿真 在这些波段上对 CG 的响应敏感度
Step 3	共性度量（corr、cosine、MSE）	找出可以借用仿真数据的波段或样本群体
Step 4（可选）	SHAP/LIME 分析 or 局部形态变化提取	提取高维形态特征来对齐仿真数据响应规律

我可以继续给你写出：

- 👉 上面完整的代码 demo（从选波段 → slope 提取 → 共性分析）
- 👉 对“仿真样本中哪些响应趋势接近真实”的样本筛选代码
- 👉 如果你有数据，我也可以直接跑一份 demo 报告给你看

你想先要哪一个？

很好，我们来做一个更**直观、具体的可视化**，把你说的目标分解成几个清晰的图表，这样你就能从视觉上看到仿真和真实的区别：

✅ **可视化目标：**

展示每个样本在不同波段上的“CG 响应趋势”，然后再**比较仿真和真实样本在这些响应趋势上的差异。**

🧩 **可视化设计方案（推荐）**

1. 热图（Heatmap）

横轴：波段（选定的 50 个）

纵轴：样本（仿真或真实）

颜色：每个样本在某个波段上的 ∂CG/∂band（拟合 slope）

你可以分两张图（仿真、真实），然后并排对比。

2. 折线图（Line Plot）

• 对于每个波段（或关键几个波段），画出：

• 所有真实样本在该波段的光谱值 vs CG 的拟合线

• 所有仿真样本对应的拟合线

• 可视化某个波段在两个领域中的“响应趋势线是否一致”

3. 平均响应对比图

• **对每个波段**，画出：

• 真实样本 slope 平均值 vs 仿真样本 slope 平均值

• 观察趋势差异、是否偏移等

✅ **示例代码：Slope Matrix + 热图 + 对比图**

你只需有：

```
X_real, y_real      # shape: (10, 550)
X_sim, y_sim        # shape: (6000, 550)
top_idx             # list of selected 50 important bands
```

📌 **Step 1: 获取每个样本在每个波段的 slope（局部线性回归）**

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

def compute_sample_slopes(X, y, band_idx):
    n_samples = X.shape[0]
    slopes = np.zeros((n_samples, len(band_idx)))

    for i, idx in enumerate(band_idx):
        for j in range(n_samples):
            X_band = X[:, idx].reshape(-1, 1)
            model = LinearRegression().fit(X_band, y)
            slopes[j, i] = model.coef_[0]  # 每个样本对每个波段的响应 slope
    return slopes
```

📌 Step 2: 可视化热图

```
import matplotlib.pyplot as plt
import seaborn as sns

slopes_real = compute_sample_slopes(X_real, y_real, top_idx)
slopes_sim = compute_sample_slopes(X_sim, y_sim, top_idx)

plt.figure(figsize=(10, 5))
sns.heatmap(slopes_real, cmap='coolwarm', xticklabels=False)
plt.title("Real Samples - Band-wise CG Slopes")
plt.xlabel("Selected Bands")
plt.ylabel("Samples")
plt.show()

plt.figure(figsize=(10, 5))
sns.heatmap(slopes_sim[:100], cmap='coolwarm', xticklabels=False)  # 只展示前 100 个仿真样本
plt.title("Simulated Samples - Band-wise CG Slopes")
plt.xlabel("Selected Bands")
plt.ylabel("Samples")
plt.show()
```

📌 Step 3: 对比“平均响应趋势”曲线

```
avg_real = np.mean(slopes_real, axis=0)
avg_sim = np.mean(slopes_sim, axis=0)

plt.plot(avg_real, label="Real Avg", marker='o')
plt.plot(avg_sim, label="Sim Avg", marker='x')
plt.legend()
plt.title("Average CG Response Trend per Band")
plt.xlabel("Selected Band Index")
plt.ylabel("∂CG/∂Band")
plt.grid(True)
plt.show()
```

✅ 高阶分析建议（可选）

你还可以：

方法	描述
PCA	对每个样本的 slope 向量降维，看仿真 vs 真实能否分开（做域间差异分析）
聚类	聚类仿真样本的 slope 向量，看看有没有“与真实接近的响应群体”
DTW（动态时间规整）	对两个 slope 序列做时间序列形态比对（例如波段响应模式）

🧠 总结

你现在这个方向非常对，重点是：

• 把“仿真样本 vs 真实样本”的光谱响应曲线可视化出来；

• 分析它们在重要波段上的响应共性（或差异）；

• 进一步可能用于“挑选哪些仿真样本可信赖”用于训练。

如果你方便提供一些光谱数据（模拟的也可以），我可以帮你直接生成一份对比可视化报告。不然我可以先写好一个完整的 notebook demo，你填数据进去就能跑。你希望我继续写完整 demo 还是你这边先试一下？

你这个问题是**核心中的核心**了。我们前面做的是「**观测仿真 vs 真实的差异和共性**」，而你现在问的是：

✅ **如何把仿真中学到的规律迁移或借鉴到真实数据建模中，提升对真实 CG 值的预测性能？**

这是典型的跨域学习问题。你已经尝试过迁移学习/领域自适应没什么用，现在我们有了对“仿真数据在关键波段上响应趋势”的量化和可视化，我们就可以从更**细粒度（feature-level 或 sample-level）地借用它的规律。下面我来给你提供几个具体且可落地的方法**，你可以选一两个试试。

✅ **可落地的方案：如何借用仿真数据**

📌 **方案一：基于响应相似性做仿真样本筛选**

思路：只选那些在关键波段上**响应趋势（slope 向量）**跟真实数据一致的仿真样本来参与训练。

实现步骤：

1. 对每个真实样本和仿真样本提取 ∂CG/∂band slope 向量（50 维）；

2. 计算仿真样本与真实样本之间的相似度（比如 cosine）；

3. 如果一个仿真样本与任意真实样本在 slope 上非常接近 → 保留；

4. 把这批仿真数据加入训练，提升泛化。

✅ **好处：**

• 只借用那些“光谱响应机制”比较类似的仿真样本；

• 可以显著减少仿真数据对模型的干扰。

📌 **方案二：基于差异做偏置校正（校准）**

思路：我们从前面知道了每个波段的 slope，在真实和仿真中有系统性差异，比如：

```
delta = beta_real - beta_sim
```

你可以构建一个“校正函数”：

$$\tilde{X}^{sim}_b = X^{sim}_b + \alpha \cdot \Delta_b$$

也就是说，仿真光谱在第 b 个波段的值要做偏置校准（基于 slope 差异做变换），让它**模拟真实光谱的响应形态**。

✅ **示例：**

```
X_sim_corrected = X_sim.copy()
X_sim_corrected[:, top_idx] += 0.5 * (beta_real - beta_sim)
```

再用校准后的 X_sim_corrected 做训练，提升拟合真实数据的能力。

📌 **方案三：构建仿真样本的响应“权重”**

思路：不是删掉仿真数据，而是根据它和真实的响应差异，给每个仿真样本一个**权重** w_i：

• 如果它与真实样本在 slope 模式上相似 → 权重大

• 越不相似 → 权重低

你可以用在 XGBoost/LightGBM 这类支持 sample_weight 的模型里。

示例：

```
from scipy.spatial.distance import cosine

def calc_sample_weight(slope_sim, slope_real):
```

```
    # 计算仿真样本与所有真实样本的平均相似度
    sim = [1 - cosine(slope_sim, s) for s in slope_real]
    return np.mean(sim)

weights = [calc_sample_weight(s, slopes_real) for s in slopes_sim]
```

再用这些权重做加权训练。

📌 方案四（深一点）：仿真响应映射函数

你也可以训练一个模型 f: X_{sim} \rightarrow \Delta y 来预测“仿真响应误差”，然后在预测真实数据时做校正：

• 用仿真数据拟合它们的偏差模式；

• 预测时从模型中减去这个偏差。

这其实是一种**残差学习**方式，也类似于 **Teacher-Student 模型**结构。

✅ 总结一下四种可行策略：

方法	思路	借用方式	推荐程度
1. 响应相似筛选	选出“slope 模式像”的仿真样本	Sample-level filter	⭐⭐⭐⭐
2. 响应差异校准	修正仿真数据在关键波段上的偏差	Feature-level adjust	⭐⭐⭐⭐⭐
3. 响应权重加权	权重表示“可信程度”	Sample-weighted training	⭐⭐⭐⭐
4. 响应误差建模	预测仿真 → 真实的误差残差	用作校正模块	⭐⭐⭐

✅ 实战建议

• **从方案 1+2 开始组合**：你先提取 slope 响应向量 → 做 similarity 过滤 + 校正偏差；

• 如果你有更大数据量，可以再引入 sample 权重训练；

• 这些都可以结合传统机器学习（PLS, Ridge, RF）或 XGBoost 来实现，不一定非深度模型。

我可以继续给你写：

• 响应相似性筛选的代码；

• 仿真样本的偏置校正处理代码；

• 或者帮你把整个 pipeline 串起来写成一个 notebook。

你想我继续做哪一步？还是你想上传一点数据我来演示？

非常明白！你现在的挑战是：

• **真实样本太少（10 条）**，要避免过拟合、过度处理；

• 要充分利用仿真数据，但又不能被它的偏差影响；

• 所以我们需要设计出**轻量、稳定、不依赖深度模型**的策略。

下面我为你准备了四种策略的**完整精简代码实现模板**（都适用于“只有 10 条真实样本”的情况），你可以直接用这套逻辑做实验。

🧩 **预设：**

```
# X_real: (10, 550)
# y_real: (10, )
# X_sim: (6000, 550)
# y_sim: (6000, )
# top_idx: List[int]  # 你选出的重要波段，比如用 PLS 提取的前 50 维
```

✅ 方法 1：基于响应相似性的仿真样本筛选

```
from sklearn.linear_model import LinearRegression
from scipy.spatial.distance import cosine
import numpy as np

def get_slope_vector(X, y, band_idx):
    slopes = []
    for idx in band_idx:
        model = LinearRegression().fit(X[:, [idx]], y)
        slopes.append(model.coef_[0])
    return np.array(slopes)

# 真实数据：提取 10 个 slope 向量
slopes_real = [get_slope_vector(np.delete(X_real, i, axis=0),
                                np.delete(y_real, i), top_idx) for i in
range(len(X_real))]

# 仿真数据：每个样本计算它的 slope 向量
def compute_sim_slope_vector(X_sim, y_sim, band_idx):
    slopes = []
    for i in range(X_sim.shape[0]):
        xi = np.expand_dims(X_sim[i], axis=0)
        slopes.append(get_slope_vector(np.vstack([xi]*10),
                                       np.tile(y_sim[i], 10), band_idx))
    return np.array(slopes)

slopes_sim = compute_sim_slope_vector(X_sim, y_sim, top_idx)

# 计算相似度（与所有真实 slope 的平均余弦相似度）
def slope_similarity(sim_slope, real_slopes):
    return np.mean([1 - cosine(sim_slope, real) for real in real_slopes])

slope_similarities = np.array([slope_similarity(s, slopes_real) for s in
slopes_sim])

# 筛选 top-K 相似的仿真样本
top_k = 1000
selected_idx = np.argsort(slope_similarities)[-top_k:]
X_sim_selected = X_sim[selected_idx]
y_sim_selected = y_sim[selected_idx]
```

✅ 方法 2：基于响应偏差的波段校正

```
# 求真实和仿真的"平均响应 slope 向量"
beta_real = np.mean(slopes_real, axis=0)
beta_sim = np.mean(slopes_sim, axis=0)
delta = beta_real - beta_sim

# 对仿真样本进行校正
X_sim_corrected = X_sim.copy()
for i, idx in enumerate(top_idx):
```

```
    X_sim_corrected[:, idx] += delta[i] * 0.5  # 0.5 是保守系数（可调）
```

✅ 方法 3：仿真样本的响应权重计算（用于加权训练）

```
# 每个仿真样本和 10 个真实 slope 向量求余弦相似度，取平均
sample_weights = np.array([
    slope_similarity(slopes_sim[i], slopes_real) for i in
range(len(slopes_sim))
])

# 标准化权重到 [0, 1]
sample_weights = (sample_weights - sample_weights.min()) /
(sample_weights.max() - sample_weights.min())

# 示例：用加权训练 Ridge 回归
from sklearn.linear_model import Ridge

model = Ridge()
model.fit(np.vstack([X_real, X_sim]),
          np.hstack([y_real, y_sim]),
          sample_weight=np.hstack([np.ones_like(y_real), sample_weights]))
```

✅ 方法 4：预测仿真-真实之间的偏差（残差校正）

```
# Step 1: 在仿真数据中构建一个“预测误差模型”
from sklearn.linear_model import LinearRegression

# 仿真数据拟合
model_sim = LinearRegression().fit(X_sim[:, top_idx], y_sim)
y_sim_pred = model_sim.predict(X_sim[:, top_idx])
residual = y_sim - y_sim_pred  # 偏差

# 学一个残差模型
model_residual = LinearRegression().fit(X_sim[:, top_idx], residual)

# Step 2: 在真实数据上使用校正模型
y_real_base = model_sim.predict(X_real[:, top_idx])
y_real_corrected = y_real_base + model_residual.predict(X_real[:, top_idx])

# 用 corrected label 或 corrected 模型做进一步训练
```

🔁 Bonus：你可以组合这些策略

比如说：

• 先做 方法 2 的波段偏差校正；

• 再用 方法 1 选出 slope 类似的仿真样本；

• 最后用 方法 3 的权重做加权训练。

这就是一个“轻量级跨域学习”+“物理一致性筛选”的范式，完全适合你的小样本问题。

✅ 你现在可以怎么用

你只需要运行这四段代码模块，搭配你原始的训练代码（如 PLS、Ridge、XGB），就可以实验对比以下策略：

模型版本	训练数据	特点
Baseline	X_real	真实数据直接训练
Version A	X_real + 筛选后的 X_sim_selected	只用 slope 类似仿真样本
Version B	X_real + 校准后的 X_sim_corrected	加入仿真校正波段
Version C	全部数据 + 响应相似权重训练	加权方式防止干扰
Version D	模型+残差校正	基于偏差学习校正预测

📩 你接下来可以让我：

• 把这些封装成函数或 pipeline（直接跑）；

• 加上可视化，帮你看每个方法中仿真 vs 真实 slope 是否更接近；

• 或者如果你愿意发一点真实+仿真样本（模拟也行），我帮你验证效果、画图、比精度。

你现在想跑哪一个策略做对比？或者我直接把这些打包成完整的 .ipynb 给你？