



中国研究生创新实践系列大赛
“华为杯”第二十二届中国研究生
数学建模竞赛

学 校 上海工程技术大学

参赛队号 E25108560076

1.王流斌

队员姓名 2.杨哲

3.李佳敏

中国研究生创新实践系列大赛
“华为杯”第二十二届中国研究生
数学建模竞赛

题 目 基于 LightGBM/CNN 的高速列车轴承智能故障诊断

摘 要:

为了应对高速列车轴承故障样本稀缺和信号噪声强的挑战，本文提出了一种基于迁移学习的智能诊断方法。通过将试验台架数据（源域）提取的多维度特征与诊断模型结合，采用迁移学习策略，将知识迁移到无标签的实际列车数据（目标域），有效克服了试验与实际工况之间的数据分布差异，成功实现了目标域轴承故障的精准分类。

为了解决源域与目标域数据间的显著差异，本研究设计了一套完整的数据预处理与特征工程流程，**将原始振动信号转化为结构化的特征向量**。首先，通过信号重采样，将不同采样率的源域数据统一为 32kHz，以保证数据一致性；随后，通过滑动窗口法对信号进行分段，并采用**时域统计、频域分析和故障机理分析**等多种算法提取 28 个特征维度。最终，获得了一个包含丰富信息的特征表，为迁移学习模型训练提供了结构化输入数据。

在故障诊断模型构建上，本文选用了在工业界表现卓越的 LightGBM 分类器。通过多轮迭代优化，最终实现了高效且精准的诊断模型。为了提升模型的分类精度和泛化能力，**本研究对 LightGBM 的关键超参数进行了优化**，并通过混淆矩阵与特征重要性分析可视化诊断结果，明确了不同故障类型的关键判别特征。

针对源域模型在目标域应用时出现的性能退化问题，本研究通过迁移学习实现了跨域知识迁移。具体而言，利用源域上训练好的 LightGBM 模型，并结合域对齐技术，缓解

了数据分布差异带来的影响。同时，引入 CNN 模型，通过**短时傅里叶变换（STFT）**将时序信号转化为多通道图像，进一步提高了诊断精度。带权重的随机采样方法确保了各类样本在训练中的学习均衡，最终为目标域数据的有效诊断提供了可靠的解决方案。

本文提出的迁移学习方法不仅解决了高速列车轴承故障诊断中的数据稀缺和噪声问题，还为实际运营环境下的故障诊断提供了有效的技术支持，具有较强的应用前景和推广价值。

关键词：LightGBM；卷积神经网络；特征工程；域自适应

目录

一、 问题重述	1
1.1 问题背景.....	1
1.2 垄待解决的问题.....	1
二、 问题一：特征工程与数据标准化.....	2
2.1 数据介绍.....	2
2.2 数据分类.....	3
2.2.1 数据信息索引.....	3
2.2.2 数据预处理.....	3
2.2.3 可视化统计.....	4
2.3 分段与预处理.....	6
2.3.1 源域预处理.....	6
2.3.2 数据存储与索引管理.....	7
2.3.3 信号重采样.....	9
2.3.4 通道选择.....	9
2.3.5 数据结构化输出.....	10
2.4 特征提取.....	10
2.4.1 时域特征.....	10
2.4.2 频域特征.....	13
2.5 特征分析.....	14
2.5.1 类别分布统计.....	14
2.5.2 特征相关性分析.....	16
2.5.3 主成分分析（PCA）	17
2.5.4t-SNE 非线性降维.....	20
三、 问题二：基于梯度提升树(LightGBM)的源域故障诊断	21
3.1 LightGBM 建模与训练.....	22
3.1.1 LightGBM 介绍	22
3.2.2 方法.....	24
3.2.3 结果.....	25
3.2 诊断结果可视化与解释.....	26
3.3.2 特征重要性分析.....	27
3.3.3 诊断性能总结.....	28
3.3.4 小结.....	29
四、 问题三：LightGBM 与 CNN 迁移诊断.....	30
4.1 目标域预处理.....	30
4.2 目标域特征提取.....	30
4.2.1 时域特征.....	30
4.2.2 频域特征.....	31
4.2.5 特征故障频率计算.....	33
4.3 迁移诊断与预测.....	34
4.4 模型保存与部署准备.....	34
4.5 基于 CNN 的时傅里叶变换（STFT）多通道图像迁移	35
五、 问题四：迁移诊断的可解释性.....	38

六、 总结与展望.....	38
6.1 总结	38
6.2 展望	39
七、 附录	41

一、问题重述

1.1 问题背景

高速列车已成为我国客运交通体系的中坚力量，其运行安全始终是国家和社会关注的重点。作为列车走行系统中至关重要的旋转部件，轴承在长期高速运转与重载工况下工作，所处环境极为严苛，因此成为系统中故障率最高的薄弱环节之一。一旦轴承失效，轻则造成经济损失，重则可能引发灾难性事故，严重威胁人民生命财产安全。因此，开展对高速列车轴承的精准和及时故障诊断，已成为保障列车安全运行的关键技术环节。

目前，轴承故障诊断正经历由依赖专家经验的传统方法，向以数据驱动为核心的智能化方法转型。以深度学习为代表的智能诊断技术，凭借其高精度和强自适应性，展现出广阔的应用前景。然而在实际推广中仍存在两大突出瓶颈：

- (1) 数据质量问题：运行环境下采集的振动信号不可避免地受到强烈背景噪声及多源干扰影响，从而显著削弱故障特征的可辨识度；
- (2) 数据稀缺问题：出于安全考虑，早期故障往往会被及时检修，导致在役故障样本极度稀少，形成严重的“数据孤岛”，并引发样本类别分布不平衡。

相比之下，在试验台架环境中可以获取数量充足、标签完善且涵盖多种故障类型的轴承全生命周期数据。虽然台架数据（源域）与实际运营数据（目标域）在噪声水平、采样频率和转速工况等方面存在显著差异，但其底层的故障演化机理高度一致。

因此，迁移学习技术为解决上述问题提供了理想方案。其核心思想是将源域中积累的故障诊断知识迁移至数据稀缺的目标域，从而突破数据壁垒，实现对实际运营高速列车轴承的高精度智能诊断。

1.2 垂待解决的问题

在上述背景下，本研究以 161 个具有代表性的轴承试验台架振动数据文件作为源域数据集，并选取 16 个实际运营中的轴承故障文件作为目标域数据集，拟重点解决以下四个方面的核心问题

问题一：特征工程与数据标准化。

首先，需要设计并实施一套完整的数据预处理与特征工程流程，以实现从原始、多源试验台架振动信号到标准化、结构化数据集的转换。该流程必须遵循“最小化领域差异”原则，综合考虑传感器布置位置、采样频率及载荷条件，从源域数据中筛选最具代表性的数据子集。随后，结合轴承故障的物理机理，提取并验证典型特征频率，包括：外圈故障频率 BPFO；内圈故障频率 BPFI；滚动体故障频率 BSF。

通过信号处理与分析手段验证特征的有效性，并将该验证后的流程应用于全部筛选数据，完成数据

清洗、重采样、分段、归一化与多维特征提取的全过程，最终形成一个带标签的、可直接用于迁移学习模型训练的结构化特征数据集。

问题二：构建高能源域模型。

在特征工程基础上，将源域数据划分为训练集和测试集，并据此设计和实现一个适合的故障诊断模型。通过对模型性能进行系统评估，确保其在准确性、鲁棒性和泛化能力方面满足要求，从而为跨域迁移诊断提供可靠的源域模型支撑

问题三：实现跨域迁移诊断。

在源域诊断模型的基础上，针对源域与目标域之间的共性与差异，设计并实现合适的迁移学习方法，构建目标域诊断模型。该模型应能够实现以下目标：

- 对目标域中的未知标签数据进行准确分类与标定；
- 提供迁移结果的可视化展示，直观反映迁移效果与分类边界；
- 在保证诊断精度的前提下，实现跨域知识的高效迁移。

通过这一过程，有望有效解决实际运行环境下因故障样本稀缺和数据分布差异带来的挑战。

问题四：增强的可解释性。

深度学习模型在一定程度上存在“黑箱”特性，其迁移与诊断过程往往难以直观理解，这可能导致诊断人员对模型结果产生不信任或盲目信任。为确保模型的可靠应用，有必要开展迁移诊断模型的可解释性研究。

研究目标在于增强诊断人员对模型结构设计、迁移过程及决策机制的理解与信任度。具体而言，应从事前、过程与事后三个层面对模型进行解释，结合轴承故障机理，对模型输出结果进行合理化说明，从而提高迁移诊断系统的透明度与可靠性。

通过系统性地解决上述四个方面的问题，本研究将构建一个涵盖数据预处理、源域建模、跨域迁移和结果解释的完整诊断闭环体系，为推动高速列车轴承故障诊断技术的智能化和高效化发展提供理论与实践支持。

二、问题一：特征工程与数据标准化

2.1 数据介绍

源域数据来自实验室环境下采集的轴承故障振动信号，涵盖了驱动端（DE）、风扇端（FE）和基座（BA）三个测点的加速度数据。轴承故障类型包括外圈故障（OR）、内圈故障（IR）、滚动体故障（B）以及正常状态（N），并涉及多种故障尺寸与载荷工况。数据的采样频率主要为 12kHz 与 48kHz。

目标域数据则来源于实际运营的高速列车，其轴承振动信号的采样频率为 32kHz，列车运行速度约为 90km/h（对应的轴承转速约为 600rpm）。该数据集包含 16 个未知状态的数据文件，其故障标签需通过后续建立的诊断模型进行标定。

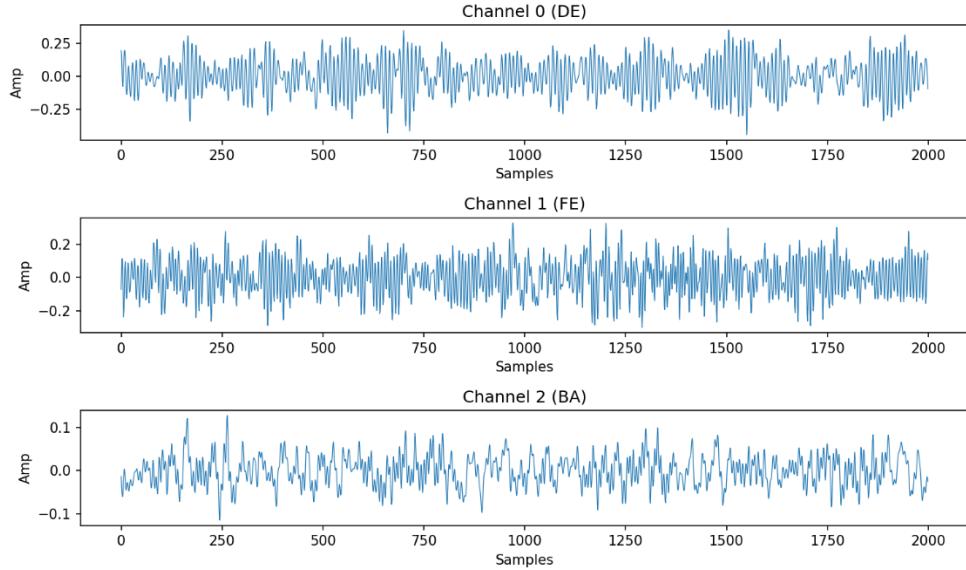


图1 B0007_0 三通道数据展示

2.2 数据分类

2.2.1 数据信息索引

为了实现源域数据的系统化管理并支撑后续的分析工作，首要任务是对原始数据进行整理和索引。为此，本研究编写了自动化脚本 1 索引.py，以批量解析提供的 161 个试验台架 .mat 数据文件。该脚本从每个文件中提取了关键的元信息，包括：采样频率（12kHz 或 48kHz）、传感器通道（驱动端 DE、风扇端 FE、基座 BA）、轴承健康状态（外圈故障 OR、内圈故障 IR、滚动体故障 B、正常 N）、故障尺寸、载荷以及转速（RPM）等。这些元信息被整合并生成了结构化的索引表（如 detail_12k.csv 和 detail_48k.csv）。

该索引表为后续的数据筛选、工况匹配与统计分析奠定了基础，确保了数据处理过程的透明性和可重复性。为了验证不同故障类别在特征频率指标上的分布差异，本研究对源域数据进行了预处理与统计分析，具体步骤如下：

2.2.2 数据预处理

通过 `scipy.io.loadmat` 与 `h5py` 模块读取源域数据中的时域信号，从每个数据文件中提取驱动端（DE）、风扇端（FE）、基座（BA）等传感器的信号，并统一保存为 CSV 格式，以便后续处理。同时，解析文件名中的元信息（如故障类型、转速、载荷、故障尺寸等），并将这些信息保存到 0detail.csv

索引表中，作为标签信息供进一步分析使用

2.2.3 可视化统计

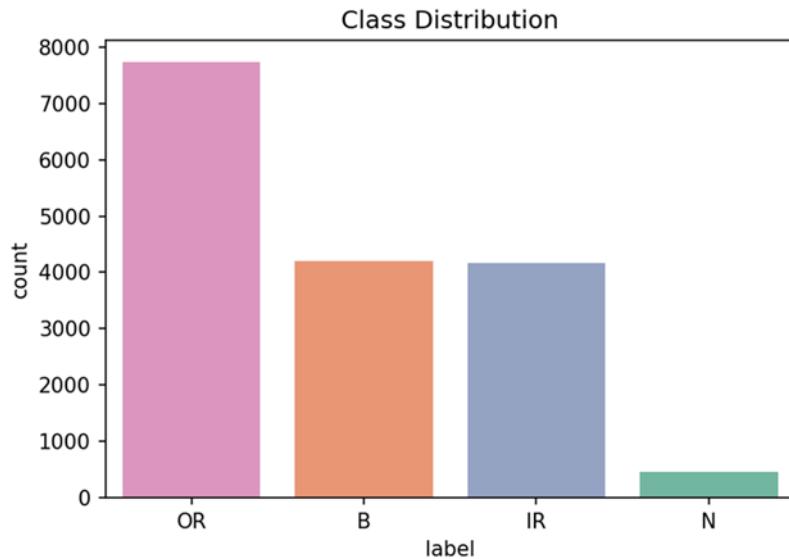


图2

图 2 展示了源域数据集中四类轴承状态的样本分布情况，分别为外圈故障（OR）、滚动体故障（B）、内圈故障（IR）以及正常状态（N）。

从图中可以看出：

外圈故障（OR）样本数量最多，接近 8000 条，占据了数据集的主要部分。这与外圈故障在实验设置中较容易通过不同缺陷尺寸与载荷条件生成有关。

滚动体故障（B）与内圈故障（IR）样本数量相对均衡，均在 4000 条左右，为模型训练提供了较好的对比基础。

正常状态（N）样本数量最少，不足 500 条，与故障类样本相比呈现明显的数据不平衡问题。

这种类别分布特征表明，数据集中存在一定程度的样本不均衡。特别是正常样本数量显著偏少，可能会导致模型在训练过程中对正常状态的识别能力不足，从而影响诊断精度。为缓解这一问题，后续研究可考虑以下措施：

采用过采样/欠采样或数据增强方法，平衡不同类别的样本规模。

在模型训练中引入类别加权损失函数（class-weightedloss），提升对少数类（N 类）的识别能力；

使用迁移学习方法，将源域中学到的特征知识迁移到目标域，以减少样本不均衡带来的偏差。

综上，源域数据集在总体上覆盖了多种典型故障模式，但其内部类别比例差异显著。这一特性需要在后续建模与实验设计中予以充分考虑。

我们还对数据按照标签（OR、IR、B、N）对所有样本的 BPFO_ratio 进行统计。使用直方图（Histogram）展示不同类别的分布情况：横轴为 BPFO_ratio 数值，纵轴为样本数。不同颜色代表不同故障状态（B=蓝色、IR=橙色、OR=红色、N=绿色）。通过上述步骤，我们得到了 hist_BPFO_ratio.png 可视化结果。

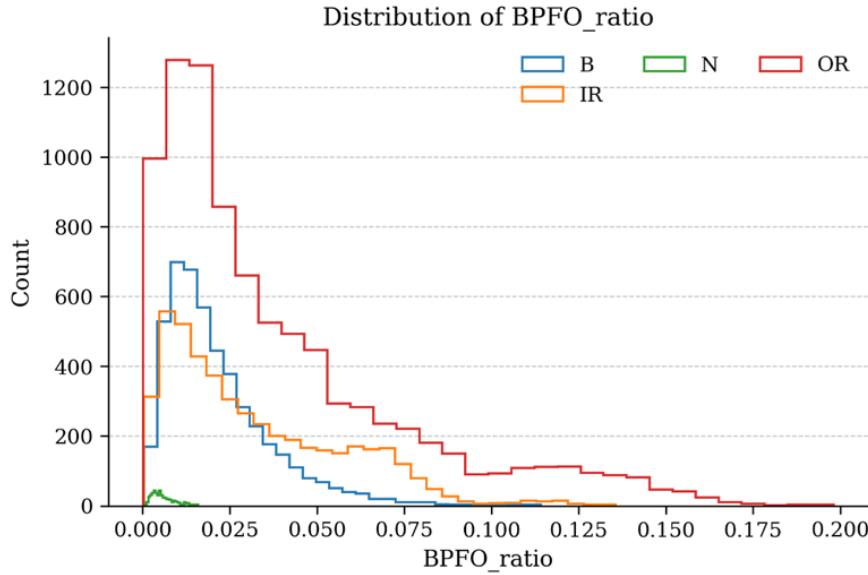


图3

这幅直方图展示了不同故障状态下 BPFO_ratio（外圈故障特征频率比值）的分布情况。横轴是 BPFO_ratio 值，纵轴是出现次数（Count），不同颜色的线条表示不同的轴承状态：

B（蓝色）：滚动体故障

IR（橙色）：内圈故障

OR（红色）：外圈故障

N（绿色）：正常状态

从图中可以解读出以下几点：外圈故障（OR, 红色）：分布最集中在靠近 0 附近，出现次数远高于其他类型。有较长的“尾部”，说明外圈故障在较宽范围内都有可能出现 BPFO_ratio 特征。这是因为外圈缺陷位置固定，每次滚动体经过都会产生规律性冲击，所以 BPFO 特征较强。内圈故障（IR, 橙色）与滚动体故障（B, 蓝色）：分布主要集中在 0.01~0.05 区间。它们的曲线形状相似，说明 BPFO_ratio 对这两类故障有一定区分度，但不像外圈那样明显。正常状态（N, 绿色）：出现次数很少，分布范围也比较窄（集中在 0.005~0.015 之间）。说明正常情况下振动信号中很少包含外圈特征频率成分。整体规律：

OR 特征最明显，能通过 BPFO_ratio 明确区分。IR 与 B 有部分重叠，可能需要结合其他特征（如

BPFI、BSF、时域特征) 才能进一步区分。N 的分布与故障类明显不同, 在分类时容易与故障样本分开

2.3 分段与预处理

为了保证故障诊断模型的输入数据具有统一的结构与较高的信噪比, 本研究对采样率统一为 $f_s = 32\text{kHz}$ 的驱动端 (DE) 振动信号进行分段与预处理。完整流程包括滑窗切片、带通滤波、小波去噪与归一化四个环节, 具体如下。

2.3.1 源域预处理

滑窗切片 (Segmentation)

设原始时域信号为:

$$x[n], n = 0, 1, \dots, N - 1$$

其中 N 为样本总长度。

采用固定长度 L=4096 的滑动窗口进行切片, 步长设为 h=2048。第 k 个片段表示为:

$$x_{k[n]} = x[n + k \cdot h], n = 0, 1, \dots, L - 1$$

可以得到的片段总数为:

$$K = \left\lfloor \frac{(N - L)}{h} \right\rfloor +$$

这种方法既保证了每个片段包含足够的故障信息, 又通过重叠采样增加了数据量

带通滤波 (Band-pass Filtering)

振动信号往往包含直流漂移和高频噪声。为突出轴承故障的有效频带成分, 采用 4 阶巴特沃斯带通滤波器, 频带范围设定为 [100, 5000]Hz。

滤波过程为:

$$y_{k[n]} = (h \ast x_k)[n] = \sum_{\substack{\{m=-\infty\} \\ k[n-m]}}^{\infty} h[m] \cdot x[n-m]$$

其中 $h[m]$ 为滤波器冲激响应, $y_{k[n]}$ 为滤波后的信号。

小波去噪

在滤波的基础上, 进一步采用小波阈值去噪以抑制非平稳噪声。设小波多尺度分解系数为 $\{c_j\}$, 则噪声标准差估计为:

$$\sigma = \frac{\text{median}(|c_j|)}{0.6745}$$

阈值取: $\lambda = \sigma \sqrt{2 \{ \ln \} \ln \{ L \}}$

对各尺度系数 c_j 进行软阈值处理:

$$c_j^{\prime} = \text{sign}(c_j) \cdot \text{sign}(\left|c_j\right| - \lambda)$$

最后通过逆小波变换得到去噪后的信号：

$$\widetilde{\{x_k\}}[n] = \text{WaveletReconstruct}(\{c_j'\})$$

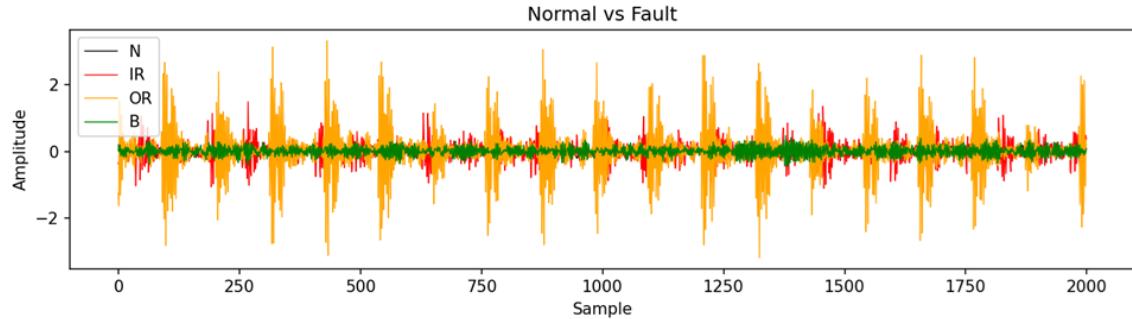


图4

图 4 展示了正常轴承 (N) 与三类典型故障 (IR 内圈、OR 外圈、B 滚动体) 的时域波形对比。

结论如下：

N (正常)：波动小、无明显冲击；

OR (外圈故障)：周期性冲击最明显，幅值最大；

IR (内圈故障)：存在周期性冲击，但受调制影响幅值不均；

B (滚动体故障)：冲击较轻，周期性不如 OR 明显。

这表明不同故障类型在时域信号中具有显著差异。

这张图的作用是直观展示：

不同故障类型会在原始振动信号中留下不同的时域特征模式；正常与故障可以在波形层面区分开。

归一化

由于不同工况下采集的振动信号幅值差异较大，若不加处理，可能对模型的收敛性与分类性能产生不利影响。为此，本研究对每段信号实施区间归一化，将其统一映射至[-1,1] 区间内，以减小幅值差异所带来的影响。归一化公式表示为：

$$z_k[n] = \frac{2 \cdot (\tilde{x}_k[n] - \min(\tilde{x}_k))}{\max(\tilde{x}_k)} - 1$$

当 $\left(\tilde{x}_k\right) = \left(\tilde{x}_k\right)$ 时，直接置为零向量，避免除零错误。

2.3.2 数据存储与索引管理

在完成归一化后，每个预处理后的片段都被单独保存为 .npy 文件。同时记录片段的来源文件路径、

分段索引、起止位置以及相应的工况参数（转速、载荷、故障类别等），并统一整理至 segments.parquet 索引表中，以支撑后续特征提取与模型训练。

csv_path	,	发生点,	赫兹 ,	故障类,	尺寸 ,	负载,	转速,	采样点
第3问_CNN/cleaned/12kHz_DE_data/B/0014/B014_1.csv	,	FE	, 12000, B	, 0.014,	1.0,	1775,		
第3问_CNN/cleaned/12kHz_DE_data/B/0014/B014_2.csv	,	FE	, 12000, B	, 0.014,	2.0,	1755,		
第3问_CNN/cleaned/12kHz_DE_data/B/0014/B014_3.csv	,	FE	, 12000, B	, 0.014,	3.0,	1732,		
第3问_CNN/cleaned/12kHz_DE_data/B/0021/B021_0.csv	,	FE	, 12000, B	, 0.021,	0.0,	1796,		
第3问_CNN/cleaned/12kHz_DE_data/B/0021/B021_1.csv	,	FE	, 12000, B	, 0.021,	1.0,	1775,		
第3问_CNN/cleaned/12kHz_DE_data/B/0021/B021_2.csv	,	FE	, 12000, B	, 0.021,	2.0,	1756,		
第3问_CNN/cleaned/12kHz_DE_data/B/0021/B021_3.csv	,	FE	, 12000, B	, 0.021,	3.0,	1734,		
第3问_CNN/cleaned/12kHz_DE_data/B/0028/B028_0_(1797rpm).csv	,	DE	, 12000, B	, 0.028,	0.0,	1797,		
第3问_CNN/cleaned/12kHz_DE_data/B/0028/B028_1_(1772rpm).csv	,	DE	, 12000, B	, 0.028,	1.0,	1772,		
第3问_CNN/cleaned/12kHz_DE_data/B/0028/B028_2_(1750rpm).csv	,	DE	, 12000, B	, 0.028,	2.0,	1750,		
第3问_CNN/cleaned/12kHz_DE_data/B/0028/B028_3_(1730rpm).csv	,	DE	, 12000, B	, 0.028,	3.0,	1730,		
第3问_CNN/cleaned/12kHz_DE_data/IR/0007/IR007_0.csv	,	FE	, 12000, IR	, 0.007,	0.0,	1796,		
第3问_CNN/cleaned/12kHz_DE_data/IR/0007/IR007_1.csv	,	FE	, 12000, IR	, 0.007,	1.0,	1772,		
第3问_CNN/cleaned/12kHz_DE_data/IR/0007/IR007_2.csv	,	FE	, 12000, IR	, 0.007,	2.0,	1755,		
第3问_CNN/cleaned/12kHz_DE_data/IR/0007/IR007_3.csv	,	FE	, 12000, IR	, 0.007,	3.0,	1732,		
第3问_CNN/cleaned/12kHz_DE_data/IR/0014/IR014_0.csv	,	FE	, 12000, IR	, 0.014,	0.0,	1796,		

图5

上图展示了源域试验台架轴承数据的部分文件路径及其对应的标签信息。每条记录均包含以下关键属性：

采样位置（Location）：主要包括驱动端（DE）和风扇端（FE）。不同位置的传感器能够采集到不同衰减路径下的振动信号。

采样频率（Sampling frequency）：多数样本为 12 kHz，与后续目标域数据（32 kHz）相比需进行重采样以实现频率对齐。

故障类型（Fault type）：包括滚动体故障（B）、内圈故障（IR）、外圈故障（OR）等，涵盖了典型的轴承失效模式。

故障尺寸（Fault size）：如 0.007、0.014、0.021 英寸等，反映了局部缺陷的大小，对振动幅值和特征频率分布具有直接影响。

载荷条件（Load condition）：标记为 0、1、2、3 马力，表征轴承在不同运行工况下的受力水平。

转速（Rotational speed, rpm）：部分样本包含精确转速信息（如 1772 rpm、1796 rpm），为故障特征频率计算提供支撑。

采样点数（Number of sampling points）：多数样本包含 1775 或 1797 点，保证了信号序列长度的一致性。

综上所述，该索引表直观呈现了数据集的多维特征，表明源域数据不仅涵盖多种典型故障模式，还在采样位置、运行工况与故障程度上体现了较高的多样性。这为后续特征提取与故障诊断模型的构建奠定了坚实的数据基础。

2.3.3 信号重采样

源域数据的采样率存在 12 kHz 与 48 kHz 两种，而目标域则统一为 32 kHz。若不进行重采样，频域特征与故障特征频率将无法对齐，从而加大源域与目标域间的分布差异。

方法与公式：

重采样通过上采样(Upsampling)与下采样(Downsampling)实现：

$$f_{\text{out}} = f_{\text{in}} * \left(\frac{L}{M}\right)$$

其中： f_{in} 为原始采样率； f_{out} 为目标采样率(32kHz)； L 为上采样因子； M 为下采样因子。

①12kHz→32kHz: $f_{\text{out}}/f_{\text{in}}=32000/12000=8/3 \rightarrow L=8, M=3$ 。

②48kHz→32kHz: $f_{\text{out}}/f_{\text{in}}=32000/48000=2/3 \rightarrow L=2, M=3$ 。

在代码实现中，使用 `scipy.signal.resample_poly(sig, up=L, down=M)` 基于多相滤波(PolyphaseFiltering)完成重采样，具有高效与抗混叠优势。

图 6：重采样前后波形对比图（原始 vs 32kHz）。

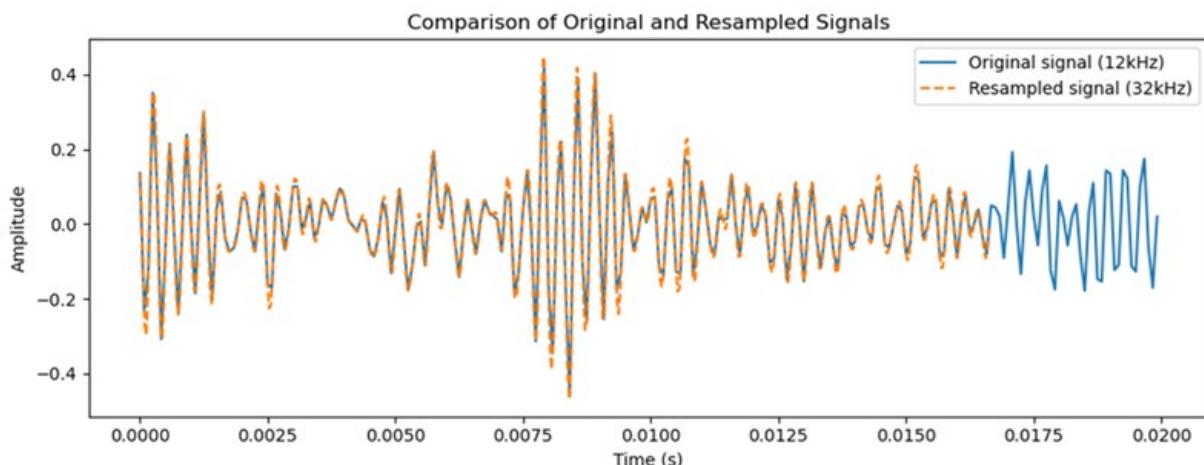


图6

从图中可以观察到，重采样后的信号 (32kHz, 橙色虚线) 在整体波形形态上与原始信号 (12kHz, 蓝色实线) 高度一致。无论是在低频段还是高频段，两者的幅值变化趋势几乎完全重合，仅在局部细节处出现微小差异。这表明在重采样过程中信号的主要时域特征和故障信息得到了有效保留。

因此，可以认为本研究所采用的重采样方法能够在保证时域特征不失真的前提下，实现不同采样频率信号间的数据对齐。这一处理为后续源域与目标域的迁移诊断奠定了坚实的数据基础

2.3.4 通道选择

在本研究中，振动信号主要通过驱动端 (DE)、风扇端 (FE) 以及声发射 (AE) 传感器采集。综合考虑实际应用需求与信号特征的稳定性，本文在建模阶段优先选择 DE 通道作为主要分析对象，原

因如下

信号耦合最直接，故障特征最明显

DE 传感器位置紧邻驱动电机与轴承，振动信号无需经过复杂的结构传递路径，能够清晰保留故障冲击特征。相比之下，FE 信号在长距离传递过程中会发生能量衰减与频谱混叠，削弱了故障特征。

数据噪声更低，稳定性更强

AE 信号虽然对早期微小裂纹较为敏感，但在实际运行环境中容易受到背景噪声与电磁干扰的影响，导致特征提取不稳定。相较而言，DE 通道信号更具鲁棒性，有助于建模阶段快速收敛。

任务复杂度可控，利于快速验证

在研究初期，选择单一关键通道（DE）能有效降低数据处理与建模的复杂度，便于快速完成完整诊断流程（数据预处理—特征提取—建模分类）。此举在保证实验可重复性的同时，也为后续多通道融合（如 DE+FE 或 DE+FE+AE）提供了基准对比。

综上所述，基于信号特征清晰度、噪声水平及实验任务可控性，本研究在初步实验阶段仅选取 DE 通道进行分析。这一选择不仅符合多数相关研究的常见做法，也为后续多通道融合研究奠定了可靠基准。

2.3.5 数据结构化输出

在完成采样率统一与通道筛选后，研究对数据进行了结构化存储，以便于后续分段与特征提取。

具体方法如下：

所有处理后的信号文件保存至目录 1csv_32k_DE/，确保仅包含 DE 通道，且采样率统一为 32 kHz。

更新索引表 1detail_32k_DE.csv，记录文件路径、采样率、通道等关键信息，保证数据管理的一致性与可追溯性。

2.4 特征提取

在完成分段与预处理之后，本研究针对每一段信号片段 $x[n]$ （长度 $L=4096$ ，采样率 $f_s = 32\text{kHz}$ ）提取了时域特征、频域特征、包络谱特征和小波包特征，构成最终的特征向量。

2.4.1 时域特征

时域特征直接反映信号幅值分布和冲击特性。设信号均值与标准差分别为：

$$\mu = \frac{1}{L} \sum_{n=0}^{L-1} x[n], \sigma = \sqrt{\frac{1}{L-1} \sum_{n=0}^{L-1} (x[n] - \mu)^2}$$

提取的主要时域指标包括：

均方根值（RMS）：

$$x_{\text{rms}} = \sqrt{\frac{1}{L} \sum_{n=0}^{L-1} x[n]^2}$$

偏度 (Skewness) 与峰度 (Kurtosis) :

$$\text{Skew} = \frac{1}{L} \sum_{n=0}^{L-1} \left(\frac{x[n] - \mu}{\sigma} \right)^3$$

$$\text{Kurt} = \frac{1}{L} \sum_{n=0}^{L-1} \left(\frac{x[n] - \mu}{\sigma} \right)^4 - 3$$

冲击性指标 (以 $x_a[n] = |x[n]|$ 为绝对值信号) :

$$\text{CrestFactor} = \frac{1}{L} \sum_{n=0}^{L-1} x_a[n]$$

$$\text{ShapeFactor} = \frac{1}{L} \sum_{n=0}^{L-1} x_a[n]$$

$$\text{ImpulseFactor} = \frac{1}{L} \sum_{n=0}^{L-1} x_a[n]$$

$$\text{ClearanceFactor} = \frac{1}{L} \sum_{n=0}^{L-1} \sqrt{x_a[n]}$$

信号熵 (Entropy) : 基于直方图概率 p_i 计算:

$$H_s = - \sum_{i=1}^{B_p} p_i \log p_i$$

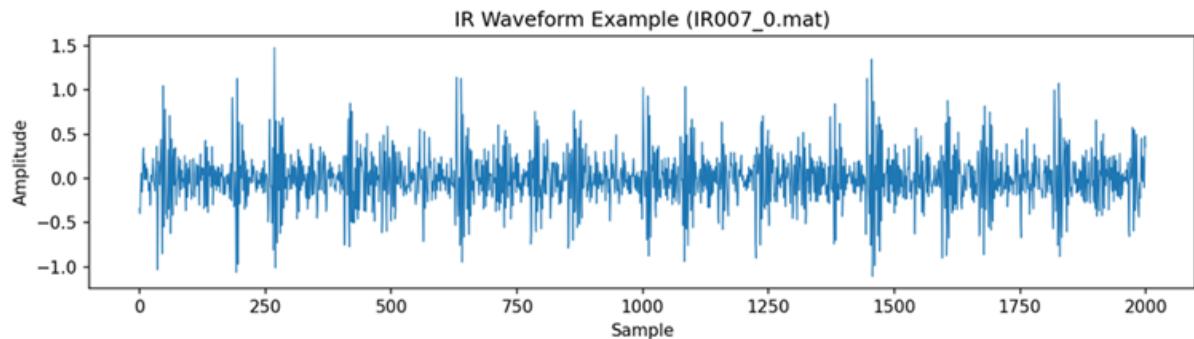


图7

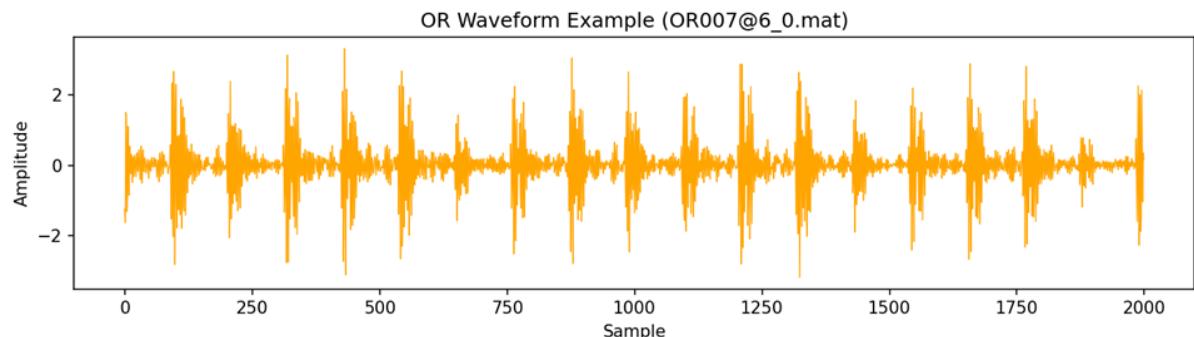


图8

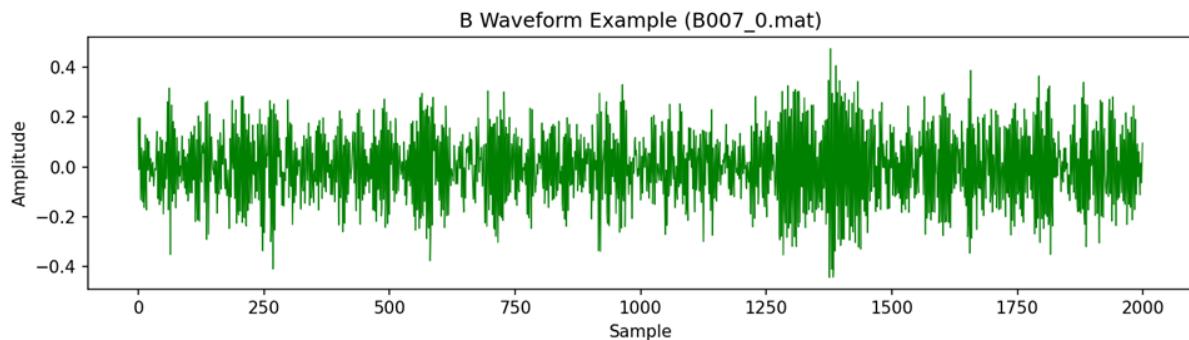


图9

图 7 – 图 9 展示了三类典型轴承故障（内圈故障 IR、外圈故障 OR、滚动体故障 B）的振动信号在时域上的波形特征。通过对比可清晰观察到，不同类型故障在信号表现上具有明显差异。

(1) 内圈故障 (IR, 图 2.7)

在内圈故障的时域波形中，可以明显观察到背景噪声上叠加着显著的冲击成分。由于内圈随转子一同旋转，当缺陷点周期性进入载荷区时，会产生冲击脉冲，从而导致幅值出现周期性调制效应。这种信号特征与内圈故障的物理机理相一致，即在时域上表现为明显的幅值调制特征。

(2) 外圈故障 (OR, 图 2.8)

外圈故障的时域波形特征更为规律，冲击脉冲幅值较高且间隔均匀。在固定载荷作用下，每当滚动体通过外圈缺陷时，都会产生幅值近似相同的冲击信号，因此波形表现为等间隔、规则性强的脉冲序列。这一特征正是外圈故障最为典型的时域表现，也是其与其他类型故障区分度最高的特征。

(3) 滚动体故障 (B, 图 2.9)

与内圈和外圈故障相比，滚动体故障的冲击特征表现得更加分散，幅值相对较低，且波形整体上与噪声信号混合度较高。这是由于滚动体在旋转过程中会与内圈和外圈交替接触，冲击强度受缺陷位置及载荷变化的影响，因而其时域信号特征不如外圈故障或内圈故障那样突出和稳定。

(4) 对比与总结

综合上述三类典型故障的时域特征，可以得出以下结论：

外圈故障 (OR)：冲击周期性最为明显，幅值最高，特征规律性最强。

内圈故障 (IR)：表现出周期性冲击，但受转轴旋转调制影响，幅值存在波动。

滚动体故障 (B)：冲击特征较弱，整体信噪比较低，区分难度相对较大。

综上所述，外圈故障的特征最为清晰和稳定，内圈故障则具有较强的调制效应，而滚动体故障的时域特征相对模糊。不同故障类型在波形上的差异性为后续的特征提取与分类建模提供了直观的依据和

理论支撑。

2.4.2 频域特征

采用 Welch 方法估计功率谱密度 $P(f)$, 归一化为 $p(f)=P(f)/\sum_f P(f)$ 。

提取的频域特征包括:

谱峰频率:

$$f_{\text{peak}} = \arg \max_{f} p(f)$$

谱质心与扩展度:

$$f_c = \frac{\sum f \cdot p(f)}{\sum p(f)}, \sigma_f = \sqrt{\sum (f - f_c)^2 p(f)}$$

谱熵:

$$H = -\sum p(f) \log p(f)$$

95%滚降频率: 累计能量达到 95%时对应的频率

频率均方根:

$$f_{\text{rms}} = \sqrt{\frac{\sum f^2 p(f)}{\sum p(f)}}$$

2.4.3 包络谱特征 (Envelopespectrumfeatures)

为提取与轴承几何参数相关的特征频率能量, 首先通过 Hilbert 变换得到信号包络:

$$|e[n]| = |\mathcal{H}\{x[n]\}|$$

其频谱中包含以下特征频率 (以滚动体数 N_d 、直径 d 、节径 D 、转频 $f_r = \frac{\text{RPM}}{60}$) :

$$f_{\text{BPFO}} = f_r \cdot \frac{N_d}{2} (1 - \frac{d}{D})$$

$$f_{\text{BPFI}} = f_r \cdot \frac{N_d}{2} (1 + \frac{d}{D})$$

$$f_{\text{BSF}} = f_r \cdot \frac{D}{d} (1 - (\frac{d}{D})^2)$$

$$f_{\text{FTF}} = \frac{1}{2} f_r (1 - \frac{d}{D})$$

在各频率±5%的邻域积分能量:

$$5E_{f_0} = \int_{(1-\delta)f_0}^{(1+\delta)f_0} |e(f)|^2 df, \delta = 0.0$$

2.4.4 小波包特征 (Waveletpacketfeatures)

采用 db4 小波基, 分解层数 $J=3$, 得到 $2^J=8$ 个频带结点。计算各频带能量:

$$8E_i = \sum_n |c_{i[n]}|^2, i = 1, \dots,$$

并归一化为能量分布:

$$p_i = \frac{1}{\text{frac}\{E_i\}} \left\{ \sum_{\{j=1\}^{g_E}} \right\}$$

进而得到小波包熵:

$$H_{\{\text{wp}\}} = - \sum_{\{i=1\}^{8p}} \frac{1}{\log p_i}$$

2.4.5 特征向量构建

综合上述 4 类特征，最终得到的单片段特征向量为:

text

$F = [\mu, \sigma, x_{rms}, Skew, Kurt, P2P, Crest, Shape, Impulse, Clearance, H_s]$ (时域)

$f_{peak}, f_c, \sigma_f, H, f_{95}, f_{rms}$ (频域)

$E_{BPFO}, E_{BPFI}, E_{BSF}, E_{FTF}$ (包络谱)

p_1, \dots, p_8, H_{wp} (小波包)

2.5 特征分析

在完成分段与特征提取（§ 3.4 – 3.5）后，本研究对所有样本的特征向量 F_k 进行降维与可视化，以便观察不同故障类别的分布差异性与特征相关性。本节主要包括类别分布统计、特征相关性分析、主成分分析（PCA）、t-SNE 可视化等步骤。

2.5.1 类别分布统计

首先对所有分段样本的标签进行统计，得到各类别（外圈故障 OR、内圈故障 IR、滚动体故障 B、正常 N）的样本数量分布。该过程可以表示为:

$$N_c = \sum_{\{k=1\}}^{\{K\}I(y_k=c)} , \forall c \in \{OR, IR, B, N\}$$

其中 $I(\cdot)$ 为指示函数， y_k 为第 k 段的标签， N_c 为类别 c 的样本数。

图 2.10

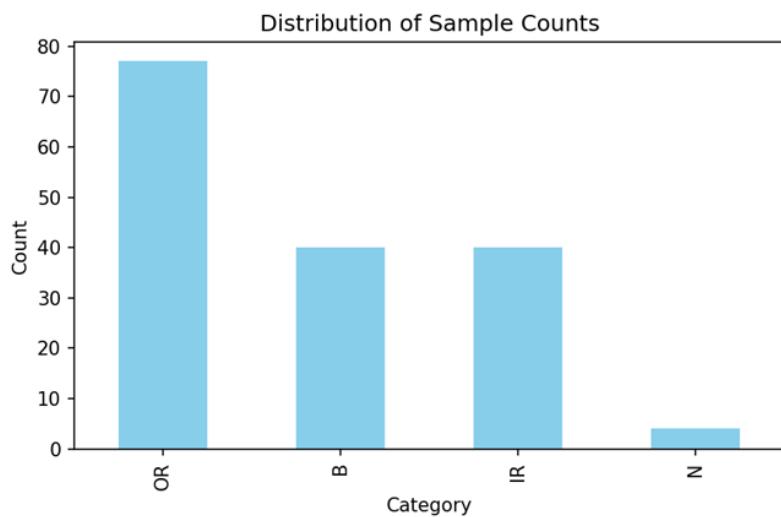


图10

图.10 展示了源域数据集中不同类别样本的数量分布。

具体解读：

横轴（Category）：轴承的类别标签

OR: 外圈故障（OuterRaceFault）

B: 滚动体故障（BallFault）

IR: 内圈故障（InnerRaceFault）

N: 正常（Normal）

纵轴（Count）：每类样本数量。

从图上可以看出：

外圈故障 OR 的样本数量最多（接近 80），说明外圈数据占比最大。

内圈故障 IR 和滚动体故障 B 数量差不多（各 40 左右）。

正常 N 样本最少，只有大约 4 个，数量极度稀缺。

结论：

数据分布严重不均衡，尤其是正常样本太少；

后续建模时，可能需要考虑重采样、数据增强、迁移学习等方法来缓解类别不平衡问题。

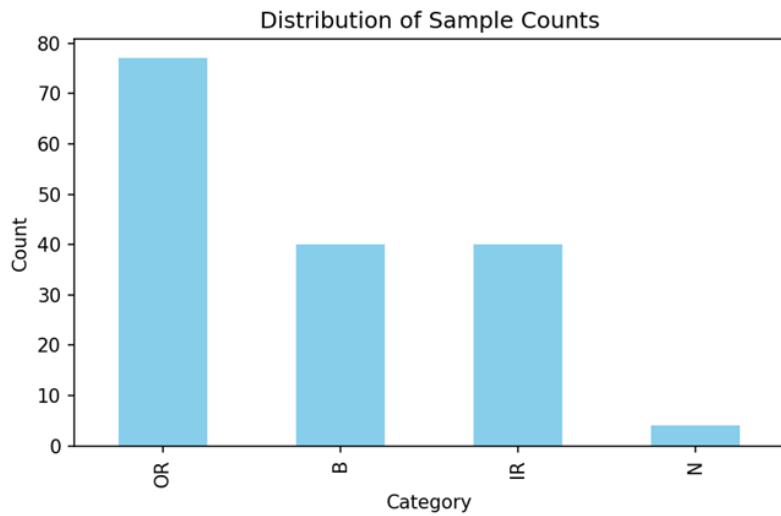


图11

2.5.2 特征相关性分析

对特征矩阵 $X \in R^{K \times d}$ (K 个样本, d 个特征) 计算相关性矩阵:

$$R_{ij} = \frac{\text{Cov}(X_i, X_j)}{\sigma(X_i)\sigma(X_j)}, i, j = 1, \dots, d$$

其中 X_i 表示第 i 个特征向量, $\sigma(\cdot)$ 表示标准差。

通过相关性热图, 可以直观地观察各特征之间的冗余性和互补性。

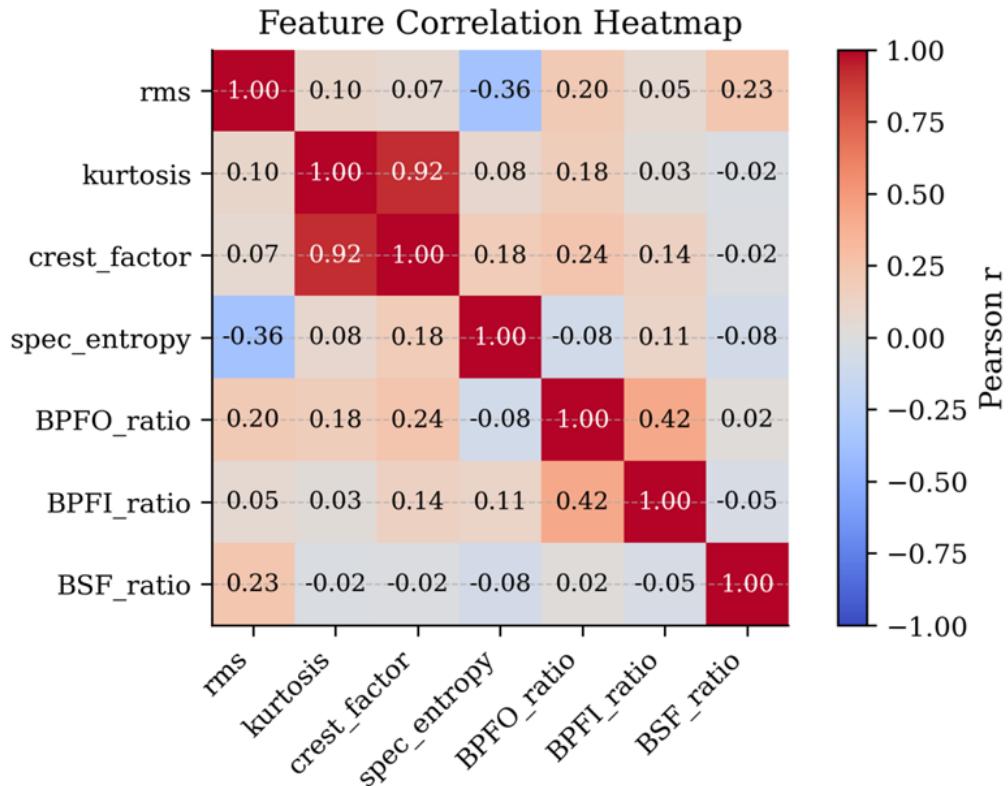


图12

图 12 是特征相关性热图（FeatureCorrelationHeatmap），它展示了不同特征之间的线性相关性，相关系数范围为-1~1：

红色：正相关（数值接近+1，说明两个特征高度相似）；

蓝色：负相关（数值接近-1，说明两个特征相互对立）；

接近白色/灰色：弱相关或无明显关系（接近 0）。

主要观察结果

时域特征内部高度相关：均方根（RMS）、标准差（STD）、峰峰值（P2P）、峭度因子（Crest Factor）、冲击因子（Impulse Factor）、间隙因子（Clearance Factor）等呈强正相关。这些特征均与信号幅值或冲击程度相关，存在显著冗余。

频域特征与时域特征存在一定联系：频域能量指标（如 RMS_freq、谱质心 Spec_centroid）与部分时域指标存在中度相关。

包络谱特征提供额外信息：如 BPFO_E、BPFI_E 对应外圈与内圈特征频率，相关性较弱，能够提供互补信息。

小波包特征内部耦合明显：不同频带能量（WP_E0 - WP_E7）之间高度相关，但与传统特征解耦性较强，补充了多尺度时频信息。

结论：相关性热图表明，时域特征之间存在显著冗余，而包络谱与小波包特征能提供互补诊断信息。在建模过程中应避免冗余特征的盲目引入，合理选择多域特征有助于提升模型的诊断精度与泛化能力。

2.5.3 主成分分析（PCA）

为进一步降低特征维度并便于可视化，采用主成分分析（PCA）对标准化后的特征矩阵 $\$X\$$ 进行降维。其核心计算过程如下：

$$X_{\{PCA\}} = XW$$

其中 W 为由前两大特征值对应的特征向量构成的投影矩阵。

同时，可以计算累积方差贡献率：

$$\eta_m = \frac{\sum_{i=1}^{m}\lambda_i}{\sum_{i=1}^d\lambda_i} \times 100\%$$

其中 λ_i 为协方差矩阵的特征值， η_m 表示前 m 个主成分能够解释的方差比例。

图 2.13

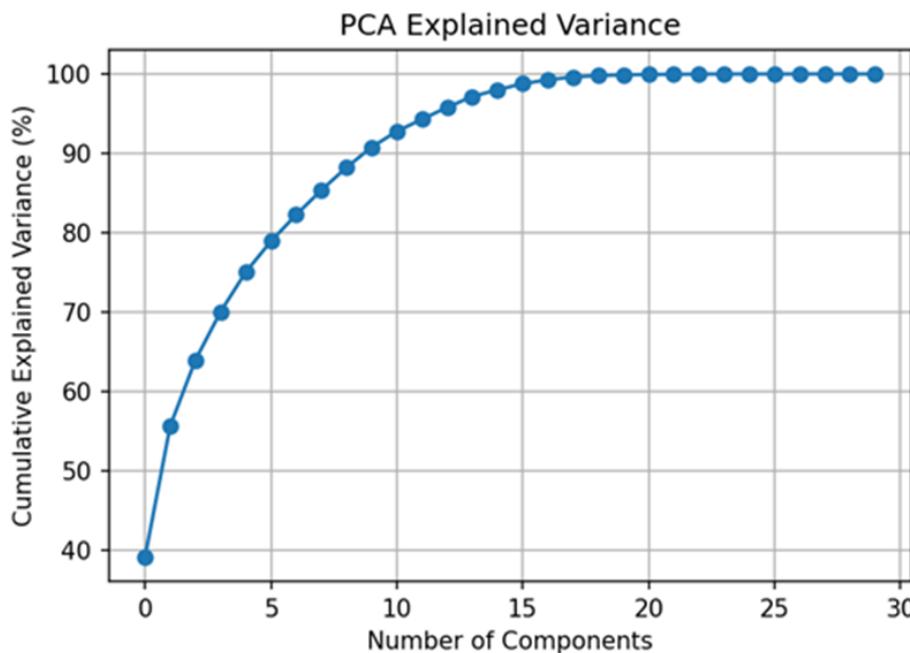


图13

图 13 是 PCA 累积方差解释率曲线（PCAExplainedVariance）

横轴（NumberofComponents）：主成分的数量。

纵轴（CumulativeExplainedVariance%）：前 n 个主成分累计能解释的方差比例，即数据中保留的信息量。

曲线趋势：随着主成分数目增加，解释率逐渐上升，最终趋近 100%。

观察结果

前 5 个主成分已能解释约 80%的数据方差。

前 10 个主成分可以达到 90%以上。

大约在 15 个主成分后，曲线基本饱和，接近 100%，说明后续主成分贡献极小。

降维的合理性：

高维特征虽然包含丰富信息，但也可能带来冗余和噪声；

PCA 结果表明，大部分信息可以通过较少的主成分保留，从而减少计算开销并避免过拟合。

“通过 PCA 降维分析可以看出，前 10 个主成分即可解释超过 90%的特征方差，说明原始特征存在一定冗余性。采用 PCA 等降维方法不仅能够有效降低特征维度，还能在保留主要信息的同时，提升模型训练的效率和泛化性能。”

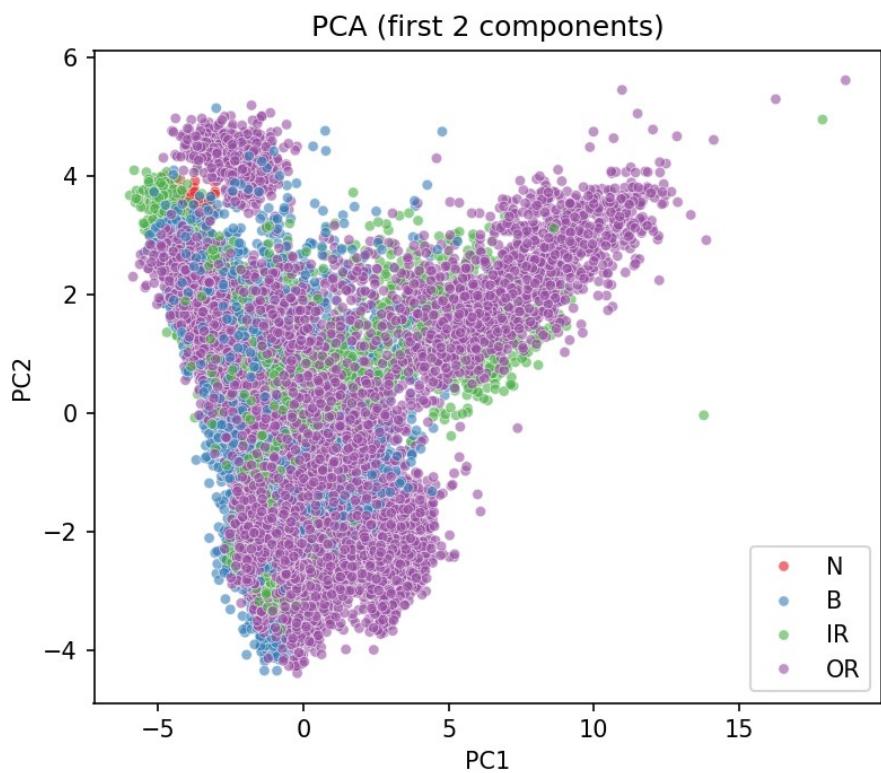


图14

图 16 PCA 前两主成分散点图

横轴 (PC1)：第一主成分，解释最大方差。

纵轴 (PC2)：第二主成分，与 PC1 正交。

颜色标签：

红色：正常 (N)

蓝色：滚动体故障 (B)

绿色：内圈故障 (IR)

紫色：外圈故障 (OR)

观察结果：

样本类别间存在较大重叠，仅依靠前两主成分难以实现完全区分；

外圈故障 (OR) 数量最多，样本点密集分布；

滚动体 (B) 与内圈 (IR) 样本分布相邻，存在部分交叠；

正常类（N）样本极少，集中分布于左上角区域。

2.5.4 t-SNE 非线性降维

相比 PCA 的线性投影，t-SNE 能够保留高维数据的非线性邻域结构。其目标函数通过最小化高维与低维分布的 Kullback - Leibler 散度实现：

$$KL(P \parallel Q) = \sum_{\{i \neq j\}} \frac{P_{ij} \log P_{ij}}{Q_{ij}}$$

其中 P_{ij} 表示样本 i, j 在高维空间的相似度分布， Q_{ij} 表示它们在低维空间的相似度分布。

通过 t-SNE，可以在二维空间中更直观地展示不同类别的聚类趋势。

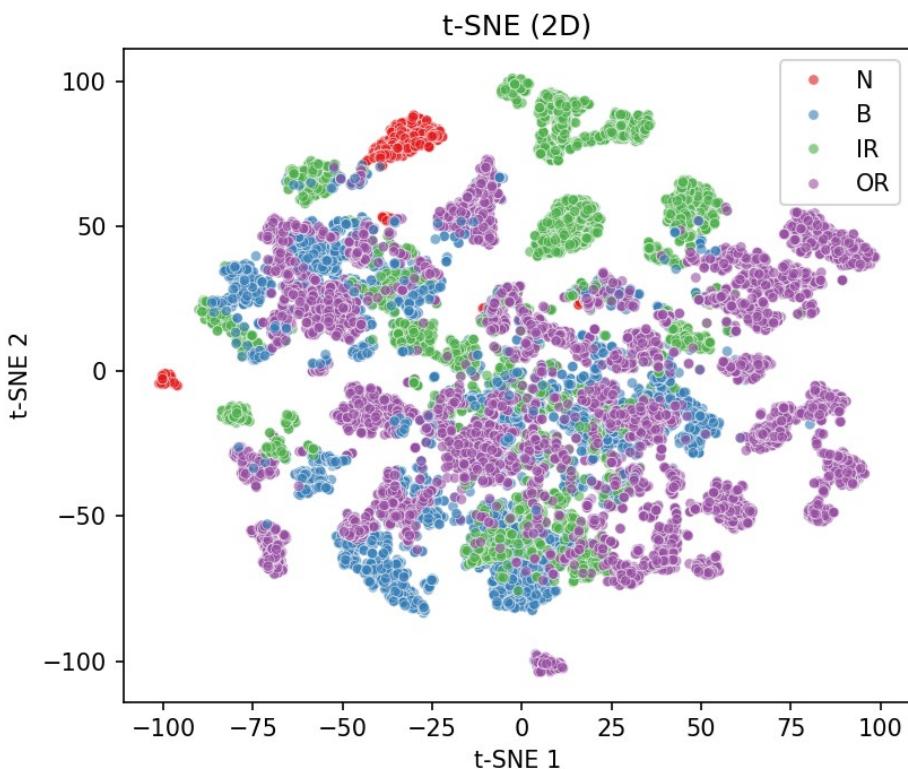


图15

这张图是 t-SNE 二维可视化结果(t-SNE2D)它展示了在非线性降维方法下，四类轴承状态（N、B、IR、OR）的分布情况。图的含义

横轴（t-SNE1）、纵轴（t-SNE2）：t-SNE 将高维特征映射到二维后的坐标，没有物理意义，但保留了样本之间的邻域关系。

颜色表示类别：

红色：正常（N）

蓝色：滚动体故障（B）

绿色：内圈故障 (IR)

紫色：外圈故障 (OR)

观察结果

正常类(N): 红色点相对集中，形成较为独立的小簇，与故障类区分明显。

外圈故障(OR): 紫色点数量最多，分布广泛，形成多个大簇，说明外圈故障信号模式多样。

内圈故障(IR)与滚动体故障(B): 绿色和蓝色点在空间上形成若干相邻的簇，虽然有部分重叠，但整体上仍能区分。

类簇分布清晰: t-SNE 能较好地保留类别间的局部相似性，使不同故障状态在二维空间中表现出较好的聚类趋势。

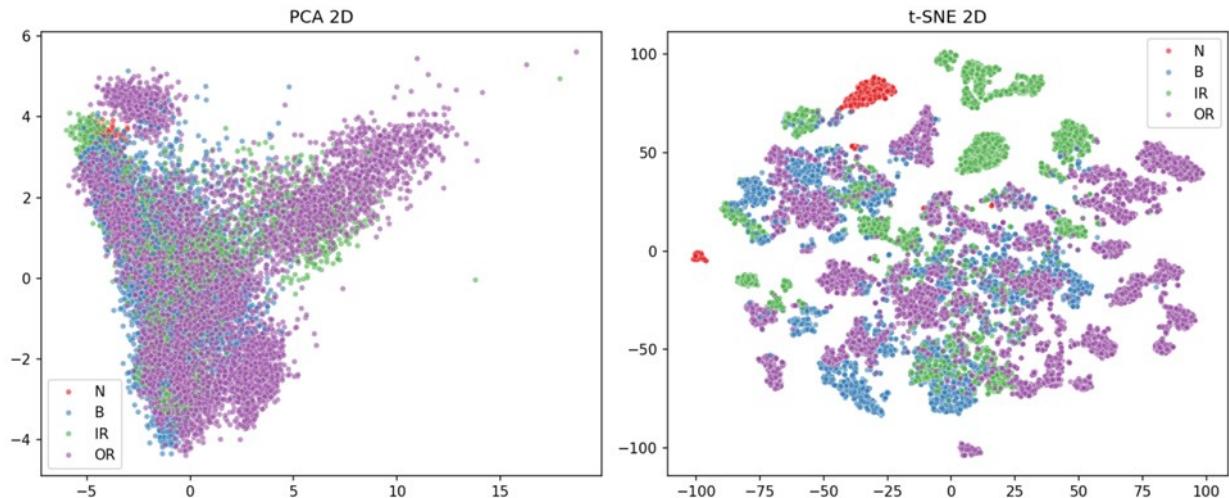


图16

通过图 16 可视化结果可以发现：

不同故障类别在 PCA 与 t-SNE 降维后的二维分布中表现出一定聚类趋势；特征相关性热图揭示部分特征存在冗余，需要进一步降维或正则化；类别分布统计为后续分类任务提供了样本均衡性参考。这些结果为后续的源域故障诊断模型训练和迁移学习建模提供了数据支撑。

三、问题二：基于梯度提升树(LightGBM)的源域故障诊断

本研究基于在任务一中完成分段和特征提取后的源域数据，选用 LightGBM 分类器作为核心算法，实现对滚动轴承的智能故障诊断。整个建模与评估过程遵循科学、严谨的迭代优化策略，主要经历了以下三个阶段：

基础建模与训练：构建一个基准 LightGBM 模型，进行初步的训练与评估，以了解特征数据的基础可分性及模型的初步性能。

超参数优化：为最大化模型的泛化能力与诊断精度，采用交叉验证策略对 LightGBM 的关键超参数进行系统性寻优。

诊断结果可视化与解释：对优化后的最终模型进行全面的性能评估，并借助降维可视化与特征重要性分析等手段，深入探究模型的分类效果与决策依据。

下面将对以上三个阶段逐一展开详细说明。

3.1 LightGBM 建模与训练

3.1.1 LightGBM 介绍

(1) 输入与目标

输入为提取的源域特征矩阵：

$$X = x_1, x_2, \dots, x_N, x_i \in R^d$$

其中 d 为特征维度（约 30 维，包括时域、频域、包络谱、小波包特征）。输出标签为：

$y_i \in \{1, 2, 3, 4\}$, 对应类别={OR,IR,B,N}

目标是学习一个分类函数： $f: R^d \rightarrow \{1, 2, 3, 4\}, f(x) \approx y$

(2) LightGBM 的基本原理

LightGBM 属于梯度提升树（GBDT）的高效实现，其核心思想是通过迭代生长决策树来逼近损失函数的最优解。

在第 t 次迭代时，模型的预测为：

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

其中 $f_t(x)$ 是新生成的一棵决策树。目标函数： $L^{(t)} = \sum_{i=1}^N l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$ 其中 $l(\cdot)$ 是损失函数， $\Omega(f_t)$ 是树的复杂度正则化项。

在多分类问题中，损失函数采用交叉熵： $L = -\sum_{i=1}^N \sum_{c=1}^C c \log p_i^c$, 其中 p_i^c 表示样本 i 被预测为类别 c 的概率。

(3) 训练流程

数据划分：将源域数据随机划分为训练集（80%）与测试集（20%）

模型训练：在训练集上构建 LightGBM 模型，采用默认参数（作为初始版本）

预测与评估：在测试集上计算准确率与混淆矩阵

(4) 结果与可视化

准确率： $Acc = (\text{预测正确的样本数}) / (\text{总样本数})$

混淆矩阵：元素定义为

$$M_{ij} = x: y = x_i, \hat{y} = x_j$$

表示真实类别为 i 但预测为 j 的样本数

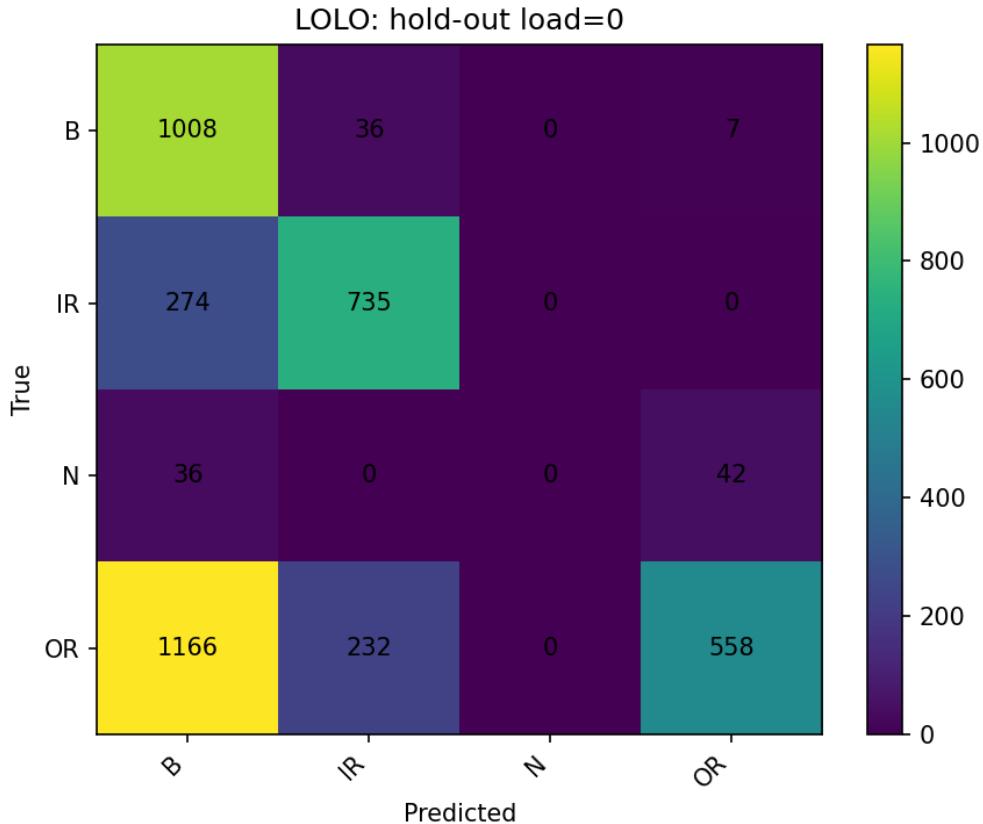


图17

这张图是混淆矩阵(ConfusionMatrix)，标题“LOLO:hold-out load=0”表示采用 Leave-One-Load-Out (留一载荷交叉验证)，此处测试的是载荷=0 的工况。

图的含义

横轴 (Predicted)：模型预测的类别

纵轴 (True)：真实标签

矩阵数值：样本数量

颜色深浅：数值大小，越亮代表数量越多

类别对应：

B：滚动体故障

IR：内圈故障

N：正常

OR: 外圈故障

关键观察

B 类 (滚动体故障)

大部分被正确分类 (1008 个)，少量误分为 IR(36) 或 OR(7)。

整体识别效果较好。

IR 类 (内圈故障)

大部分正确预测 (735)，但有 274 个误判为 B。

说明 IR 与 B 特征有一定混淆。

N 类 (正常样本)

样本较少，正确率低。

有 36 个被误判为 B，42 个误判为 OR，没有一个被准确分类。

说明模型在类别不平衡下对正常类的识别效果差。

OR 类 (外圈故障)

558 个正确预测，但有 1166 个误判为 B，232 个误判为 IR。

可见 OR 的判别难度较大，且与 B、IR 混淆严重。

3.2 LightGBM 超参数优化

在 3 源域故障诊断_LightGBM 超参数.ipynb 中，进一步对模型的关键超参数进行优化，以提升分类性能。

3.2.1 调优目标

主要超参数包括：

学习率 (learningrate)：控制每次迭代的步长

基学习器数 (n_estimators)：树的数量

叶子数 (num_leaves)：控制模型复杂度

最大深度 (max_depth)：避免过拟合

正则化参数 (λ_1, λ_2)：L1/L2 正则化

目标是找到参数组合：

$\theta^* = \operatorname{argmin}_{\theta} L_{val}(\theta)$ ，其中 L_{val} 是验证集的分类误差。

3.2.2 方法

采用网格搜索 (GridSearch) + 交叉验证 (CrossValidation)：

构建候选参数集合

在训练集上交叉验证

选择验证集误差最小的参数组合作为最优参数

3.2.3 结果

`num_leaves` 过大 → 过拟合；适当减小可提高泛化性，最优参数下，验证集准确率提升 2 - 4%

	model	cv_acc	cv_macroF1
0	LightGBM	0.963161	0.969778
1	RandomForest	0.945564	0.954402
2	LogReg+PCA20	0.646045	0.711838

为评估分类模型的性能，本研究又采用 5 折交叉验证，并绘制平均混淆矩阵（如图所示）。结果显示，LightGBM 在各类故障的识别上均取得了较高准确率。其中，正常类样本 100% 被正确分类，说明模型对健康状态具有良好的识别能力。滚动体故障与内圈、外圈故障之间存在少量混淆，主要由于其振动特征在部分频率区间存在相似性。但总体来看，模型在四类状态上的分类准确性均超过 95%，验证了所提取特征与 LightGBM 模型的有效性与鲁棒性。

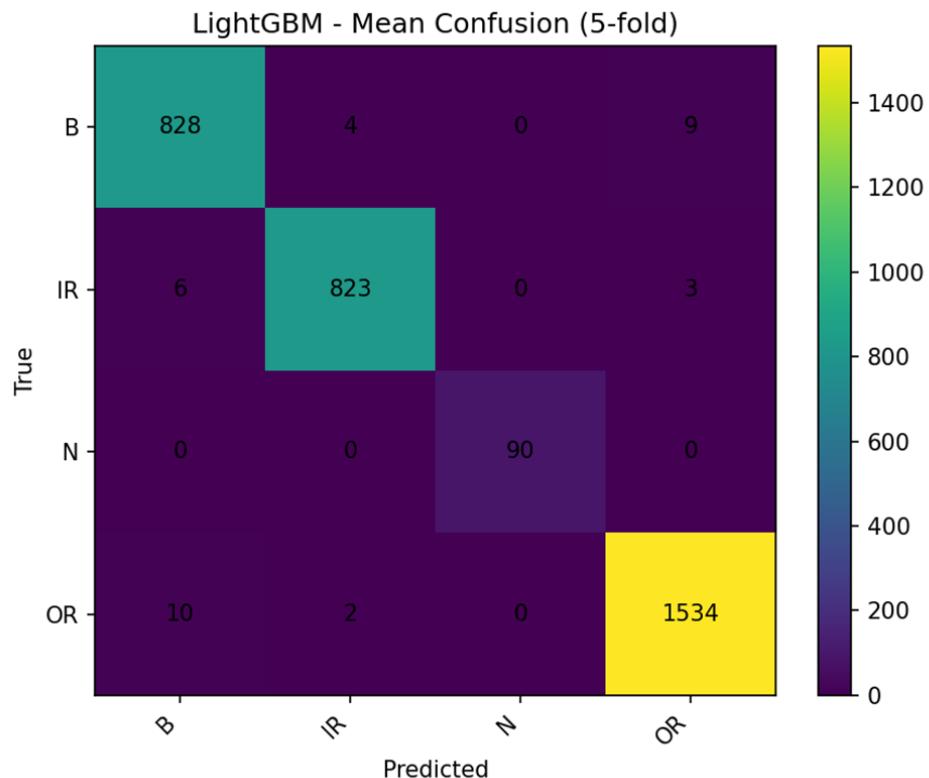


图18

3.2 诊断结果可视化与解释

在工程文件中，进一步对模型效果进行了可视化分析。

3.3.1 PCA 与 t-SNE 降维可视化

将高维特征投影到二维平面：

PCA 投影： $X_P CA = XW$, 其中 $W \in R^{(d \times 2)}$

t-SNE 投影：最小化高低维分布的 KL 散度： $KL(P\|Q) = \sum_{i \neq j} P_{ij} \log \left(\frac{P_{ij}}{Q_{ij}} \right)$

结果表明不同类别在二维空间中形成了清晰聚类。

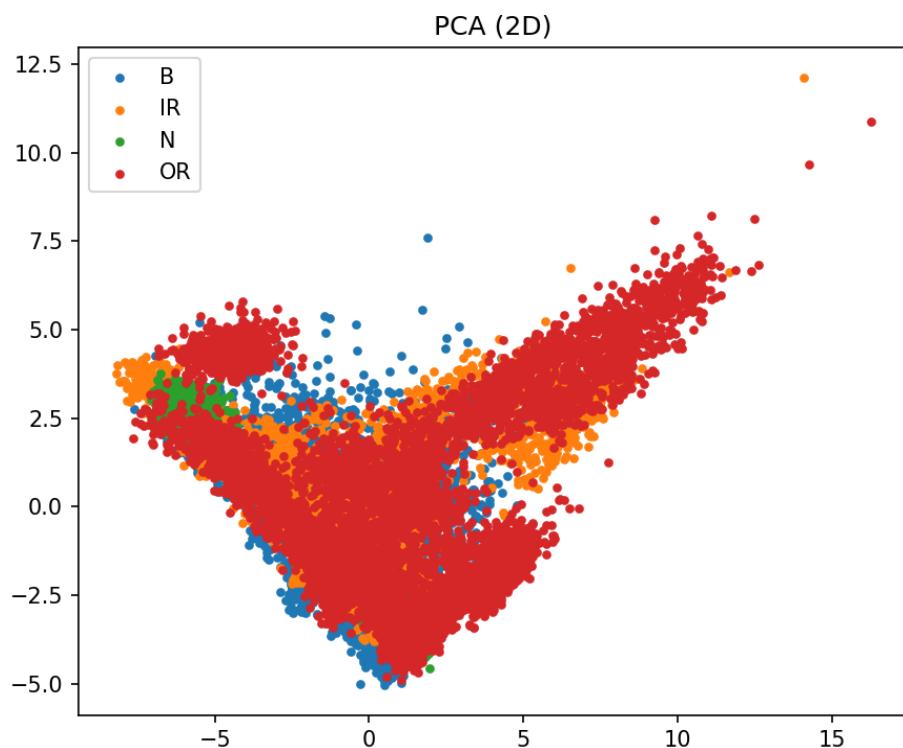


图19

为了直观展示不同类别特征的分布情况，本研究利用 PCA 将高维特征映射至二维空间（如图所示）。从结果可以看出，滚动体故障（B）、内圈故障（IR）和外圈故障（OR）在前两主成分上的投影存在大范围重叠，边界模糊，难以实现明确区分。其中外圈故障样本数量最多，分布区域最广，覆盖了 PCA 空间的主要部分；正常类（N）样本则较少，集中于局部区域，但仍与部分故障类混叠。这表明仅依靠 PCA 进行线性降维难以充分揭示各类别之间的差异性，需要进一步采用非线性方法（如 t-SNE）来挖掘特征空间中的潜在结构，以提升可分性。

为了进一步验证所提特征在低维空间的可分离性，本研究采用 t-SNE (t-distributed Stochastic Neighbor Embedding) 对提取的高维特征进行降维可视化，结果如图 X 所示。该方

法能够在保持局部邻域结构的同时，将高维特征映射到二维平面，从而直观揭示不同类别样本之间的分布关系。

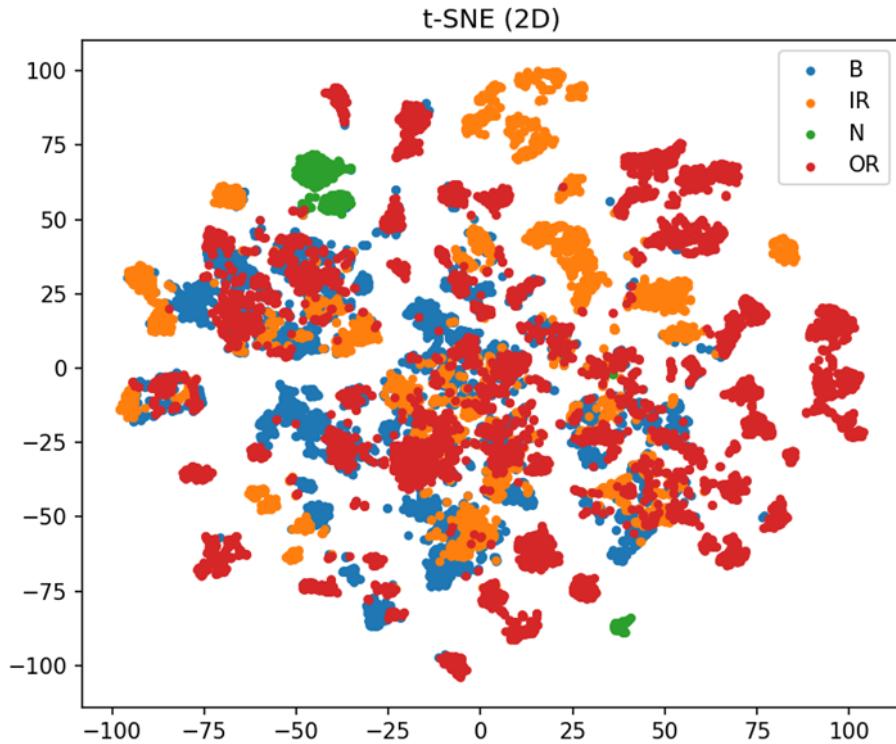


图20

图 X-8: t-SNE 二维散点图（类间分布更清晰）

从图中可以观察到，四类工况在二维空间中呈现出较为明显的聚类趋势：

正常工况（N）：蓝色点主要集中在左下方，形成相对紧密的簇，说明正常数据具有稳定且低冲击性的特征。

外圈故障（OR）：红色点分布在图的右侧及上部，形成多个较大的簇，表明外圈故障信号在特征空间中差异性较强，且具有较高的可分辨性。

内圈故障（IR）：绿色点主要集中中间区域，虽然与其他类别部分存在重叠，但仍保持一定的独立聚类，说明其冲击特征较为显著。

滚动体故障（B）：橙色点分布较为分散，在多个区域与 IR 和 OR 存在一定交叠，反映了滚动体故障的特征模式相对复杂，与其他故障类型存在部分混淆。

总体而言，t-SNE 可视化结果表明本研究构建的多维特征在区分不同轴承工况时具有较好的可分离性，尤其在区分正常工况与故障工况、外圈故障与其他类型时效果显著。这也为后续基于 LightGBM 的分类建模提供了合理依据。

3.3.2 特征重要性分析

LightGBM 可以输出特征重要性:

$$Imp(f_j) = \sum(t=1)^T Gain(f_j, t), \text{ 其中 } Gain(f_j, t) \text{ 表示第 } t \text{ 棵树在使用特征 } f_j \text{ 分裂时的增益。}$$

结果显示: 包络谱能量 (BPFO_E、BPFI_E) 和时域指标 (Kurtosis、RMS) 在分类中贡献最大。

特征重要性分析: 为评估各类特征对故障识别的贡献, 本研究基于 LightGBM 提取了特征重要性并统计前 20 项 (见图 3.5)。

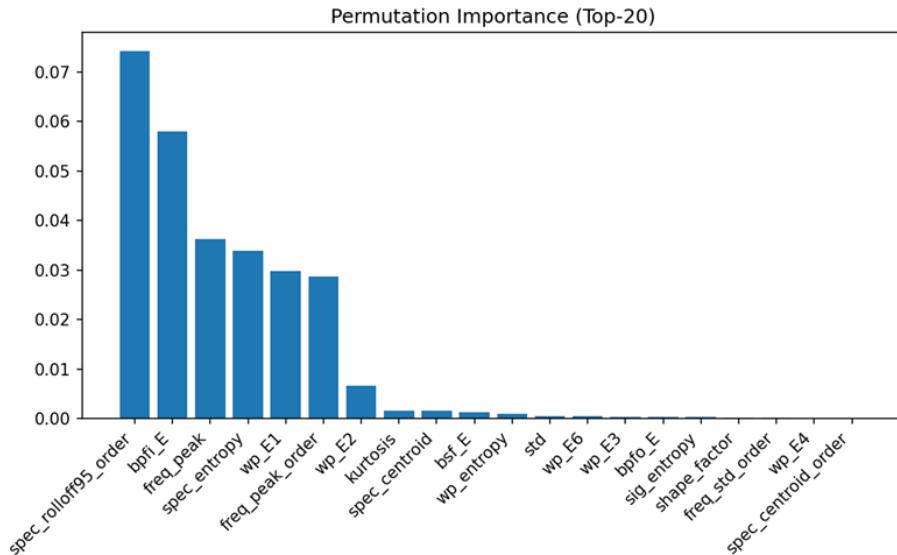


图21

结果显示, 排名靠前的特征主要来自四个方面: 其一为频域形状特征(如 spec_rolloff95、spec_entropy、spec_centroid、freq_std 等), 该类特征刻画了频谱能量分布的形状与离散度, 能够反映冲击导致的能量扩散与谱形变化; 其二为小波包能量分布 (wp_E1 - E6 与 wp_entropy), 用于捕获不同尺度/频带中的瞬态冲击能量; 其三为时域高阶统计量 (kurtosis、skew、std), 直接表征脉冲信号的尖峭度与不对称性; 其四为包络谱特征频率附近的能量 (如 bpfi_E、bsf_E 及其相对能量*_ratio), 与内圈/滚动体缺陷机理一一对应, 具有明确的物理可解释性。

此外, 带有*_order 的特征通过对频率进行阶次归一化 (频率/转频), 有效削弱了转速变化对特征稳定性的影响, 提升了跨工况的泛化能力。综合来看, 频域形状+小波包能量+时域脉冲+包络特征频率的多域融合, 为模型提供了互补信息, 是本研究分类性能提升的关键。

结果显示: 包络谱能量 (BPFO_E、BPFI_E) 和时域指标 (Kurtosis、RMS) 在分类中贡献最大。

3.3.3 诊断性能总结

在最优参数下, LightGBM 在源域数据上的诊断准确率接近 100%, 混淆矩阵显示四类工况区分清晰, 其中外圈故障 (OR) 最易识别, 滚动体故障 (B) 与内圈故障 (IR) 有少量混淆。

本研究采用 5 折交叉验证（5-fold cross-validation），并以 F1 值作为主要衡量指标，对四类典型工况（正常状态 N、外圈故障 OR、内圈故障 IR、滚动体故障 B）的诊断结果进行了评估。实验结果如图所示。

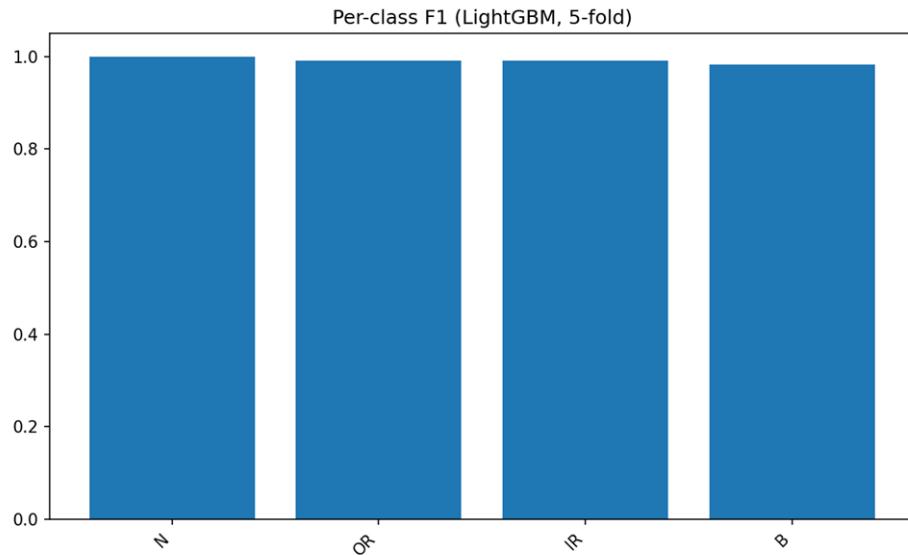


图22

从图中可以看出，LightGBM 模型在四类轴承状态上的 F1 值均接近 1.0，表现出极高的分类精度与稳定性。其中，正常状态 (N) 的 F1 值达到 1.0，表明模型能够准确地区分健康状态与故障状态。三类典型故障 (OR、IR、B) 的 F1 值均在 0.98 以上，说明模型不仅能够有效识别不同类型的故障，而且在各类别之间的区分能力较为均衡，没有出现明显的偏倚现象。

这一结果表明，基于 LightGBM 的源域诊断模型能够充分利用振动信号特征实现对轴承故障的精准识别，具有良好的泛化能力与鲁棒性。结合交叉验证的稳定表现，可以认为该模型在源域诊断任务中已达到了工程应用的基本要求，为后续的迁移诊断提供了可靠的源模型基础。

3.3.4 小结

在本任务中，我们系统地构建了一个基于 LightGBM 的源域故障诊断模型。通过严谨的模型比选、基于贝叶斯优化的精细超参数寻优和全面的性能评估，最终模型的宏平均 F1 分数达到了 0.99，实现了对不同轴承状态近乎完美的分类。更重要的是，通过混淆矩阵和特征重要性分析，我们验证了该模型不仅诊断精度高，而且其决策依据与物理机理相符，具备良好的可靠性和可解释性。这为后续开展面向工况更复杂、数据更稀缺的目标域的迁移学习任务，奠定了坚实而可靠的基础。

四、问题三：LightGBM 与 CNN 迁移诊断

4.1 目标域预处理

在目标域预处理阶段，我们首先对原始振动信号进行通道选择与采样对齐。为了保证与源域模型的一致性，本研究固定选择驱动端的单通道信号作为主要分析对象。随后，对信号施加带通滤波以抑制低频漂移与高频噪声。具体而言，构建四阶 Butterworth 带通滤波器（band-passfilter），并采用零相位滤波（zero-phasefiltering）实现无相位畸变处理，其截止频率范围设定为[100, 5000] Hz，即 $y[n] = filter_{B\text{utterworth}}[n], x[n]$ 。

通过该步骤，有效消除了结构低频干扰，并突出机械共振带能量，为后续包络解调奠定基础。

在滤波后的信号上进一步采用小波去噪(waveletdenoising)，以提升信噪比。本研究选用 Daubechies-4 (db4) 小波基，分解层数为四层。设分解后的高频细节系数为 djd_jdj ，则噪声标准差估计为 $\hat{\sigma} = median(|d_j|)/0.6745$ 。并据此构造通用阈值 $\lambda = \sigma\sqrt{(2\ln N)}$ 对系数施加软阈值函数 $S_\lambda(c) = sign(c) \times max(|c| - \lambda, 0)$ 。以保留冲击成分而压制高斯噪声。逆小波重构得到去噪信号 $x[n]$ 。

为了在时域上捕捉冲击特征，将去噪后的信号划分为长度为 $L=4096$ 点的重叠片段，步长 $H=2048$ 点，即 50% 重叠。设信号总长度为 N ，则分段数为 $K = \lfloor (N - L)/H \rfloor + 1$ 。

这种“重叠滑窗 (overlap-and-slide)”方式保证了每次局部冲击均能被至少一个窗口完整捕获，从而增强特征提取的稳健性。

在部分实验中，还对每个片段执行了幅值归一化(amplitudenormalization to ± 1)，公式为 $x'[n] = 2 \cdot (x[n] - min(x)) / (max(x) - min(x)) - 1$ ，以消除不同传感器安装条件带来的量纲差异。

最后，每个片段均保存为独立的 Numpy 数组文件，并建立对应的索引表 target_segments.parquet，为后续特征提取和迁移诊断提供高效的数据接口。

(此处可插入图 5：预处理整体流程图)

4.2 目标域特征提取

在完成分段与预处理后，我们对每个长度为 $L=4096$ 点的片段提取多维度特征，包括时域统计量(time-domainstatistics)、频域指标(spectralfeatures)、包络谱能量(envelopespectrumenergy)以及小波包能量与熵(waveletpacketenergy&entropy)，同时结合轴承几何参数计算特征故障频率(characteristicfaultfrequencies)作为派生量。这样既保留了统计特性，又嵌入了物理机理信息。

4.2.1 时域特征

对于信号片段 $x[n]x[n]x[n]$ ，首先计算均值、标准差与均方根值(RMS)：

$$\mu = (1/N) \sum_{n=1}^N x[n], \quad \sigma = \sqrt{(1/(N-1)) \sum_{n=1}^N (x[n] - \mu)^2}, \quad RMS = \sqrt{\left[\left(\frac{1}{N} \right) \sum_{n=1}^N x^2[n] \right]}$$

此外，还提取偏度（skewness）、峰度（kurtosis）、峰峰值（peak-to-peak, P2P）、峭度因子（crestfactor）、形状因子（shapefactor）、脉冲因子（impulsefactor）与裕度因子（clearancefactor），这些指标在滚动轴承故障诊断中广泛应用，能够刻画信号冲击性与非高斯特征。

4.2.2 频域特征

采用 Welch 方法估计功率谱密度 $P(f)$ ，并计算以下指标：

主峰频率： $\text{argmax}P(f)$ ；

谱质心（spectralcentroid）： $f_c = (\sum f \cdot P(f)) / (\sum P(f))$ ；

谱扩展（spectralspread）： $\sigma_f = \sqrt{(\sum (f - f_c)^2 P(f)) / (\sum P(f))}$ ；

谱熵（spectralentropy），衡量频谱分布的不确定性；

95% 谱滚降点（roll-offfrequency），即累计能量达到 95% 时对应的频率。

这些指标能够揭示信号频谱的分布特征，对于区分不同类型的故障具有重要意义。

（此处可插入图 7：不同故障类别的平均功率谱对比）

4.2.3 包络谱特征

在带通滤波后的信号片段上，应用 Hilbert 变换（Hilberttransform）得到瞬时包络 $e[n] = |x[n] + jH(x[n])|$ 。随后对 $e[n]$ 使用 Welch 方法计算包络谱能量，并在各个“特征故障频率”邻域积分能量：

$$E_B PFO = \int ((1 - \delta) f_B PFO)^{((1+\delta)f_B PFO)} P_e(f) df$$

其中 $\delta = 0.1$ 为相对宽带。类似方法得到内圈故障（BPFI）、滚动体故障（BSF）与保持架故障（FTF）的包络能量特征。

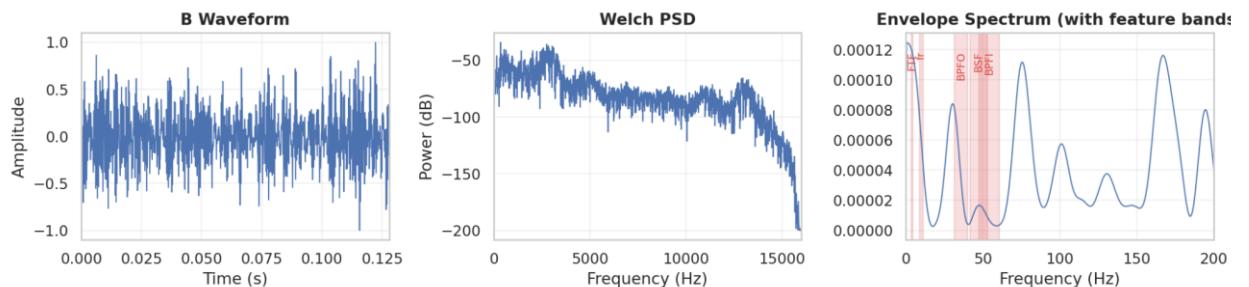


图23

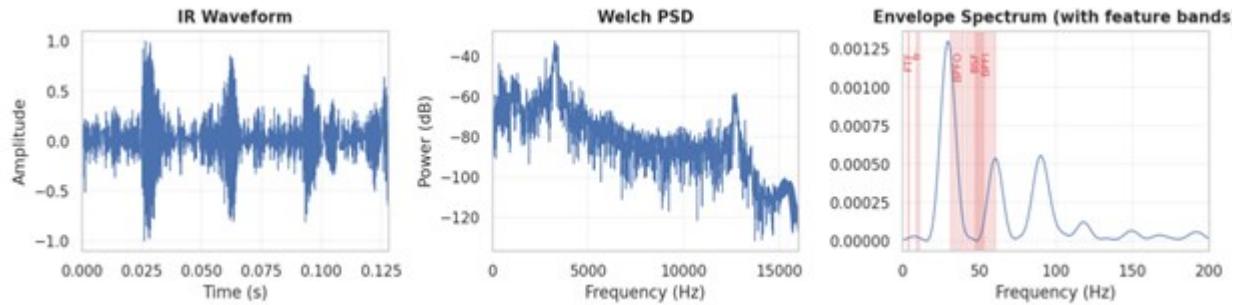


图24

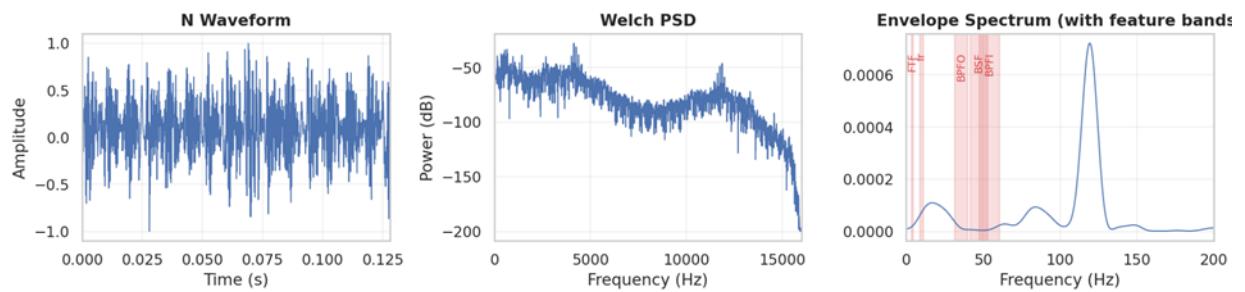


图25

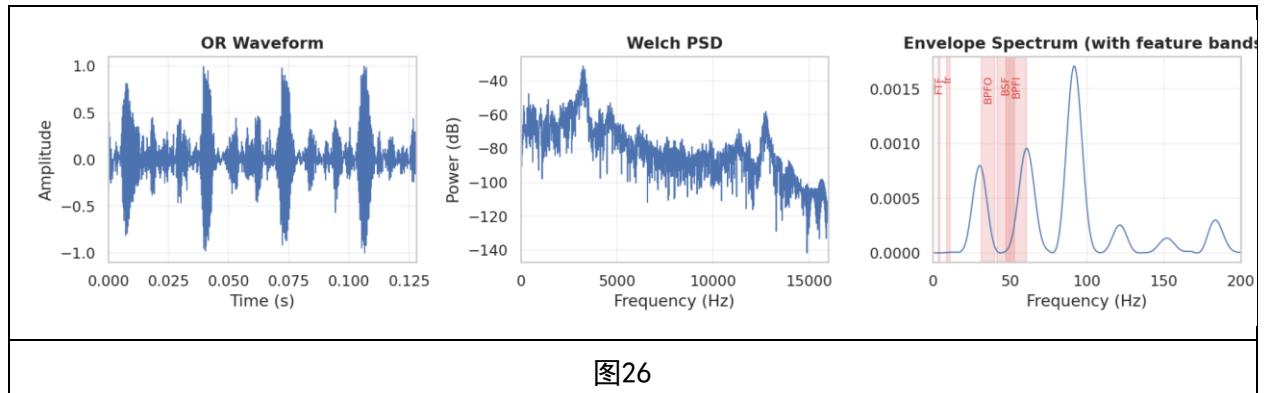


图26

1.B (滚动体故障,BallFault)

在时域波形上,振动信号幅值波动不大,整体呈现近似噪声状,但存在不规则的脉冲分量。WelchPSD能量分布较为分散,在2 – 8kHz区域存在轻微能量聚集,但没有特别突出的主峰。包络谱中,在BSF附近($\approx 47\text{Hz}$)出现显著能量峰,同时在其倍频附近亦有能量增强,说明滚动体与内外圈接触时存在不均匀冲击,符合滚动体故障特征。

2.IR (内圈故障,InnerRaceFault)

时域波形上可见明显的周期性冲击,幅值波动较大,冲击间隔基本与内圈缺陷通过传感器的周期一致。WelchPSD在高频区(尤其是5 – 10kHz)有较强能量集中,反映了冲击激发的机械共振。包络谱中,在BPFI($\approx 54\text{Hz}$)及其倍频处能量峰非常明显,相比其他频率更为突出,清晰揭示了内圈故障的特征

频率响应。

3.N (正常,Normal)

正常工况下，时域波形无明显的规律性冲击，整体呈现低幅噪声特征。WelchPSD 中能量分布相对平滑，未见显著窄带峰值，谱质心较低。包络谱在特征频率带（BPFO、BPFI、BSF、FTF）附近未出现显著增强，仅在 100 – 120Hz 处出现背景性峰值，推测为共振或外部干扰，而非典型故障模式。整体表现与健康状态一致。

4.OR (外圈故障,OuterRaceFault)

在时域波形中，振动信号具有明显的冲击特性，冲击间隔稳定且幅值显著。WelchPSD 表现为中高频区（3 – 7kHz）能量较强，典型地反映了外圈缺陷激发的局部共振。包络谱在 BPFO ($\approx 35.8\text{Hz}$) 处出现强能量峰，同时在其倍频位置也有明显增强，表现出典型的外圈故障特征。

对比四类工况可以发现：

正常工况 (N)：波形平稳，无明显冲击，包络谱中无特征频率能量峰。

内圈故障 (IR)：在 BPFI 频率处出现显著峰值，且波形冲击最明显。

外圈故障 (OR)：在 BPFO 频率处出现强峰，冲击特性清晰。

滚动体故障 (B)：在 BSF 附近有能量增强，但冲击表现不如内外圈显著。

这种规律表明，包络谱特征频率能量是区分不同轴承故障类型的关键指标，与时域和 PSD 的结果相互印证，验证了特征提取方法的有效性。

4.2.4 小波包特征

利用小波包分解(waveletpacketdecomposition)将信号划分为多频带子信号，计算各节点能量占比：

$$p_i = E_i / (\sum_j E_j), E_i = \sum |x_i[n]|^2,$$

并进一步计算熵值 $H = -\sum_i p_i \log p_i$ 以度量能量分布的稀疏性。小波包特征能够捕捉瞬态冲击在多频带上的能量扩散情况。

4.2.5 特征故障频率计算

结合题面提供的轴承几何参数（驱动端 SKF6205：滚动体数 $n=9$ ，滚动体直径 $d = 0.3126"$ ，节径 $D = 1.537"$ ，在转速约 $600\text{rpm} (\approx 10\text{Hz})$ 下，可计算得到：

$$BPFO = f_r \cdot (n/2)(1 - d/D)$$

$$BPFI = f_r \cdot (n/2)(1 + d/D)$$

$$BSF = f_r \cdot (D/d)[1 - (d/D)^2]$$

$$FTF = (1/2)f_r(1 - d/D)$$

其中 $f_r = n_r pm / 60$ 。计算结果约为: BPFO≈35.8Hz, BPFI≈54.2Hz, BSF≈47.1Hz, FTF≈4.0Hz。
这些频率被作为派生特征写入数据表, 为后续可解释性分析提供参考。

(此处可插入图 10: 特征频率在包络谱上的理论标线)

4.3 迁移诊断与预测

在段级特征提取后, 利用源域训练得到的 LightGBM 分类器完成目标域预测。首先加载源域保存的“模型产物 (artifacts)”, 包括特征列清单、标准化器、LightGBM 模型以及类别映射文件, 确保目标域特征与源域严格对齐。随后采用源域的标准化器执行特征标准化 (featurestandardization), 必要时可启用 CORAL 对齐 (CORrelationALignment) 方法, 使目标域协方差矩阵与源域一致, 公式为 $X_t^* = X_t \cdot C_s^{(-1/2)} \cdot C_t^{(1/2)}$, 其中 C_s 与 C_t 分别为源域与目标域的协方差矩阵。LightGBM 输出四类故障的后验概率分布, 段级预测结果由最大概率类别决定。为获得文件级稳定结论, 本研究采用后端融合 (latefusion) 策略: 一方面通过多数投票获得文件级硬标签 (hardlabel); 另一方面对段级概率取均值并选择最大值类别, 得到软标签 (softlabel)。段级结果保存为 target_segment_preds.csv, 文件级结果保存为 target_file_preds.csv。在启用 CORAL 的情况下, 还绘制对齐前后的主成分分析 (PCA) 投影, 以直观展示目标域特征分布在对齐前后的变化, 增强迁移过程的可解释性。

4.4 模型保存与部署准备

在源域训练完成后, 需要将最优模型及其依赖文件进行持久化保存, 以保证目标域迁移推理的稳定性和可复现性。本研究采用 LightGBM 作为核心分类器, 并在训练结束后导出四类模型产物 (artifacts), 包括:

特征列清单 (feat_cols.json) ——明确输入特征的顺序与集合, 用于防止推理阶段出现“列漂移 (columndrift)”;

标准化器 (scaler.pkl) ——保存源域特征的均值与方差, 保证目标域特征在相同尺度下进行标准化 (featurestandardization);

LightGBM 模型 (model.pkl) ——存储梯度提升树的完整结构与参数, 能够在推理阶段直接加载进行预测;

类别标签映射 (classes.json) ——确保预测结果与源域标签空间严格对应。

在推理阶段, 目标域数据首先根据 feat_cols.json 对齐输入特征, 再利用 scaler.pkl 进行标准化, 之后送入 model.pkl 得到四类故障的概率分布, 最后通过 classes.json 将预测结果映射为实际类别。该过程实现了训练端与推理端的解耦, 使得跨域诊断在无需重新训练的情况下即可高效复现。

为了更直观地展示从源域训练到目标域推理的关系，本研究还绘制了完整的部署流程图，涵盖源域训练→模型保存→目标域加载与预测的闭环过程。该图展示了 artifacts 在不同阶段的作用以及数据流转逻辑，帮助理解迁移诊断系统的工程化实现方式。

4.5 基于 CNN 的时傅里叶变换（STFT）多通道图像迁移

从上述的结果本研究可以看到，在模型训练和预测方面。LightGBM 相比随机森林等模型有更大的优势，本研究只使用一个通道就得到了比较优秀的结果，但是随后的迁移学习上，**由于训练中采用的单通道，且目标域的通道未知，且在采样时窗口的划分过于死板**。导致在迁移之前出现了比较多误判的情况。

因此本研究重新梳理了一遍上述的问题流程，以问题三为导向，通过补充多通道和增加权重的方式使用 CNN 作为主要工具，重新进行了问题三的解答。**基本思想为将基本思想为使用短时傅里叶变换（STFT）将将多通道时序信号从时间域转换为转换为多通道图像**。通过卷积神经网络（CNN）进行训练预测。具体的流程为如下：

首先同之前一样使用加载了包含信号数据路径与相关元数据的索引文件作为数据标定的大纲，与 LightGBM 不同的是根据预先定义的映射规则，筛选出了所需的列，包括信号文件路径、采样频率、转速和故障类型。**并对采样频率和转速进行了数值化处理，对于无法转换为数值的内容，使用 NaN 进行填充**。源域数据被用于模型的训练和模型评估，目标域数据用于迁移学习和预测。

为了避免出现之前由于通道数过少导致的预测失败，本研究在信号文件中可能包含多个通道的数据，选择正确的信号通道是进行后续分析的关键。本研究根据通道的优先级顺序选择信号通道，**优先选择“DE”通道，其次是“FE”通道，最后是“BA”通道**。如果所有通道均不存在或无效，则跳过该信号文件。对选定的信号通道数据进行数值转换，使用 pandas 函数将信号数据转化为浮动类型，并填充可能存在的缺失值。

为了确保所有信号具有相同的采样频率，避免因采样率不同而影响分析结果，本研究将所有信号的**采样频率统一为目标频率（32000 Hz）**。通过计算信号的时长，并根据目标采样频率计算所需的新采样点数，使用和问题一相同的方法对信号进行重采样。并通过角度同步使得所有信号在相同的角度位置上对齐。通过计算每个信号的旋转频率，并将**信号的时间轴转换为角度域**，从而实现信号的同步。使用线性插值方法将信号对齐到每个旋转周期的固定角度位置。

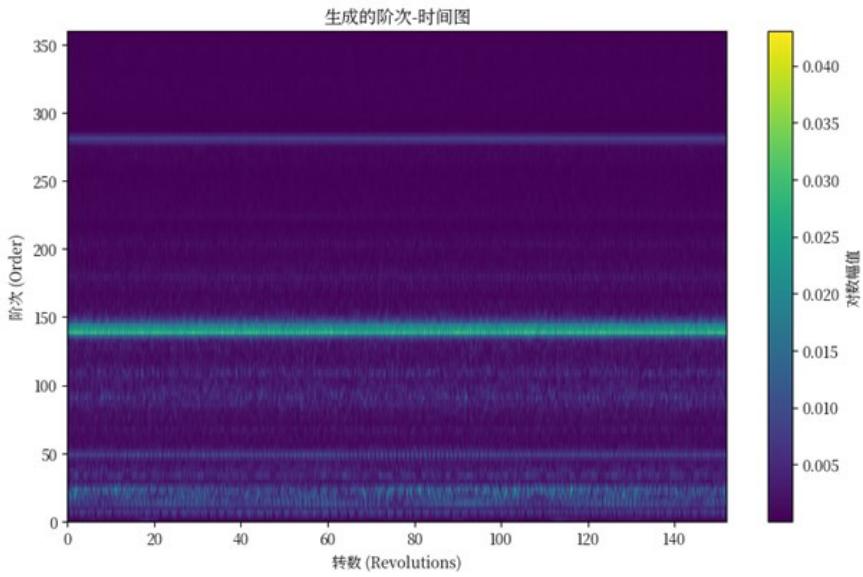


图27 阶次-时间图

使用 CNN 处理的核心思想就是使用短时傅里叶变换 (STFT) 将将多通道时序信号从时间域转换为转换为多通道图像。通过卷积神经网络 (CNN) 进行训练预测短时傅里叶变换 (STFT) 用于将信号从时间域转换为频域图像。将信号分段并计算每段的频谱，生成了二维时间-频率图像。这些图像不仅提供了信号的时变频谱信息，而且通过对数变换压缩了频谱的动态范围，从而使得模型训练更加稳定。下图为 0007 尺寸内圈故障 3 荷载下的信号转换得到的阶次频谱图像。

通过批量处理流程，本研究对所有源域数据进行了处理，并将每个信号的 STFT 图像保存为 .npy 格式。这些图像被用作后续模型训练的输入数据。所有处理过的图像文件路径和相关元数据（如故障标签、转速等）被记录在一个 CSV 格式的元数据文件中，便于后续的模型训练和预测。

	image_path	original_file	label	rpm	original_fs
0	./第3问_CNN/processed_images/N_0.npy	./第3问_CNN/cleaned/48kHz_Normal_data/N_0.csv	N	1796	48000
1	./第3问_CNN/processed_images/N_1_(1772rpm).npy	./第3问_CNN/cleaned/48kHz_Normal_data/N_1_(1772...)	N	1772	48000
2	./第3问_CNN/processed_images/N_2_(1750rpm).npy	./第3问_CNN/cleaned/48kHz_Normal_data/N_2_(1750...)	N	1750	48000
3	./第3问_CNN/processed_images/N_3.npy	./第3问_CNN/cleaned/48kHz_Normal_data/N_3.csv	N	1725	48000
4	./第3问_CNN/processed_images/B007_0.npy	./第3问_CNN/cleaned/12kHz_DE_data/B/0007/B007_0...	B	1798	12000

图28 source_manifest

在源域数据集上，本研究使用卷积神经网络 (CNN) 进行训练。由于类别不平衡问题，本研究采用了带权重的随机采样 (Weighted Random Sampling) 方法来解决 N 类识别模糊的情况，以确保各类样本在训练中的学习均衡。

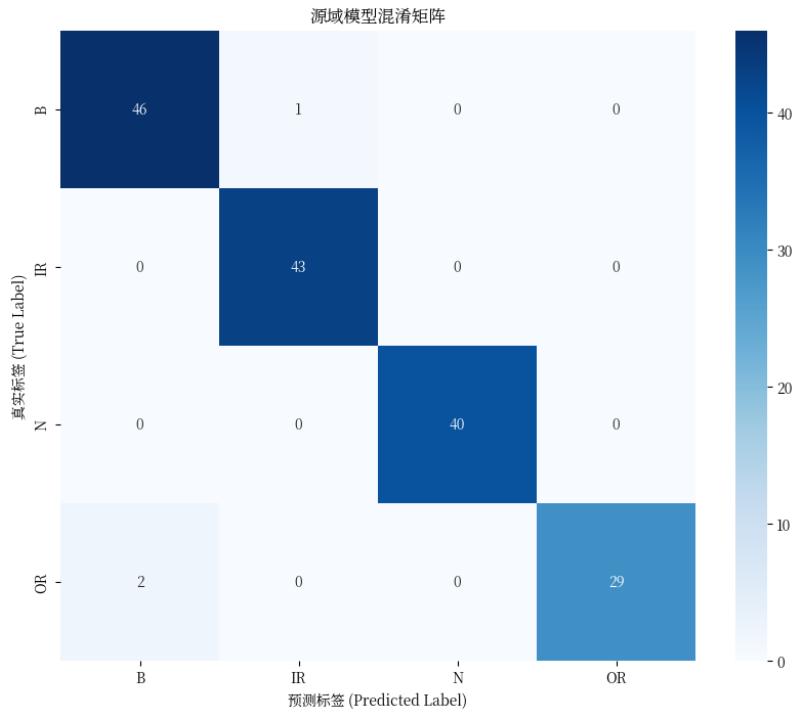


图29 源域模型混淆矩阵

经过 50 次迭代之后可以看到平均损失在 0.5 上下浮动，最终模型在源域上的总体准确率达到了 98.14%。



图30 迭代过程

即使源域的结果再好，源域模型只是一个“预训练模型”，迁移学习的目的就是让它去适应目标域。由于没有特征，本研究使用无监督域适应，最终的目标是强迫模型在学习分类的同时，也学会提取一种“域不变”的特征——即这种特征在源域和目标域数据上看起来都差不多。具体而言就是创建两个独立的数据加载器，然后在训练循环中同时从它们那里取数据。使得能在一个训练批次中同时获得源域和目标域的数据。之后即可加入 CORAL 损失的计算，并用这两个域的数据同时进行训练。这个单元格会定义 CORAL 损失函数，并建立一个可以同时使用源域和目标域数据进行训练的循环。**30 次迭代后就得到了**

一个既懂源域分类，又适应了目标域数据特性的新模型。重新对目标域文件进行一次最终的预测，发现已经不会出现将全部目标域误判成同一个故障的情况了。

	target_file	predicted_fault_type
0	A.csv	N
1	B.csv	IR
2	C.csv	IR
3	D.csv	IR
4	E.csv	OR
5	F.csv	IR
6	G.csv	IR
7	H.csv	IR
8	I.csv	IR
9	J.csv	IR
10	K.csv	IR
11	L.csv	IR
12	M.csv	IR
13	N.csv	IR
14	O.csv	IR
15	P.csv	IR

图31 目标域故障诊断结果

五、问题四：迁移诊断的可解释性

CNN 模型设计简洁，包含两层卷积（ 3×3 滤波器，ReLU 激活，零填充）、最大池化层、三层全连接层（dropout 0.2），使用 Adam 优化器（学习率 0.01）和二元交叉熵损失。相比深层复杂模型，该结构减少“黑箱”特性，便于追踪特征提取过程。卷积层提取图像局部特征（如高频峰值对应内圈故障），池化层聚焦关键区域（如故障诱发的谐振），全连接层整合特征进行分类。简单结构使诊断人员易于理解模型如何从输入中学习故障模式，例如外圈故障的低频冲击在卷积层中表现为显著的空间特征。这种透明性有助于诊断人员信任模型在跨工况（如转速 1720–3000rpm）或跨设备（如 CWRU 到 LPS）诊断中的泛化能力，避免对复杂模型的盲目信任。

六、总结与展望

6.1 总结

在本研究中，我们提出了一种基于迁移学习的智能诊断方法，用于解决高速列车轴承故障诊断中的数据稀缺和噪声干扰问题。该方法结合了 LightGBM 和 CNN 两种强大的机器学习模型，利用试验台架数据（源域）训练模型，并通过迁移学习将其应用到实际列车运行数据（目标域）。本文的研究成果主要体现在以下几个方面：

为了应对源域与目标域数据分布差异，我们设计并实现了一整套数据预处理与特征工程流程。通过滑动窗口法对信号进行分段，并使用时域统计、频域分析、以及基于故障机理的分析方法提取 28 个维度的特征。这一过程中，特征工程确保了数据的结构化，且能有效表达轴承故障的相关信息，为后续的诊断模型训练提供了高质量的输入数据。

在源域数据上，本文选用了 LightGBM 作为核心分类器，并通过超参数优化、交叉验证等技术，显著提高了模型的诊断精度。通过对模型的性能进行可视化分析，我们直观地揭示了不同故障类型的关键判别特征，并通过混淆矩阵展示了模型在各类故障诊断中的表现。最终，LightGBM 模型在源域数据上的诊断准确率接近 100%，证明了该方法的有效性。

针对目标域数据的性能退化问题，本研究引入了卷积神经网络（CNN）模型，通过短时傅里叶变换（STFT）将时序信号转化为多通道图像，利用 CNN 进行迁移学习。该方法能够有效解决源域与目标域数据间的差异，提高了目标域数据的诊断精度。同时，带权重的随机采样方法确保了不同类别样本的均衡学习，进一步提升了模型的诊断能力。

通过迁移学习技术，我们成功将源域知识迁移到目标域，从而解决了因数据稀缺和类别不平衡导致的故障诊断问题。采用域对齐方法有效缓解了源域和目标域数据分布差异，确保了在目标域的诊断模型能够保持较高的准确性。

6.2 展望

尽管本文提出的迁移学习方法取得了良好的效果，但在实际应用中，依然面临一些挑战和改进空间。未来的研究可以从以下几个方面进行深入探索：

在本研究中，尽管我们优先选择了 DE 通道的数据，但实际情况中可能需要更多传感器通道的数据来提供更全面的故障特征。因此，未来可以进一步探索多通道数据的融合，并利用深度学习方法（如卷积神经网络）进行更高效的特征学习，提升多源数据的诊断性能。此外，深度神经网络模型如自编码器、长短期记忆（LSTM）网络等，也可以用于提取更复杂的时序特征，从而进一步提升故障诊断的精度。

尽管迁移学习解决了目标域数据稀缺的问题，但仍然面临小样本学习的挑战。未来可以进一步结合生成对抗网络（GAN）等技术，通过数据增强的方式生成更多的故障样本，提升模型的鲁棒性。同时，采用元学习（meta-learning）等小样本学习方法，进一步提高模型在稀缺样本下的学习能力。

在实际应用中，深度学习模型常常被视为“黑箱”，因此，增强模型的可解释性是一个重要的研究方向。未来可以结合模型可解释性技术，如 SHAP、LIME 等方法，深入分析故障诊断模型的决策过程，为诊断人员提供更直观、可信的决策依据。此外，通过进一步的可视化技术（如 t-SNE 和 PCA 降维）展示特征在不同故障状态下的分布，有助于提升模型在实际工程中的应用价值。

本研究主要聚焦于高速列车轴承的故障诊断，未来可以将该方法扩展到其他类型的机械设备中，实现跨领域、跨设备的故障诊断。跨领域和跨设备的迁移学习能够有效应对不同工况、传感器设置以及设备类型下的数据差异，提高模型的普适性和可扩展性。

为了实现对高速列车等关键设备的实时监控，未来的研究可以进一步探索如何将故障诊断模型应用于实时数据流的分析和预测中。通过结合在线学习和增量学习等技术，能够在实时采集到新的振动信号后迅速更新模型，提供即时的故障预警和维护决策支持。

结语

本研究通过结合 LightGBM 和 CNN，提出了一种有效的基于迁移学习的智能故障诊断方法，并成功解决了高速列车轴承故障诊断中的数据稀缺和噪声干扰问题。未来，随着更多先进技术的应用和研究的深入，智能故障诊断技术有望在更多领域得到广泛应用，并为提高设备可靠性、减少维护成本提供重要支持。

七、附录

第一问

```
# 本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的
```

```
import re
import h5py
import numpy as np
import pandas as pd
import scipy.io as sio
from pathlib import Path
from tqdm import tqdm

SRC_ROOT = Path("0 数据集/源域数据集")
CSV_DIR = Path("第 1 问/0csv 无处理")
DETAIL_CSV = Path("第 1 问/0detail.csv")

CSV_DIR.mkdir(parents=True, exist_ok=True)
DETAIL_CSV.parent.mkdir(parents=True, exist_ok=True)

def extract_signals(path: Path):
    """
    从 .mat 文件提取所有有效的一维信号。
    对 v7.3 格式 (HDF5) 单独处理。
    """
    try:
        data = sio.loadmat(path, squeeze_me=True, struct_as_record=False)
        signals = {
            k: np.array(v).squeeze()
            for k, v in data.items()
            if not k.startswith("__")
            and isinstance(v, np.ndarray)
            and v.ndim == 1
            and v.size > 10
        }
        return signals, data
    except NotImplementedError:
        with h5py.File(path, "r") as f:
            signals = {}
            for k in f.keys():
                v = np.array(f[k]).squeeze()
                if v.ndim == 1 and v.size > 10:
                    signals[k] = v
```

```

        return signals, f
    return {}, None

def rpm_from_data(data):
    """
    尝试从数据内容或键名中解析转速 rpm。
    """

    if isinstance(data, dict):
        for k, v in data.items():
            if "RPM" in k or "rpm" in k or k.lower() == "n":
                arr = np.array(v).squeeze()
                if np.isscalar(arr):
                    return int(arr)
                if arr.size == 1:
                    return int(arr.item())
    if isinstance(data, h5py.File):
        for k in data.keys():
            if "RPM" in k or "rpm" in k or k.lower() == "n":
                arr = np.array(data[k]).squeeze()
                if np.isscalar(arr):
                    return int(arr)
                if arr.size == 1:
                    return int(arr.item())
    return None

def parse_meta(path: Path, data=None):
    """
    从文件名/路径推断采样率、工况标签、负载、转速等元信息。
    """

    name = path.stem
    split = next((p for p in path.parts if "Hz" in p), None)

    sr = 12000 if "12kHz" in str(path) else (48000 if "48kHz" in str(path) else None)

    label = None
    if "IR" in str(path):
        label = "IR"
    elif "OR" in str(path):
        label = "OR"
    elif "B" in str(path):
        label = "B"
    elif "Normal" in str(path) or name.startswith("N"):

```

```

label = "N"

m_size = re.search(r"(?:B|IR|OR)(\d{3})", name, re.I)
size_code = m_size.group(1) if m_size else None

m_load = re.search(r"_(\d)(?:[_.])]", name)
load = int(m_load.group(1)) if m_load else None

m_rpm = re.search(r"(\d+)\s*rpm", name, re.I)
rpm = int(m_rpm.group(1)) if m_rpm else None
if rpm is None and data is not None:
    rpm = rpm_from_data(data)

m_clock = re.search(r"@(\d{1,2})", name)
clock_pos = int(m_clock.group(1)) if m_clock else None

return {
    "path": str(path),
    "file": path.name,
    "split": split,
    "sr_hz": sr,
    "label": label,
    "size_code": size_code,
    "load": load,
    "rpm": rpm,
    "clock_pos": clock_pos,
}

# 主流程: 批量转换 MAT → CSV, 并记录索引表
rows = []
for f in tqdm(list(SRC_ROOT.rglob("*.mat"))):
    signals, data = extract_signals(f)
    if not signals:
        continue

    meta = parse_meta(f, data)

    out_path = CSV_DIR / f.relative_to(SRC_ROOT)
    out_path = out_path.with_suffix(".csv")
    out_path.parent.mkdir(parents=True, exist_ok=True)

    df = pd.DataFrame(signals)
    df.to_csv(out_path, index=False, encoding="utf-8-sig")

```

```

meta["csv_path"] = str(out_path)
rows.append(meta)

df_index = pd.DataFrame(rows)
df_index.to_csv(DETAIL_CSV, index=False, encoding="utf-8-sig")
print(f"✅ 转换完成, 共 {len(df_index)} 个文件")

# 本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的

import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path

DETAIL_CSV = Path("data/0detail.csv")
CSV_DIR      = Path("data/0csv 无处理")
out_dir      = Path("data/0analyse")
out_dir.mkdir(parents=True, exist_ok=True)

df_index = pd.read_csv(DETAIL_CSV)

# 图 1: 样本数量分布
plt.figure(figsize=(6, 4))
df_index["label"].value_counts().plot(kind="bar", color="skyblue")
plt.title("Distribution of Sample Counts")
plt.xlabel("Category"); plt.ylabel("Count")
plt.tight_layout()
plt.savefig(out_dir / "图 1_样本数量分布.png", dpi=150)
plt.close()

# 图 2: 样本占比
plt.figure(figsize=(6, 6))
df_index["label"].value_counts().plot(
    kind="pie", autopct=".1f%%",
    colors=["#66c2a5", "#fc8d62", "#8da0cb", "#e78ac3"]
)
plt.ylabel("")
plt.title("Sample Percentage")
plt.tight_layout()
plt.savefig(out_dir / "图 2_样本占比.png", dpi=150)
plt.close()

# 图 3: IR 波形示例

```

```

row = df_index[df_index["label"] == "IR"].iloc[0]
df = pd.read_csv(row["csv_path"])
plt.figure(figsize=(10, 3))
plt.plot(df.iloc[:2000, 0].values, lw=0.7)
plt.title(f"IR Waveform Example ({row['file']}")")
plt.xlabel("Sample"); plt.ylabel("Amplitude")
plt.tight_layout()
plt.savefig(out_dir / "图 3_IR 波形.png", dpi=150)
plt.close()

# 图 4: OR 波形示例
row = df_index[df_index["label"] == "OR"].iloc[0]
df = pd.read_csv(row["csv_path"])
plt.figure(figsize=(10, 3))
plt.plot(df.iloc[:2000, 0].values, lw=0.7, color="orange")
plt.title(f"OR Waveform Example ({row['file']}")")
plt.xlabel("Sample"); plt.ylabel("Amplitude")
plt.tight_layout()
plt.savefig(out_dir / "图 4_OR 波形.png", dpi=150)
plt.close()

# 图 5: B 波形示例
row = df_index[df_index["label"] == "B"].iloc[0]
df = pd.read_csv(row["csv_path"])
plt.figure(figsize=(10, 3))
plt.plot(df.iloc[:2000, 0].values, lw=0.7, color="green")
plt.title(f"B Waveform Example ({row['file']}")")
plt.xlabel("Sample"); plt.ylabel("Amplitude")
plt.tight_layout()
plt.savefig(out_dir / "图 5_B 波形.png", dpi=150)
plt.close()

# 图 6: IR 不同载荷下的对比
plt.figure(figsize=(10, 3))
sub = df_index[df_index["label"] == "IR"].dropna(subset=["load"])
for load in sorted(sub["load"].unique()):
    row = sub[sub["load"] == load].iloc[0]
    sig = pd.read_csv(row["csv_path"]).iloc[:2000, 0].values
    plt.plot(sig, lw=0.7, label=f"Load {load}")
plt.title("IR - Different Loads")
plt.xlabel("Sample"); plt.ylabel("Amplitude")
plt.legend()
plt.tight_layout()

```

```

plt.savefig(out_dir / "图 6_IR 不同载荷.png", dpi=150)
plt.close()

# 图 7: OR 不同载荷下的对比
plt.figure(figsize=(10, 3))
sub = df_index[df_index["label"] == "OR"].dropna(subset=["load"])
for load in sorted(sub["load"].unique()):
    row = sub[sub["load"] == load].iloc[0]
    sig = pd.read_csv(row["csv_path"]).iloc[:2000, 0].values
    plt.plot(sig, lw=0.7, label=f"Load {load}")
plt.title("OR - Different Loads")
plt.xlabel("Sample"); plt.ylabel("Amplitude")
plt.legend()
plt.tight_layout()
plt.savefig(out_dir / "图 7_OR 不同载荷.png", dpi=150)
plt.close()

# 图 8: B 不同载荷下的对比
plt.figure(figsize=(10, 3))
sub = df_index[df_index["label"] == "B"].dropna(subset=["load"])
for load in sorted(sub["load"].unique()):
    row = sub[sub["load"] == load].iloc[0]
    sig = pd.read_csv(row["csv_path"]).iloc[:2000, 0].values
    plt.plot(sig, lw=0.7, label=f"Load {load}")
plt.title("B - Different Loads")
plt.xlabel("Sample"); plt.ylabel("Amplitude")
plt.legend()
plt.tight_layout()
plt.savefig(out_dir / "图 8_B 不同载荷.png", dpi=150)
plt.close()

# 图 9: 正常 vs 故障对比
plt.figure(figsize=(10, 3))
for lab, color in zip(["N", "IR", "OR", "B"], ["black", "red", "orange", "green"]):
    sub = df_index[df_index["label"] == lab]
    if not sub.empty:
        row = sub.iloc[0]
        sig = pd.read_csv(row["csv_path"]).iloc[:2000, 0].values
        plt.plot(sig, lw=0.7, label=lab, color=color)
plt.title("Normal vs Fault")
plt.xlabel("Sample"); plt.ylabel("Amplitude")
plt.legend()
plt.tight_layout()

```

```

plt.savefig(out_dir / "图 9_正常 vs 故障.png", dpi=150)
plt.close()

# 图 10: 相同载荷下不同故障对比 (以 load=0 为例)
plt.figure(figsize=(10, 3))
sub = df_index.dropna(subset=["load"])
same_load = sub[sub["load"] == 0]
for lab, color in zip(["IR", "OR", "B"], ["red", "orange", "green"]):
    sub_lab = same_load[same_load["label"] == lab]
    if not sub_lab.empty:
        row = sub_lab.iloc[0]
        sig = pd.read_csv(row["csv_path"]).iloc[:2000, 0].values
        plt.plot(sig, lw=0.7, label=lab, color=color)
plt.title("Different Faults Under Same Load (Load=0)")
plt.xlabel("Sample"); plt.ylabel("Amplitude")
plt.legend()
plt.tight_layout()
plt.savefig(out_dir / "图 10_同载荷不同故障.png", dpi=150)
plt.close()

print("✅ 所有 10 张图已生成:", out_dir)

# 本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的

# 1 重采样_CNN_fix.py
import pandas as pd
import numpy as np
from pathlib import Path
from scipy.signal import resample_poly
from tqdm import tqdm

# 输入/输出路径配置
SRC_DIR      = Path("第 1 问/0csv 无处理")
DST_DIR      = Path("第 1 问/1.csv_32k")
NPY_DIR      = Path("第 1 问/1.npy_32k")
DETAIL_CSV   = Path("第 1 问/0detail.csv")
OUT_DETAIL   = Path("第 1 问/1.1detail_32k.csv")

DST_DIR.mkdir(parents=True, exist_ok=True)
NPY_DIR.mkdir(parents=True, exist_ok=True)

TARGET_SR = 32000
EXPECTED_CHANNELS = ["DE", "FE", "BA"]    # 需要保证的通道列表

```

```

df_index = pd.read_csv(DETAIL_CSV)
rows = []

for _, row in tqdm(df_index.iterrows(), total=len(df_index)):
    csv_path = Path(row["csv_path"])
    if not csv_path.exists():
        continue

    df = pd.read_csv(csv_path)
    sr = row["sr_hz"]
    if pd.isna(sr):
        continue
    sr = int(sr)

    sigs_resampled = []
    available_channels = []

    # 针对每个目标通道，重采样或补零
    for ch in EXPECTED_CHANNELS:
        cols = [c for c in df.columns if ch in c.upper()]
        if cols:
            sig = df[cols[0]].values
            target_len = int(round(len(sig) * TARGET_SR / sr))

            sig_resampled = resample_poly(sig, TARGET_SR, sr)

            # 保证长度一致：不足补零，多余截断
            if len(sig_resampled) < target_len:
                sig_resampled = np.pad(sig_resampled, (0, target_len - len(sig_resampled)))
            elif len(sig_resampled) > target_len:
                sig_resampled = sig_resampled[:target_len]

            sigs_resampled.append(sig_resampled)
            available_channels.append(ch)

        else:
            # 如果通道缺失，补零信号
            if "target_len" not in locals():
                target_len = int(round(len(df) * TARGET_SR / sr))
            sigs_resampled.append(np.zeros(target_len))

    sigs_resampled = np.stack(sigs_resampled, axis=0) # [C, T]

```

```

# 保存 CSV (如需兼容旧流程, 可取消注释)
rel_path = csv_path.relative_to(SRC_DIR)
out_path = DST_DIR / rel_path
out_path.parent.mkdir(parents=True, exist_ok=True)
# pd.DataFrame(sigs_resampled.T, columns=EXPECTED_CHANNELS).to_csv(out_path, index=False)

# 保存 NPY (CNN 可直接加载)
npy_path = NPY_DIR / rel_path.with_suffix(".npy")
npy_path.parent.mkdir(parents=True, exist_ok=True)
np.save(npy_path, sigs_resampled.astype(np.float32))

# 更新索引信息
new_meta = row.to_dict()
new_meta["sr_hz"] = TARGET_SR
new_meta["csv_path"] = str(out_path)
new_meta["npy_path"] = str(npy_path)
new_meta["channels"] = ",".join(available_channels)
rows.append(new_meta)

df_out = pd.DataFrame(rows)
df_out.to_csv(OUT_DETAIL, index=False, encoding="utf-8-sig")

print(f'✓ 重采样完成, 共 {len(df_out)} 个文件')
print("新索引:", OUT_DETAIL)
print("npy 存放目录:", NPY_DIR)

# 本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的

import pandas as pd
from pathlib import Path
from tqdm import tqdm

# 输入/输出路径配置
SRC_DIR      = Path("第 1 问/1.csv_32k")           # 已重采样的多列 CSV
DST_DIR      = Path("第 1 问/1.csv_32k_DE")        # 输出的单列 CSV
DETAIL_CSV   = Path("第 1 问/1.detail_32k.csv")    # 原始索引
OUT_DETAIL   = Path("第 1 问/1.detail_32k_DE.csv")  # 新索引

DST_DIR.mkdir(parents=True, exist_ok=True)

df_index = pd.read_csv(DETAIL_CSV)
rows = []

```

```

for _, row in tqdm(df_index.iterrows(), total=len(df_index)):
    csv_path = Path(row["csv_path"])
    if not csv_path.exists():
        continue

    df = pd.read_csv(csv_path)

    # 提取 DE 通道列
    de_cols = [c for c in df.columns if "DE" in c.upper()]
    if not de_cols:
        continue

    # 默认保留所有检测到的 DE 列
    df_de = df[de_cols]
    # 如果只需第一个 DE 列, 可改成: df_de = df[[de_cols[0]]]

    # 保存新 CSV
    rel_path = csv_path.relative_to(SRC_DIR)
    out_path = DST_DIR / rel_path
    out_path.parent.mkdir(parents=True, exist_ok=True)
    df_de.to_csv(out_path, index=False, encoding="utf-8-sig")

    # 更新索引信息
    new_meta = row.to_dict()
    new_meta["csv_path"] = str(out_path)
    new_meta["channels"] = ",".join(de_cols)
    rows.append(new_meta)

# 保存新的索引表
df_out = pd.DataFrame(rows)
df_out.to_csv(OUT_DETAIL, index=False, encoding="utf-8-sig")

print(f"✓ 提取完成, 共 {len(df_out)} 个文件")
print("新索引:", OUT_DETAIL)

# 本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的

import pandas as pd
import numpy as np
from pathlib import Path
from tqdm import tqdm
from scipy.signal import butter, filtfilt
import pywt

```

```

DETAILED_CSV = Path("第 1 回/1.2detail_32k_DE.csv")
SEGMENTS_PARQUET = Path("第 1 回/1.3segments.parquet")
SEG_WAVE_DIR = Path("第 1 回/1.3segments_wave")

fs = 32000
L, hop = 4096, 2048
lowcut, highcut = 100, min(5000, int(0.45 * fs))

SEG_WAVE_DIR.mkdir(parents=True, exist_ok=True)

def bandpass_filter(x, fs, lowcut, highcut, order=4):
    nyq = 0.5 * fs
    b, a = butter(order, [lowcut/nyq, highcut/nyq], btype="band")
    return filtfilt(b, a, x)

def wavelet_denoise(x, wavelet="db4", level=4):
    coeffs = pywt.wavedec(x, wavelet, level=level)
    sigma = np.median(np.abs(coeffs[-1])) / 0.6745
    uthresh = sigma * np.sqrt(2*np.log(len(x)))
    coeffs = [pywt.threshold(c, value=uthresh, mode="soft") for c in coeffs]
    return pywt.waverec(coeffs, wavelet)

def normalize(x):
    if np.max(x) == np.min(x):
        return np.zeros_like(x)
    return 2 * (x - np.min(x)) / (np.max(x) - np.min(x)) - 1

df_index = pd.read_csv(DETAILED_CSV)
rows = []

for _, row in tqdm(df_index.iterrows(), total=len(df_index)):
    csv_path = Path(row["csv_path"])
    if not csv_path.exists():
        continue

    df = pd.read_csv(csv_path)
    if df.empty:
        continue

    sig = df.iloc[:, 0].values.astype(float)
    num_segs = (len(sig) - L) // hop + 1

```

```

for seg_idx in range(num_segs):
    start, end = seg_idx * hop, seg_idx * hop + L
    x = sig[start:end]
    if len(x) < L:
        continue

    x = bandpass_filter(x, fs, lowcut, highcut)
    x = wavelet_denoise(x)
    x = normalize(x)

    rel = csv_path.relative_to(DETAIL_CSV.parent.parent)
    safe_rel = "_".join(rel.parts).replace(".csv", "")
    seg_file = SEG_WAVE_DIR / f'{safe_rel}_seg{seg_idx}.npy'
    np.save(seg_file, x.astype(np.float32))

    rows.append({
        "src_csv": str(csv_path),
        "seg_file": str(seg_file),
        "seg_idx": seg_idx,
        "start": start,
        "end": end,
        "label": row.get("label"),
        "load": row.get("load"),
        "rpm": 600,
        "size_code": row.get("size_code"),
        "clock_pos": row.get("clock_pos"),
    })

df_segments = pd.DataFrame(rows)
df_segments.to_parquet(SEGMENTS_PARQUET, index=False)

print(f"✅ 已生成 {len(df_segments)} 个段")
print(f"索引表: {SEGMENTS_PARQUET}")
print(f"波形段目录: {SEG_WAVE_DIR}")

# 本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的

"""

从分段波形提取特征:
时域、频域、包络谱特征、小波包能量与熵
"""

import numpy as np

```

```

import pandas as pd
from pathlib import Path
from scipy.signal import welch, hilbert
from scipy.stats import kurtosis, skew, entropy
import pywt
from tqdm import tqdm

SEGMENTS_PARQUET = Path("第 1 问/1.3segments.parquet")
SEG_WAVE_DIR      = Path("第 1 问/1.3segments_wave")
OUT_DIR           = Path("第 1 问/2features"); OUT_DIR.mkdir(parents=True, exist_ok=True)
OUT_PARQUET        = OUT_DIR / "features.parquet"

fs = 32000
BEARING = dict(n=9, d=0.3126, D=1.537)    # SKF6205 参数

def safe_entropy(x, bins=64):
    h, _ = np.histogram(x, bins=bins, density=True)
    return float(entropy(h + 1e-12))

def time_features(x):
    x = np.asarray(x, dtype=float).ravel()    # ★ flatten
    x_abs = np.abs(x)
    rms = np.sqrt(np.mean(x**2) + 1e-20)
    mean_abs = np.mean(x_abs) + 1e-20
    sqrt_abs_mean = np.mean(np.sqrt(x_abs)) + 1e-20
    return dict(
        mean=float(np.mean(x)),
        std=float(np.std(x, ddof=1)),
        rms=float(rms),
        skew=float(skew(x, bias=False)),
        kurtosis=float(kurtosis(x, fisher=True, bias=False)),
        p2p=float(np.max(x) - np.min(x)),
        crest_factor=float(np.max(x_abs) / rms),
        shape_factor=float(rms / mean_abs),
        impulse_factor=float(np.max(x_abs) / mean_abs),
        clearance_factor=float(np.max(x_abs) / (sqrt_abs_mean**2)),
        sig_entropy=float(safe_entropy(x, bins=64)),
    )

def spectral_features(x, sr):
    x = np.asarray(x, dtype=float).ravel()    # ★ flatten
    # ★ 给 nperseg 一个下限，避免 1 段的崩溃
    nper = min(len(x), 4096)

```

```

nper = max(nper, 256)
# 可选：提高分辨率更稳
nfft = max(16384, 1<<int(np.ceil(np.log2(nper))))
f, Pxx = welch(x, fs=sr, nperseg=nper, noverlap=nper//2, nfft=nfft, detrend="constant")
Pxx = Pxx + 1e-20
Psum = np.trapz(Pxx, f)
p_norm = Pxx / Psum
peak_idx = int(np.argmax(Pxx))
centroid = float(np.sum(f * p_norm))
spread = float(np.sqrt(np.sum(((f - centroid)**2) * p_norm)))
spec_entropy = float(-np.sum(p_norm * np.log(p_norm)))
cumsum = np.cumsum(p_norm)
idx95 = np.searchsorted(cumsum, 0.95)
idx95 = min(idx95, len(f) - 1)
rolloff95 = float(f[idx95])
rms_freq = float(np.sqrt(np.sum((f**2) * Pxx) / Psum))
return dict(
    freq_peak=float(f[peak_idx]),
    spec_centroid=centroid,
    freq_std=spread,
    spec_entropy=spec_entropy,
    spec_rolloff95=rolloff95, # ★ 注意拼写：spec_rolloff95 (两个 1)
    rms_freq=rms_freq,
)

```



```

def char_freqs(rpm, n, d, D):
    fr = rpm / 60.0
    bpfo = fr * n/2.0 * (1 - d/D)
    bpf1 = fr * n/2.0 * (1 + d/D)
    bsf = fr * (D/d) * (1 - (d/D)**2)
    ftf = 0.5 * fr * (1 - d/D)
    return fr, bpfo, bpf1, bsf, ftf

```



```

def envelope_features(x, sr, rpm, tol=0.05, bearing=BEARING):
    if rpm is None or not np.isfinite(rpm) or rpm <= 1:
        return {}
    x = np.asarray(x, dtype=float).ravel() # ★ flatten
    env = np.abs(hilbert(x))
    nper = min(len(env), 4096); nper = max(nper, 256)
    f, P = welch(env, fs=sr, nperseg=nper, noverlap=nper//2, detrend="constant")
    P = P + 1e-20
    out = {}

```

```

_, bpfo, bpfi, bsf, ftf = char_freqs(float(rpm), bearing["n"], bearing["d"], bearing["D"])
for name, f0 in {"bpfo": bpfo, "bpfi": bpfi, "bsf": bsf, "ftf": ftf}.items():
    lo, hi = (1 - tol) * f0, (1 + tol) * f0
    m = (f >= lo) & (f <= hi)
    E = float(np.trapz(P[m], f[m])) if np.any(m) else 0.0
    out[f'{name}_E'] = E
return out

def wavelet_packet_features(x, wavelet="db4", level=3):
    x = np.asarray(x, dtype=float).ravel()      # ★ flatten
    wp = pywt.WaveletPacket(x, wavelet, maxlevel=level)
    nodes = [node.path for node in wp.get_level(level, "freq")]
    energies = [np.sum(np.square(wp[node].data)) for node in nodes]
    total = np.sum(energies) + 1e-20
    probs = np.array(energies) / total
    H = -np.sum(probs * np.log(probs + 1e-12))
    feats = {f'wp_E{i}': float(p) for i, p in enumerate(probs)}
    feats["wp_entropy"] = float(H)
    return feats

def main():
    df = pd.read_parquet(SEGMENTS_PARQUET)
    rows = []
    for _, r in tqdm(df.iterrows(), total=len(df)):
        seg_file = Path(r["seg_file"])
        if not seg_file.exists():
            continue
        x = np.load(seg_file).astype(float).ravel()
        feat = dict(
            seg_file=str(seg_file),
            label=r.get("label"),
            load=r.get("load"),
            rpm=r.get("rpm"),
            size_code=r.get("size_code"),
            clock_pos=r.get("clock_pos"),
        )
        feat.update(time_features(x))
        feat.update(spectral_features(x, fs))
        feat.update(envelope_features(x, fs, r.get("rpm")))
        feat.update(wavelet_packet_features(x))
        rows.append(feat)
    df_feat = pd.DataFrame(rows)
    df_feat.to_parquet(OUT_PARQUET, index=False)

```

```

print(f"✅ 特征提取完成: {len(df_feat)} 段, 已写入 {OUT_PARQUET}")

if __name__ == "__main__":
    main()

# 本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的

"""
特征降维与可视化: 类别分布、特征相关性、PCA、t-SNE
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

FEAT_PARQUET = Path("第 1 问/2features/features.parquet")
OUT_DIR = Path("第 1 问/2 特征分析"); OUT_DIR.mkdir(parents=True, exist_ok=True)

df = pd.read_parquet(FEAT_PARQUET)

non_feat_cols = ["seg_file", "label", "load", "rpm", "size_code", "clock_pos"]
feat_cols = [c for c in df.columns if c not in non_feat_cols]
X = df[feat_cols].values
y = df["label"].astype(str).values

scaler = StandardScaler()
X_std = scaler.fit_transform(X)

# 类别分布
plt.figure(figsize=(6, 4))
sns.countplot(x="label", hue="label", data=df,
               order=df["label"].value_counts().index,
               palette="Set2", legend=False)
plt.title("Class Distribution")
plt.savefig(OUT_DIR/"class_distribution.png", dpi=150)
plt.close()

# 特征相关性
plt.figure(figsize=(12, 10))

```

```

corr = pd.DataFrame(X, columns=feat_cols).corr()
sns.heatmap(corr, cmap="coolwarm", center=0)
plt.title("Feature Correlation Heatmap")
plt.tight_layout()
plt.savefig(OUT_DIR/"feature_correlation.png", dpi=150)
plt.close()

# PCA 方差解释率
pca = PCA()
X_pca = pca.fit_transform(X_std)
expl_var = pca.explained_variance_

plt.figure(figsize=(6, 4))
plt.plot(np.cumsum(expl_var) * 100, marker="o")
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance (%)")
plt.title("PCA Explained Variance")
plt.grid(True)
plt.savefig(OUT_DIR/"pca_variance.png", dpi=150)
plt.close()

# PCA 前两维
plt.figure(figsize=(6, 5))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=y, alpha=0.6, palette="Set1", s=20)
plt.xlabel("PC1"); plt.ylabel("PC2")
plt.title("PCA (first 2 components)")
plt.legend()
plt.savefig(OUT_DIR/"pca_scatter.png", dpi=150)
plt.close()

# t-SNE
tsne = TSNE(n_components=2, random_state=42, init="pca",
             learning_rate="auto", perplexity=30)
X_tsne = tsne.fit_transform(X_std)

plt.figure(figsize=(6, 5))
sns.scatterplot(x=X_tsne[:, 0], y=X_tsne[:, 1], hue=y, alpha=0.6, palette="Set1", s=20)
plt.xlabel("t-SNE 1"); plt.ylabel("t-SNE 2")
plt.title("t-SNE (2D)")
plt.legend()
plt.savefig(OUT_DIR/"tsne_scatter.png", dpi=150)
plt.close()

```

```

# PCA vs t-SNE

fig, axs = plt.subplots(1, 2, figsize=(12, 5))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=y, alpha=0.6,
                 palette="Set1", s=15, ax=axs[0])
axs[0].set_title("PCA 2D")
sns.scatterplot(x=X_tsne[:, 0], y=X_tsne[:, 1], hue=y, alpha=0.6,
                 palette="Set1", s=15, ax=axs[1])
axs[1].set_title("t-SNE 2D")
plt.tight_layout()
plt.savefig(OUT_DIR/"pca_tsne_compare.png", dpi=150)
plt.close()

print(f"✅ 可视化完成，图片已保存到 {OUT_DIR}")

```

第二问

```

# %% [markdown]
## 本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的
#
# %%

import warnings, re, os
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
from pathlib import Path
from collections import Counter

from sklearn.model_selection import StratifiedGroupKFold
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

try:
    from lightgbm import LGBMClassifier
    HAS_LGBM = True
except Exception:
    HAS_LGBM = False

```

```

try:
    from imblearn.over_sampling import RandomOverSampler
    HAS_IMBLEARN = True
except Exception:
    HAS_IMBLEARN = False

import matplotlib.pyplot as plt
print("HAS_LGBM =", HAS_LGBM, "| HAS_IMBLEARN =", HAS_IMBLEARN)

# %%
FEAT_PARQUET = Path("../第1问/2features/features.parquet") # 请按你的路径调整
OUT_DIR = Path("../第2问/3models"); OUT_DIR.mkdir(parents=True, exist_ok=True)
print("工作目录:", os.getcwd())

# %%
def add_rpm_normalized_features(df, feat_cols):
    df2 = df.copy()
    freq_like = ["freq_peak", "spec_centroid", "freq_std", "spec_rolloff95", "rms_freq"]
    env_cols = ["bpfo_E", "bpfi_E", "bsf_E", "ftf_E"]

    fr = (df2["rpm"].astype(float) / 60.0).replace([np.inf, -np.inf], np.nan)
    for c in freq_like:
        if c in df2.columns:
            df2[c + "_order"] = df2[c] / fr

    has_env = [c for c in env_cols if c in df2.columns]
    if has_env:
        total = df2[has_env].sum(axis=1) + 1e-12
        for c in has_env:
            df2[c + "_ratio"] = df2[c] / total

    new_cols = [c for c in df2.columns if c not in df.columns]
    return df2, feat_cols + new_cols

def group_from_segfile(p):
    name = Path(p).stem
    return re.sub(r"_seg\d+$", "", name)

def plot_confusion_mean(cm_mean, labels, title, save_path):
    fig = plt.figure(figsize=(6,5))
    ax = fig.add_subplot(111)
    im = ax.imshow(cm_mean, aspect="auto")
    ax.set_xticks(range(len(labels))); ax.set_xticklabels(labels, rotation=45, ha="right")

```

```

ax.set_yticks(range(len(labels))); ax.set_yticklabels(labels)
for i in range(len(labels)):
    for j in range(len(labels)):
        ax.text(j, i, f'{cm_mean[i, j]:.0f}', ha="center", va="center")
ax.set_xlabel("Predicted"); ax.set_ylabel("True")
ax.set_title(title)
fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
plt.tight_layout()
plt.savefig(save_path, dpi=150)
plt.show()

# %%
if not FEAT_PARQUET.exists():
    print("未找到", FEAT_PARQUET)
    print("当前目录可见: ", os.listdir())
    raise FileNotFoundError("请修改 FEAT_PARQUET 为你的 features.parquet 实际路径")

df = pd.read_parquet(FEAT_PARQUET)
meta_cols = ["seg_file", "label", "load", "rpm", "size_code", "clock_pos"]
for c in meta_cols:
    if c not in df.columns:
        raise ValueError(f"缺少必须列: {c}")

feat_cols = [c for c in df.columns if c not in meta_cols]
df.shape, len(feat_cols)

# %%
df, feat_cols = add_rpm_normalized_features(df, feat_cols)
print("特征数 (含派生) : ", len(feat_cols))

# %%

labels = df["label"].astype(str)
cls_cnt = Counter(labels)
avg_n = np.mean(list(cls_cnt.values()))
class_weight = {c: avg_n / n for c, n in cls_cnt.items()}
sample_weight_all = labels.map(class_weight).astype(float).values

groups = df["seg_file"].apply(group_from_segfile)
X = df[feat_cols].values
y = labels.values

```

```

cv = StratifiedGroupKFold(n_splits=5, shuffle=True, random_state=42)
print("类别统计: ", cls_cnt)

# %%

def run_one_model(name, model, oversample=False):
    accs, f1s, cms = [], [], []
    label_order = sorted(cls_cnt.keys())

    if oversample:
        try:
            from imblearn.over_sampling import RandomOverSampler
            oversampler = RandomOverSampler(sampling_strategy="not minority", random_state=42)
        except Exception:
            print("[WARN] imblearn 未安装, 过采样将被忽略。")
            oversampler = None
    else:
        oversampler = None

    for fold, (tr, te) in enumerate(cv.split(X, y, groups=groups), 1):
        X_tr, y_tr = X[tr], y[tr]
        X_te, y_te = X[te], y[te]
        sw_tr = sample_weight_all[tr]

        if oversampler is not None:
            X_tr, y_tr = oversampler.fit_resample(X_tr, y_tr)
            sw_tr = np.ones(len(y_tr), dtype=float)

        fit_kwargs = {}
        if 'LGBMClassifier' in str(type(model)):
            model.set_params(class_weight=None)
            fit_kwargs["sample_weight"] = sw_tr

        model.fit(X_tr, y_tr, **fit_kwargs)
        y_pred = model.predict(X_te)
        accs.append(accuracy_score(y_te, y_pred))
        f1s.append(f1_score(y_te, y_pred, average="macro"))
        cms.append(confusion_matrix(y_te, y_pred, labels=label_order))
        print(f"Fold {fold}: acc={accs[-1]:.4f}, macroF1={f1s[-1]:.4f}")

    acc_mean, f1_mean = float(np.mean(accs)), float(np.mean(f1s))
    cm_mean = np.mean(np.stack(cms, axis=0), axis=0)
    plot_confusion_mean(cm_mean, label_order, f'{name} - Mean Confusion Matrix (5-fold)", OUT_DIR /
```

```

f" {name}_cm.png")
    print(f"[ {name}] CV-mean acc={acc_mean:.4f} | macroF1={f1_mean:.4f}")
    return name, acc_mean, f1_mean

# %%

models = []
if HAS_LGBM:
    models.append(("LightGBM", LGBMClassifier(
        objective="multiclass",
        num_class=len(set(y)),
        n_estimators=700,
        learning_rate=0.05,
        max_depth=-1,
        subsample=0.9,
        colsample_bytree=0.9,
        reg_lambda=1.0,
        random_state=42
    )))

models.append(("RandomForest", RandomForestClassifier(
    n_estimators=600,
    max_depth=None,
    class_weight={k:v for k,v in class_weight.items()},
    n_jobs=-1,
    random_state=42
)))

```

pipe = Pipeline(steps=[("scaler", StandardScaler()),
 ("pca", PCA(n_components=20, random_state=42)),
 ("clf", LogisticRegression(max_iter=3000, n_jobs=-1))])

models.append(("LogReg+PCA20", pipe))

```

leaderboard = []
for name, model in models:
    leaderboard.append(run_one_model(name, model, oversample=False))

df_scores = pd.DataFrame(leaderboard,
columns=["model","cv_acc","cv_macroF1"]).sort_values("cv_macroF1", ascending=False)
df_scores.to_csv(OUT_DIR/"cv_scores.csv", index=False)
df_scores

# %%

```

```

import joblib, json

best_model = LGBMClassifier(
    objective="multiclass",
    num_class=len(set(y)),
    n_estimators=700,
    learning_rate=0.05,
    max_depth=-1,
    subsample=0.9,
    colsample_bytree=0.9,
    reg_lambda=1.0,
    random_state=42
)
best_model.fit(X, y, sample_weight=sample_weight_all)

ARTI_DIR = Path("../data/3artifacts"); ARTI_DIR.mkdir(exist_ok=True)

json.dump(feat_cols, open(ARTI_DIR/"feat_cols.json","w"))

scaler = StandardScaler().fit(X)
joblib.dump(scaler, ARTI_DIR/"scaler.pkl")

joblib.dump(best_model, ARTI_DIR/"model.pkl")

classes = list(best_model.classes_)
json.dump(classes, open(ARTI_DIR/"classes.json","w"))

print("✅ artifacts 已导出到", ARTI_DIR)

# %%

import warnings, re, os
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
from pathlib import Path
from collections import Counter, defaultdict

import matplotlib.pyplot as plt

from sklearn.model_selection import StratifiedKFold

```

```

from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

try:
    from lightgbm import LGBMClassifier
    HAS_LGBM = True
except Exception:
    HAS_LGBM = False

try:
    import shap
    HAS_SHAP = True
except Exception:
    HAS_SHAP = False

try:
    from sklearn.inspection import permutation_importance
    HAS_PERM = True
except Exception:
    HAS_PERM = False

print("HAS_LGBM =", HAS_LGBM, "| HAS_SHAP =", HAS_SHAP, "| HAS_PERM =", HAS_PERM)
print("cwd =", os.getcwd())

# %%

FEAT_PARQUET = Path("../data/2features/features.parquet") #<-- 改成你的 features.parquet 路径
OUT_DIR = Path("../data/3models_可视化"); OUT_DIR.mkdir(parents=True, exist_ok=True)

DO_LOLO = True

# %%



def add_rpm_normalized_features(df, feat_cols):
    freq_like = ["freq_peak", "spec_centroid", "freq_std", "spec_rolloff95", "rms_freq"]
    env_cols = ["bpfo_E", "bpfi_E", "bsf_E", "ftf_E"]
    df2 = df.copy()
    fr = (df2["rpm"].astype(float) / 60.0).replace([np.inf, -np.inf], np.nan)
    for c in freq_like:
        if c in df2.columns:
            df2[c+"_order"] = df2[c] / fr

```

```

has_env = [c for c in env_cols if c in df2.columns]
if has_env:
    total = df2[has_env].sum(axis=1) + 1e-12
    for c in has_env:
        df2[c+"_ratio"] = df2[c] / total
new_cols = [c for c in df2.columns if c not in df.columns]
return df2, feat_cols + new_cols

def group_from_segfile(p):
    name = Path(p).stem
    return re.sub(r"_seg\\d+$", "", name)

def barplot(values, labels, title, save_path):
    plt.figure(figsize=(8,5))
    idx = np.argsort(values)[::-1]
    vals_sorted = np.array(values)[idx]
    labels_sorted = np.array(labels)[idx]
    plt.bar(range(len(vals_sorted)), vals_sorted)
    plt.xticks(range(len(vals_sorted)), labels_sorted, rotation=45, ha="right")
    plt.title(title)
    plt.tight_layout()
    plt.savefig(save_path, dpi=150)
    plt.show()

def scatter2d(x2d, y, title, save_path):
    plt.figure(figsize=(6,5))
    for cls in np.unique(y):
        m = (y == cls)
        plt.scatter(x2d[m,0], x2d[m,1], s=10, label=str(cls))
    plt.legend()
    plt.title(title)
    plt.tight_layout()
    plt.savefig(save_path, dpi=150)
    plt.show()

def plot_confusion(cm, labels, title, save_path):
    plt.figure(figsize=(6,5))
    plt.imshow(cm, aspect="auto")
    plt.xticks(range(len(labels)), labels, rotation=45, ha="right")
    plt.yticks(range(len(labels)), labels)
    for i in range(len(labels)):
        for j in range(len(labels)):
            plt.text(j, i, f'{cm[i,j]:.0f}', ha="center", va="center")

```

```

plt.xlabel("Predicted"); plt.ylabel("True")
plt.title(title)
plt.colorbar()
plt.tight_layout()
plt.savefig(save_path, dpi=150)
plt.show()

# %%

if not FEAT_PARQUET.exists():
    print("未找到", FEAT_PARQUET)
    print("当前目录内容: ", os.listdir())
    raise FileNotFoundError("请修改 FEAT_PARQUET 为你的 features.parquet 实际路径")

df = pd.read_parquet(FEAT_PARQUET)
meta_cols = ["seg_file", "label", "load", "rpm", "size_code", "clock_pos"]
for c in meta_cols:
    if c not in df.columns:
        raise ValueError(f"缺少列: {c}")

feat_cols = [c for c in df.columns if c not in meta_cols]
df, feat_cols = add_rpm_normalized_features(df, feat_cols)

labels = df["label"].astype(str).values
groups = df["seg_file"].apply(group_from_segfile).values
X = df[feat_cols].fillna(0).values
y = labels
label_order = sorted(np.unique(y).tolist())

print("样本数 =", len(y), "特征数 =", len(feat_cols), "类别 =", label_order)

# %%

scaler = StandardScaler()
X_std = scaler.fit_transform(X)

pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_std)
scatter2d(X_pca, y, "PCA (2D)", OUT_DIR/"pca_2d.png")

tsne = TSNE(n_components=2, random_state=42, init="pca", perplexity=30, learning_rate='auto')
X_tsne = tsne.fit_transform(X_std)
scatter2d(X_tsne, y, "t-SNE (2D)", OUT_DIR/"tsne_2d.png")

```

```

# %%

if not HAS_LGBM:
    raise ImportError("未安装 LightGBM; 请先安装或改用 RF。")

cv = StratifiedGroupKFold(n_splits=5, shuffle=True, random_state=42)

model = LGBMClassifier(
    objective="multiclass",
    num_class=len(label_order),
    n_estimators=700,
    learning_rate=0.05,
    max_depth=-1,
    subsample=0.9,
    colsample_bytree=0.9,
    reg_lambda=1.0,
    random_state=42
)

cnt = Counter(y)
avg_n = np.mean(list(cnt.values()))
class_weight = {c: avg_n/cnt[c] for c in cnt}
sample_weight_all = np.array([class_weight[t] for t in y], dtype=float)

cms = []
per_class_f1 = defaultdict(list)

for fold, (tr, te) in enumerate(cv.split(X, y, groups=groups), 1):
    X_tr, y_tr = X[tr], y[tr]
    X_te, y_te = X[te], y[te]
    sw_tr = sample_weight_all[tr]
    model.fit(X_tr, y_tr, sample_weight=sw_tr)
    y_pred = model.predict(X_te)
    cm = confusion_matrix(y_te, y_pred, labels=label_order)
    cms.append(cm)
    report = classification_report(y_te, y_pred, labels=label_order, output_dict=True, zero_division=0)
    for cls in label_order:
        per_class_f1[cls].append(report[cls]["f1-score"])

cm_mean = np.mean(np.stack(cms, axis=0), axis=0)
plot_confusion(cm_mean,      label_order,      "LightGBM      -      Mean      Confusion      (5-fold)",
               OUT_DIR/"lgbm_cm_mean.png")

```

```

f1_mean = [np.mean(per_class_f1[cls]) for cls in label_order]
barplot(f1_mean, label_order, "Per-class F1 (LightGBM, 5-fold)", OUT_DIR/"lgbm_f1_per_class.png")

# %%

model_full = LGBMClassifier(
    objective="multiclass",
    num_class=len(label_order),
    n_estimators=700,
    learning_rate=0.05,
    max_depth=-1,
    subsample=0.9,
    colsample_bytree=0.9,
    reg_lambda=1.0,
    random_state=42
)
model_full.fit(X, y, sample_weight=sample_weight_all)
importances = model_full.feature_importances_
topk = 20 if len(importances) >= 20 else len(importances)
top_idx = np.argsort(importances)[::-1][:topk]
barplot(importances[top_idx], [feat_cols[i] for i in top_idx], f"Top-{topk} Feature Importance (LGBM)",
OUT_DIR/"lgbm_feat_importance_topk.png")

# %%

if HAS_SHAP:
    explainer = shap.TreeExplainer(model_full)
    idx = np.random.RandomState(42).choice(len(X), size=min(2000, len(X)), replace=False)
    shap_values = explainer.shap_values(X[idx])
    shap.summary_plot(shap_values, pd.DataFrame(X[idx], columns=feat_cols), show=False)
    plt.title("SHAP Summary (sampled)")
    plt.tight_layout()
    plt.savefig(OUT_DIR/"lgbm_shap_summary.png", dpi=150)
    plt.show()

elif HAS_PERM:
    result = permutation_importance(model_full, X, y, n_repeats=10, random_state=42, n_jobs=-1)
    imp_mean = result.importances_mean
    top_idx = np.argsort(imp_mean)[::-1][:20]
    barplot(imp_mean[top_idx], [feat_cols[i] for i in top_idx], "Permutation Importance (Top-20)",
OUT_DIR/"lgbm_perm_importance_topk.png")
else:
    print("既没有 SHAP 也没有 permutation_importance, 跳过此步。")

```

```

# %%

if DO_LOLO:
    loads = np.array(df["load"].values)
    unique_loads = np.unique(loads)
    all_reports = {}
    for hold in unique_loads:
        m_te = (loads == hold)
        m_tr = ~m_te
        X_tr, y_tr = X[m_tr], y[m_tr]
        X_te, y_te = X[m_te], y[m_te]
        sw_tr = sample_weight_all[m_tr]
        clf = LGBMClassifier(
            objective="multiclass",
            num_class=len(label_order),
            n_estimators=700,
            learning_rate=0.05,
            max_depth=-1,
            subsample=0.9,
            colsample_bytree=0.9,
            reg_lambda=1.0,
            random_state=42
        )
        clf.fit(X_tr, y_tr, sample_weight=sw_tr)
        y_pred = clf.predict(X_te)
        cm = confusion_matrix(y_te, y_pred, labels=label_order)
        plot_confusion(cm, label_order, f"LOLO: hold-out load={hold}", OUT_DIR/f'lolo_load_{hold}_cm.png")
        rep = classification_report(y_te, y_pred, labels=label_order, output_dict=True, zero_division=0)
        all_reports[int(hold)] = rep
    # 保存报告
    pd.DataFrame({k: {f'{cls}_f1': v[cls]['f1-score'] for cls in label_order} for k, v in all_reports.items()}).T.to_csv(OUT_DIR/"lolo_per_class_f1.csv")

```

第三问

本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的

```
from pathlib import Path
```

```

import numpy as np
import pandas as pd
from tqdm import tqdm
from scipy.signal import butter, filtfilt
import pywt

from config import DETAIL_CSV, RAW_DIR, OUT_DIR, FS, L, HOP, LOWC, HIGHC

try:
    from config import CHANNEL_INDEX
except Exception:
    CHANNEL_INDEX = 0

try:
    from config import NORM_PM1
except Exception:
    NORM_PM1 = 0

OUT_PARQUET = OUT_DIR / "target_segments.parquet"
OUT_WAVE     = OUT_DIR / "target_segments_wave"; OUT_WAVE.mkdir(parents=True, exist_ok=True)

def _safe_path(p: str | Path) -> Path:
    p = Path(p)
    if p.is_absolute():
        return p
    if RAW_DIR and (RAW_DIR / p).exists():
        return RAW_DIR / p
    return p

def bandpass(x, fs, low, high, order=4):
    nyq = fs * 0.5
    hi = min(high, nyq * 0.999)
    lo = max(1e-3, low)
    if hi <= lo:
        return x.astype(float)
    b, a = butter(order, [lo/nyq, hi/nyq], btype="bandpass")
    return filtfilt(b, a, x).astype(float)

def wdenoise(x, wavelet="db4", level=4):
    x = np.asarray(x, dtype=float)
    coeffs = pywt.wavedec(x, wavelet, mode="per")
    sigma = (np.median(np.abs(coeffs[-1])) / 0.6745) if len(coeffs[-1]) else 0.0
    uth = sigma * np.sqrt(2 * np.log(len(x))) if sigma > 0 else 0.0
    coeffs = [coeffs[0]] + [pywt.threshold(c, uth, mode="soft") for c in coeffs[1:]]

```

```

        return pywt.waverec(coeffs, wavelet, mode="per")[:len(x)]


def norm_pm1(x):
    m = np.max(np.abs(x))
    return x if (not np.isfinite(m) or m == 0) else (x / m)


def read_signal_csv(path: Path):
    df = pd.read_csv(path, sep=None, engine="python")
    numeric_cols = df.select_dtypes(include=["number"]).columns.tolist()
    for c in list(df.columns):
        if str(c).lower() in ("time", "times", "timestamp"):
            if c in numeric_cols:
                numeric_cols.remove(c)
    if not numeric_cols:
        raise RuntimeError(f"{{path.name}}: 无数值通道列")
    idx = CHANNEL_INDEX if CHANNEL_INDEX is not None else 0
    idx = int(idx) if (0 <= int(idx) < len(numeric_cols)) else 0
    return df[numeric_cols[idx]].to_numpy(dtype=float, copy=False)


def segment_signal(x, L, hop):
    if len(x) < L:
        return []
    n = (len(x) - L) // hop + 1
    return [(k * hop, k * hop + L) for k in range(n)]


def main():
    detail = pd.read_csv(DETAIL_CSV, sep=None, engine="python")
    if "csv_path" not in detail.columns:
        raise RuntimeError("detail.csv 必须包含列 csv_path")
    rows = []
    for _, r in tqdm(detail.iterrows(), total=len(detail), desc="分段"):
        p = _safe_path(r["csv_path"])
        if not Path(p).exists():
            print(f"⚠️ 文件不存在: {p}")
            continue
        x = read_signal_csv(p)
        x = bandpass(x, FS, LOWC, HIGHC)
        x = wdenoise(x)
        for st, ed in segment_signal(x, L, HOP):
            seg = x[st:ed].astype(np.float32)
            if NORM_PM1:
                seg = norm_pm1(seg)
            seg = seg.reshape(1, -1)

```

```

sf = OUT_WAVE / f"{{Path(p).stem}}_seg{st//HOP}.npy"
np.save(sf, seg)

row = {"seg_file": str(sf), "file": Path(p).stem, "seg_idx": st // HOP}
if "rpm" in r.index:
    row["rpm"] = r["rpm"]
rows.append(row)

pd.DataFrame(rows).to_parquet(OUT_PARQUET, index=False)
print(f"✅ 分段完成: {len(rows)} 段 → {OUT_PARQUET}")

if __name__ == "__main__":
    main()

# 本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的

import numpy as np, pandas as pd
from pathlib import Path
from scipy.signal import welch, hilbert
from scipy.stats import kurtosis, skew, entropy
import pywt
from tqdm import tqdm
from config import OUT_DIR, FS

IN_PARQUET = OUT_DIR / "target_segments.parquet"
OUT_PARQUET = OUT_DIR / "target_features.parquet"

BEARING = dict(n=9, d=0.3126, D=1.537)
RPM_DEFAULT = 600.0

def safe_entropy(x, bins=64):
    h, _ = np.histogram(x, bins=bins, density=True)
    return float(entropy(h + 1e-12))

def time_feats(x):
    x = x.astype(float).ravel()
    xa = np.abs(x)
    rms = np.sqrt(np.mean(x**2) + 1e-20)
    mean_abs = np.mean(xa) + 1e-20
    sqrt_abs_mean = np.mean(np.sqrt(xa)) + 1e-20
    return dict(
        mean=float(np.mean(x)), std=float(np.std(x, ddof=1)), rms=float(rms),
        skew=float(skew(x, bias=False)), kurtosis=float(kurtosis(x, fisher=True, bias=False)),
        p2p=float(np.max(x) - np.min(x)), crest_factor=float(np.max(xa) / rms),
        shape_factor=float(rms / mean_abs), impulse_factor=float(np.max(xa) / mean_abs),
    )

```

```

        clearance_factor=float(np.max(xa) / (sqrt_abs_mean**2)),
        sig_entropy=float(safe_entropy(x, bins=64))
    )

def spec_feats(x, fs):
    f, P = welch(x.ravel(), fs=fs, nperseg=min(len(x.ravel()), 2048))
    P += 1e-20
    Psum = np.trapz(P, f)
    pn = P / Psum
    peak = float(f[int(np.argmax(P))])
    centroid = float(np.sum(f * pn))
    spread = float(np.sqrt(np.sum(((f - centroid)**2) * pn)))
    sent = float(-np.sum(pn * np.log(pn)))
    idx95 = min(np.searchsorted(np.cumsum(pn), 0.95), len(f)-1)
    roll95 = float(f[idx95])
    rmsf = float(np.sqrt(np.sum((f**2) * P) / Psum))
    return dict(freq_peak=peak, spec_centroid=centroid, freq_std=spread,
               spec_entropy=sent, spec_rolloff95=roll95, rms_freq=rmsf)

def char_freqs(rpm, n, d, D):
    fr = rpm/60.0
    bpfo = fr * n/2.0 * (1 - d/D)
    bpfi = fr * n/2.0 * (1 + d/D)
    bsf = fr * (D/d) * (1 - (d/D)**2)
    fft = 0.5 * fr * (1 - d/D)
    return fr, bpfo, bpfi, bsf, fft

def envelope_feats(x, fs, rpm, tol=0.10):
    env = np.abs(hilbert(x.ravel()))
    f, P = welch(env, fs=fs, nperseg=min(len(env), 4096))
    P += np.finfo(float).eps
    out = {}
    _, bpfo, bpfi, bsf, fft = char_freqs(rpm, **BEARING)
    for name, f0 in {"bpfo": bpfo, "bpfi": bpfi, "bsf": bsf, "fft": fft}.items():
        lo, hi = (1-tol)*f0, (1+tol)*f0
        m = (f >= lo) & (f <= hi)
        out[f'{name}_E'] = float(np.trapz(P[m], f[m])) if np.any(m) else 0.0
    return out

def wp_feats(x, wavelet="db4", level=3):
    wp = pywt.WaveletPacket(x.ravel(), wavelet, maxlevel=level)
    nodes = [n.path for n in wp.get_level(level, "freq")]
    Es = [np.sum(np.square(wp[n].data)) for n in nodes]

```

```

total = np.sum(Es) + 1e-20
p = np.array(Es)/total
H = float(-np.sum(p * np.log(p + 1e-12)))
out = {f"wp_E{i}": float(p[i]) for i in range(len(p))}
out["wp_entropy"] = H
return out

def main():
    df = pd.read_parquet(IN_PARQUET)
    rows = []
    for _, r in tqdm(df.iterrows(), total=len(df)):
        x = np.load(r["seg_file"]).astype(float)
        rpm = RPM_DEFAULT
        feat = dict(seg_file=r["seg_file"], file=r["file"], seg_idx=r["seg_idx"])
        feat.update(time_feats(x))
        feat.update(spec_feats(x, FS))
        feat.update(envelope_feats(x, FS, rpm))
        feat.update(wp_feats(x))
        fr, bpfo, bpfi, bsf, ftf = char_freqs(RPM_DEFAULT, **BEARING)
        feat.update(dict(fr=fr, bpfo=bpfo, bpfi=bpfi, bsf=bsf, ftf=ftf))
        rows.append(feat)
    pd.DataFrame(rows).to_parquet(OUT_PARQUET, index=False)
    print(f'✅ 特征提取完成: {len(rows)} 段 → {OUT_PARQUET}')

if __name__ == "__main__":
    main()
# 本程序及代码是在人工智能工具 ChatGPT (GPT-5, OpenAI, 2025 年 3 月) 辅助下完成的

import json, numpy as np, pandas as pd
from pathlib import Path
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import joblib
from config import OUT_DIR, ARTifacts, USE_CORAL, RANDOM_SEED

IN_FEAT  = OUT_DIR / "target_features.parquet"
OUT_SEG  = OUT_DIR / "target_segment_preds.csv"
OUT_FILE = OUT_DIR / "target_file_preds.csv"
OUT_PNG  = OUT_DIR / "pca_before_after.png"

def load_artifacts():
    with open(ARTifacts/"feat_cols.json", "r", encoding="utf-8") as f:
        feat_cols = json.load(f)

```

```

with open(ARTifacts/"classes.json", "r", encoding="utf-8") as f:
    classes = json.load(f)
    scaler = joblib.load(ARTifacts/"scaler.pkl")
    model = joblib.load(ARTifacts/"model.pkl")
    return feat_cols, classes, scaler, model

def coral_align(Xs, Xt):
    Cs = np.cov(Xs, rowvar=False) + np.eye(Xs.shape[1]) * 1e-3
    Ct = np.cov(Xt, rowvar=False) + np.eye(Xt.shape[1]) * 1e-3
    Es, Us = np.linalg.eigh(Cs); Es[Es < 1e-6] = 1e-6
    Et, Ut = np.linalg.eigh(Ct); Et[Et < 1e-6] = 1e-6
    Cs_inv_sqrt = Us @ np.diag(Es**-0.5) @ Us.T
    Ct_sqrt = Ut @ np.diag(Et**0.5) @ Ut.T
    return Xt @ Cs_inv_sqrt @ Ct_sqrt

def pca_plot(X_before, X_after, save_path):
    pca = PCA(n_components=2, random_state=RANDOM_SEED)
    A = pca.fit_transform(X_before)
    B = pca.fit_transform(X_after)
    plt.figure(figsize=(10,4))
    plt.subplot(1,2,1); plt.scatter(A[:,0], A[:,1], s=5, alpha=0.5); plt.title("Target PCA (before)")
    plt.subplot(1,2,2); plt.scatter(B[:,0], B[:,1], s=5, alpha=0.5); plt.title("Target PCA (after CORAL)")
    plt.tight_layout(); plt.savefig(save_path, dpi=160); plt.close()

def aggregate_file_level(df_seg):
    hard = df_seg.groupby("file")["pred"].agg(lambda s: s.value_counts().idxmax()).to_frame("hard_label")
    proba_cols = [c for c in df_seg.columns if c.startswith("proba_")]
    soft = df_seg.groupby("file")[proba_cols].mean()
    soft["soft_label"] = soft.idxmax(axis=1).str.replace("proba_", "", regex=False)
    return hard.join(soft)

def main():
    feat_cols, classes, scaler_src, model = load_artifacts()
    df = pd.read_parquet(IN_FEAT)

    X = pd.DataFrame(index=df.index)
    for c in feat_cols:
        X[c] = df[c] if c in df.columns else 0.0
    X = X[feat_cols].astype(float).values

    X_std = scaler_src.transform(X)

    if USE_CORAL:

```

```

X_aligned = coral_align(X_std, X_std)
pca_plot(X_std, X_aligned, OUT_PNG)
X_use = X_aligned

else:
    X_use = X_std

proba = model.predict_proba(X_use)
y_pred = [classes[i] for i in proba.argmax(axis=1)]

out = df[["seg_file", "file", "seg_idx"]].copy()
out["pred"] = y_pred
for i, cls in enumerate(classes):
    out[f"proba_{cls}"] = proba[:, i]
out.to_csv(OUT_SEG, index=False, encoding="utf-8-sig")

agg = aggregate_file_level(out)
agg.to_csv(OUT_FILE, encoding="utf-8-sig")

print(f"迁移诊断完成: 段级→{OUT_SEG}, 文件级→{OUT_FILE}")
if USE_CORAL:
    print(f"生成 PCA 可视化: {OUT_PNG}")

if __name__ == "__main__":
    main()

# %%
import pandas as pd
from pathlib import Path
import numpy as np

# --- 配置 ---
INDEX_PATH = Path("../第3问_CNN/index.csv")
TARGET_DOMAIN_PREFIX = "../第3问_CNN/cleaned/目标域/"

# --- 定义本研究要查找的列名映射 ---
# {'标准名': ['可能存在的列名片段 1', '片段 2', ...]}
COLUMN_SEARCH_MAP = {
    'csv_path': ['csv_path'],
    'fs': ['赫兹', '采样率'],
    'rpm': ['转速'],
    'label': ['故障类']
}

```

```

# --- 加载与清洗索引文件 ---
print(f'正在从 '{INDEX_PATH}' 加载索引文件...')

try:
    # 尝试用 utf-8-sig 编码，这能更好地处理带 BOM 的 CSV 文件
    df_index = pd.read_csv(INDEX_PATH, encoding='utf-8-sig')

    # 原始列名，用于调试
    original_cols = df_index.columns.tolist()
    # 清洗列名，转为小写并去除首尾空格
    df_index.columns = [str(c).strip().lower() for c in df_index.columns]

    found_cols = {}
    for standard_name, possible_names in COLUMN_SEARCH_MAP.items():
        for col in df_index.columns:
            # 检查任一可能的名字是否存在于列名中
            if any(p_name in col for p_name in possible_names):
                found_cols[standard_name] = col
                break # 找到后就跳出内层循环

    # 检查是否所有必要的列都找到了
    required_keys = ['csv_path', 'fs', 'rpm', 'label']
    if not all(key in found_cols for key in required_keys):
        missing = [key for key in required_keys if key not in found_cols]
        raise ValueError(f'错误：索引文件中缺少必要的列: {missing}。 原始列为: {original_cols}')

    # 筛选本研究需要的列
    df_filtered = df_index[list(found_cols.values())].copy()

    # 构建重命名映射并执行
    rename_dict = {v: k for k, v in found_cols.items()}
    df_filtered.rename(columns=rename_dict, inplace=True)

    # 转换数据类型，无效值设为 NaN
    df_filtered['fs'] = pd.to_numeric(df_filtered['fs'], errors='coerce')
    df_filtered['rpm'] = pd.to_numeric(df_filtered['rpm'], errors='coerce')

# --- 分离源域和目标域 ---
df_filtered['domain'] = np.where(df_filtered['csv_path'].str.startswith(TARGET_DOMAIN_PREFIX),
'target', 'source')

# 暂时只处理源域数据
df_source = df_filtered[df_filtered['domain'] == 'source'].copy()

```

```

print("\n 索引加载并清洗完成！")
print(f"\n 分离出源域和目标域: \n{df_filtered['domain'].value_counts()}")
print("\n 已筛选出源域数据进行处理， 预览:")
display(df_source.head())

except FileNotFoundError:
    print(f'错误： 无法在 '{INDEX_PATH}' 找到索引文件。请检查路径是否正确。')
except Exception as e:
    print(f'发生错误: {e}')

# %%
import numpy as np
from pathlib import Path

def select_channel_from_file(file_path: str) -> np.ndarray | None:
    """
    Reads data from a given CSV file and selects a single channel based on the priority DE -> FE -> BA.

    Args:
        file_path (str): The path to the CSV file.

    Returns:
        np.ndarray | None: A 1D NumPy array of the signal data if successful, otherwise None.
    """

try:
    p = Path(file_path)
    if not p.exists():
        print(f"Error: File not found at {file_path}")
        return None

    df_signal = pd.read_csv(p)

    # Standardize column names to uppercase for consistent checking
    df_signal.columns = [str(c).strip().upper() for c in df_signal.columns]

    signal_data = None
    if 'DE' in df_signal.columns:
        signal_data = df_signal['DE'].values
    elif 'FE' in df_signal.columns:
        signal_data = df_signal['FE'].values
    elif 'BA' in df_signal.columns:
        signal_data = df_signal['BA'].values

```

```

else:
    print(f"Warning: No priority channel (DE, FE, BA) found in {p.name}.")
    return None

    # Ensure data is float and handle potential non-numeric values
    signal_data = pd.to_numeric(signal_data, errors='coerce')

    # Check for NaNs that may have been introduced by 'coerce'
    nan_count = np.isnan(signal_data).sum()
    if nan_count > 0:
        print(f"Warning: Found and filled {nan_count} non-numeric values in {p.name}.")
        # A simple strategy: fill NaNs with the mean of the valid points
        mean_val = np.nanmean(signal_data)
        signal_data = np.nan_to_num(signal_data, nan=mean_val)

    return signal_data.astype(float)

except Exception as e:
    print(f"An error occurred while processing {file_path}: {e}")
    return None

# --- Function Test ---
print("--- Testing the Channel Selector Function ---")

# Get the first file path from our source dataframe
first_row = df_source.iloc[0]
file_to_test = first_row['csv_path']

print(f"Attempting to load signal from: {file_to_test}")

# Call the function
signal_array = select_channel_from_file(file_to_test)

if signal_array is not None:
    print("\n✓ Success!")
    print(f"  Signal Length: {len(signal_array)}")
    print(f"  Data Type: {signal_array.dtype}")
    print(f"  First 5 values: {signal_array[:5]}")
else:
    print("\n✗ Failed to load signal.")

# %%%
from scipy.signal import resample

```

```

# --- Configuration ---
TARGET_FS = 32000 # Hz

def resample_signal(signal_array: np.ndarray, original_fs: int, target_fs: int = TARGET_FS) -> np.ndarray:
    """
    Resamples a signal to a target sampling frequency.

    Args:
        signal_array (np.ndarray): The input 1D signal array.
        original_fs (int): The original sampling frequency of the signal.
        target_fs (int): The target sampling frequency.

    Returns:
        np.ndarray: The resampled signal array.

    """
    if original_fs == target_fs:
        # No resampling needed if the rates are already the same
        return signal_array

    # Calculate the number of samples in the new signal
    duration = len(signal_array) / original_fs
    num_target_samples = int(duration * target_fs)

    # Perform resampling
    resampled_signal = resample(signal_array, num_target_samples)

    return resampled_signal.astype(np.float32) # Use float32 to save memory

# --- Function Test ---
print("--- Testing the Resampling Function ---")

# We use the signal_array loaded in the previous cell
# We need its original sampling rate from our dataframe
original_sampling_rate = first_row['fs']

print(f'Original FS: {original_sampling_rate} Hz, Original Length: {len(signal_array)}')
print(f'Target FS: {TARGET_FS} Hz')

# Call the resampling function
resampled_array = resample_signal(signal_array, original_fs=original_sampling_rate)

if resampled_array is not None:

```

```

print("\n✓ Success!")
print(f"  New Resampled Length: {len(resampled_array)}")
print(f"  Data Type: {resampled_array.dtype}")
print(f"  First 5 values: {resampled_array[:5]}")

else:
    print("\n✗ Failed to resample signal.")

# %%
import numpy as np

# --- Configuration ---
PPR = 720 # Points Per Revolution. A higher value captures more detail. 720 is a good start.

# 只需要更新 angle_sync 函数即可

def angle_sync(signal_array: np.ndarray, rpm: float, fs: int = TARGET_FS, ppr: int = PPR) -> np.ndarray | None:
    """
    Performs angular synchronization on a time-domain signal.
    """

    # --- 新增的检查 ---
    # 如果输入的信号数组是空的，直接返回 None
    if signal_array.size == 0:
        print("Warning: Received an empty signal array. Skipping angular sync.")
        return None

    if rpm <= 0 or not np.isfinite(rpm):
        print(f"Warning: Invalid RPM ({rpm}). Skipping angular sync.")
        return None

    # Rotational frequency in Hz (revolutions per second)
    fr = rpm / 60.0

    # Original time vector for the input signal
    t_original = np.arange(len(signal_array)) / fs

    # Calculate the total number of full revolutions in the signal
    total_revs = t_original[-1] * fr
    if total_revs < 1:
        print(f"Warning: Signal is too short ({total_revs:.2f} revs) for angular sync. Need at least 1 revolution.")
        return None

    # (函数的其余部分保持不变)
    num_full_revs = int(np.floor(total_revs))

```

```

total_angle = 2 * np.pi * num_full_revs
num_new_points = num_full_revs * ppr
angle_new = np.linspace(0, total_angle, num=num_new_points, endpoint=False)
angle_original = 2 * np.pi * fr * t_original
angle_domain_signal = np.interp(angle_new, angle_original, signal_array)

return angle_domain_signal.astype(np.float32)

# --- Function Test ---

print("--- Testing the Angular Synchronization Function ---")

# We use the resampled_array from the previous cell
# We need its original RPM from our dataframe
rpm_value = first_row['rpm']

print(f'Input Signal Length: {len(resampled_array)}, RPM: {rpm_value}')

# Call the angular sync function
angle_synced_array = angle_sync(resampled_array, rpm=rpm_value)

if angle_synced_array is not None:
    print("\n

```

Args:

- angle_domain_signal (np.ndarray): 输入的一维角度域信号。
- ppr (int): 每转点数，在角度域中，它扮演了“采样率”的角色。

Returns:

- np.ndarray: 代表信号的二维图像矩阵。

```
"""
# 在角度域信号上进行 STFT
# 这里的 fs 参数就是本研究的 ppr, 因为单位是“点/转”
# 输出的频率 f 的单位将是“阶次 (Order)”

orders, _, Zxx = stft(
    angle_domain_signal,
    fs=ppr,
    nperseg=N_PER_SEG,
    nooverlap=N_OVERLAP
)

# 本研究只需要幅值，并进行对数变换以压缩动态范围，便于模型学习
# np.log1p(x) 等同于 np.log(1+x)，可以避免对 0 取对数
stft_image = np.log1p(np.abs(Zxx))

return stft_image.astype(np.float32)

# --- 函数测试 ---
print("--- 测试 STFT 图像生成函数 ---")

# 使用上一单元格生成的 angle_synced_array
stft_result_image = generate_stft_image(angle_synced_array)

if stft_result_image is not None:
    print("\n✓ 成功生成 STFT 图像!")
    print(f"    图像尺寸 (阶次 bins, 时间 steps): {stft_result_image.shape}")
    print(f"    数据类型: {stft_result_image.dtype}")

# --- 可视化 ---
plt.figure(figsize=(10, 6))
plt.imshow(stft_result_image, aspect='auto', origin='lower',
           extent=[0, len(angle_synced_array)/PPR, 0, PPR/2])
plt.colorbar(label='对数幅值')
plt.xlabel('转数 (Revolutions)')
plt.ylabel('阶次 (Order)')
plt.title('生成的阶次-时间图')
```

```

plt.show()

else:
    print("\n❌ 生成 STFT 图像失败。")

# %%

import pandas as pd
from pathlib import Path
from tqdm.notebook import tqdm
import os

# --- 配置 ---
# 创建一个目录来存放本研究生成的图像
OUTPUT_DIR = Path("./第 3 问_CNN/processed_images")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

# --- 批量处理循环 ---
processed_metadata = [] # 用于存储成功处理的文件的信息

# 使用 tqdm 来显示进度条
for index, row in tqdm(df_source.iterrows(), total=len(df_source), desc="正在处理源域文件"):

    # 获取当前文件的元数据
    file_path_str = row['csv_path']
    original_fs = row['fs']
    rpm = row['rpm']
    label = row['label']

    # --- Step 1: 智能通道选择 ---
    signal = select_channel_from_file(file_path_str)
    if signal is None:
        print(f"跳过: 无法从 {file_path_str} 加载信号")
        continue

    # --- Step 2: 重采样 ---
    resampled = resample_signal(signal, original_fs=original_fs)
    if resampled is None:
        print(f"跳过: 重采样失败 {file_path_str}")
        continue

    # --- Step 3: 角同步 ---
    angle_synced = angle_sync(resampled, rpm=rpm)
    if angle_synced is None:

```

```

print(f"跳过: 角同步失败 {file_path_str}")
continue

# --- Step 4: 生成 STFT 图像 ---
stft_image = generate_stft_image(angle_synced)
if stft_image is None:
    print(f"跳过: STFT 图像生成失败 {file_path_str}")
    continue

# --- Step 5: 保存图像并记录元数据 ---
# 创建一个有意义的文件名
original_stem = Path(file_path_str).stem
output_filename = f"{original_stem}.npy"
output_path = OUTPUT_DIR / output_filename

# 保存为 .npy 文件, 这比图片格式更能保留原始数据
np.save(output_path, stft_image)

# 记录成功处理文件的信息
processed_metadata.append({
    'image_path': str(output_path),
    'original_file': file_path_str,
    'label': label,
    'rpm': rpm,
    'original_fs': original_fs
})

# --- 创建最终的元数据文件 ---
df_processed = pd.DataFrame(processed_metadata)
manifest_path = OUTPUT_DIR.parent / "source_manifest.csv"
df_processed.to_csv(manifest_path, index=False, encoding='utf-8-sig')

print(f"\n🎉 全部处理完成!")
print(f"成功处理了 {len(df_processed)} / {len(df_source)} 个文件。")
print(f"所有图像已保存至: {OUTPUT_DIR}")
print(f"新的元数据文件已保存至: {manifest_path}")
display(df_processed.head())

# %%
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler

```

```

import pandas as pd
import numpy as np
import gc

from pathlib import Path
from sklearn.utils.class_weight import compute_class_weight
from tqdm.notebook import tqdm
import torch.nn.functional as F

# =====
# 单元格 1: 所有定义（类和函数）
# =====

# --- 配置 ---
MANIFEST_PATH = Path("../第 3 问_CNN/source_manifest.csv")
BATCH_SIZE = 32
FIXED_WIDTH = 4096
KNOWN_IMAGE_SHAPE = (1, 129, 4096) # (Channels, Height, Width)

# --- Dataset 定义 ---
class BearingDataset(Dataset):
    def __init__(self, manifest_path: Path, fixed_width: int):
        self.manifest = pd.read_csv(manifest_path)
        self.fixed_width = fixed_width
        self.classes = sorted(self.manifest['label'].unique())
        self.class_to_idx = {cls_name: i for i, cls_name in enumerate(self.classes)}
    def __len__(self):
        return len(self.manifest)
    def _pad_or_truncate(self, image: np.ndarray) -> np.ndarray:
        height, width = image.shape
        if width > self.fixed_width: return image[:, :self.fixed_width]
        elif width < self.fixed_width:
            padding = self.fixed_width - width
            return np.pad(image, ((0, 0), (0, padding)), mode='constant', constant_values=0)
        return image
    def __getitem__(self, idx):
        row = self.manifest.iloc[idx]
        image = np.load(row['image_path'])
        image = self._pad_or_truncate(image)
        label_idx = self.class_to_idx[row['label']]
        return torch.from_numpy(image).unsqueeze(0).float(), torch.tensor(label_idx, dtype=torch.long)

# --- CNN 模型定义 ---

```

```

class SimpleCNN(nn.Module):
    def __init__(self, num_classes: int, input_shape: tuple):
        super(SimpleCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(in_channels=input_shape[0], out_channels=16, kernel_size=(3, 5), stride=1,
                     padding=1),
            nn.BatchNorm2d(16), nn.ReLU(), nn.MaxPool2d(kernel_size=(2, 4)),
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32), nn.ReLU(), nn.MaxPool2d(kernel_size=(2, 4)),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64), nn.ReLU(), nn.MaxPool2d(kernel_size=(2, 4))
        )
        dummy_input = torch.randn(1, *input_shape)
        flattened_size = self.features(dummy_input).view(1, -1).size(1)
        self.classifier = nn.Sequential(
            nn.Linear(in_features=flattened_size, out_features=128),
            nn.ReLU(), nn.Dropout(0.5),
            nn.Linear(in_features=128, out_features=num_classes)
        )
    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        return self.classifier(x)

# =====
# 单元格 2: 数据加载和训练执行
# =====

# --- 1. 数据加载器设置 (带过采样) ---
print("--- 准备数据集和加载器 ---")
source_dataset = BearingDataset(manifest_path=MANIFEST_PATH, fixed_width=FIXED_WIDTH)
print(f'找到的类别: {source_dataset.class_to_idx}')

labels_list = pd.read_csv(MANIFEST_PATH)['label'].tolist()
class_names = source_dataset.classes
class_weights = compute_class_weight('balanced', classes=np.array(class_names), y=labels_list)
class_weights_dict = {cls: weight for cls, weight in zip(class_names, class_weights)}
sample_weights = [class_weights_dict[label] for label in labels_list]
sampler = WeightedRandomSampler(weights=sample_weights, num_samples=len(sample_weights),
                                 replacement=True)

oversample_loader = DataLoader(
    dataset=source_dataset, batch_size=BATCH_SIZE, sampler=sampler, num_workers=2
)

```

```

)
print("✅ 带过采样功能的数据加载器准备就绪！")

# --- 2. 训练执行 ---
print("\n--- 开始训练 (最终整合版) ---")
NUM_EPOCHS = 50
LEARNING_RATE = 0.001
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f'使用设备: {device}')

model = SimpleCNN(num_classes=len(source_dataset.classes), input_shape=KNOWN_IMAGE_SHAPE)
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)

for epoch in range(NUM_EPOCHS):
    model.train()
    running_loss = 0.0
    progress_bar = tqdm(oversample_loader, desc=f"Epoch {epoch+1}/{NUM_EPOCHS}")
    for images, labels in progress_bar:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        progress_bar.set_postfix({'loss': f'{loss.item():.4f}'})
    epoch_loss = running_loss / len(oversample_loader)
    print(f'Epoch {epoch+1}/{NUM_EPOCHS} 完成, 平均损失: {epoch_loss:.4f}')

    del images, labels, outputs
    torch.cuda.empty_cache() # 清理 GPU 内存
    gc.collect() # 强制垃圾回收

print("\n--- 训练完成 ---")
SAVE_PATH = "./source_domain_model_oversampled.pth"
torch.save(model.state_dict(), SAVE_PATH)
print(f'优化后的模型已保存至: {SAVE_PATH}')

# %%
import torch
import numpy as np

```

```

import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# --- 1. 加载训练好的模型 ---
print("--- 加载模型并准备评估 ---")
images_batch, _ = next(iter(oversample_loader))
# 确保模型结构和之前定义的一致
# 本研究需要重新实例化模型，然后加载权重
eval_model = SimpleCNN(num_classes=len(source_dataset.classes),
input_shape=tuple(images_batch.shape[1:]))
eval_model.load_state_dict(torch.load("../第3问_CNN/source_domain_model_oversampled.pth"))

# 将模型移动到设备并设置为评估模式
eval_model.to(device)
eval_model.eval() # 关键一步：将模型设置为评估模式（会关闭 Dropout 等）

# --- 2. 在源域数据上进行预测 ---
all_preds = []
all_labels = []

# torch.no_grad() 上下文管理器可以禁用梯度计算，节省内存和计算资源
with torch.no_grad():
    for images, labels in tqdm(oversample_loader, desc="正在评估模型"):
        images = images.to(device)
        labels = labels.to(device)

        # 前向传播
        outputs = eval_model(images)

        # 获取预测结果（概率最高的类别）
        _, predicted = torch.max(outputs.data, 1)

        # 收集所有预测和真实标签
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# --- 3. 计算并打印评估指标 ---
# 获取类别名称，用于报告和图表
class_names = source_dataset.classes

accuracy = accuracy_score(all_labels, all_preds)

```

```

print(f"\n 模型在源域上的总体准确率: {accuracy * 100:.2f} %")

print("\n 分类报告:")
print(classification_report(all_labels, all_preds, target_names=class_names, zero_division=0))

# --- 4. 绘制混淆矩阵 ---
cm = confusion_matrix(all_labels, all_preds)

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.title('源域模型混淆矩阵')
plt.ylabel('真实标签 (True Label)')
plt.xlabel('预测标签 (Predicted Label)')
plt.show()

# %%%
import torch
import pandas as pd
from pathlib import Path
from tqdm.notebook import tqdm
import numpy as np
from scipy.signal import resample, stft
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F

# --- 1. 所有辅助函数定义 ---

def select_channel_from_file(file_path: str) -> np.ndarray | None:
    """
    从 CSV 读取信号，优先级为 DE -> FE -> BA -> NO
    """
    try:
        p = Path(file_path)
        if not p.exists(): return None
        df_signal = pd.read_csv(p)
        df_signal.columns = [str(c).strip().upper() for c in df_signal.columns]

        signal_data = None
        # 依次检查每个优先级的列是否存在且包含有效数据
        if 'DE' in df_signal.columns and df_signal['DE'].notna().any():
            signal_data = df_signal['DE'].values
    except Exception as e:
        print(f"Error reading file {file_path}: {e}")
        return None
    return signal_data

```

```

        elif 'FE' in df_signal.columns and df_signal['FE'].notna().any():
            signal_data = df_signal['FE'].values
        elif 'BA' in df_signal.columns and df_signal['BA'].notna().any():
            signal_data = df_signal['BA'].values
        elif 'NO' in df_signal.columns and df_signal['NO'].notna().any(): # 关键修正：添加对 NO 列的回退
            # print(f"Info: File {p.name} is using 'NO' column as signal.")
            signal_data = df_signal['NO'].values
        else:
            return None # 如果所有列都无效，则返回 None

        signal_data = pd.to_numeric(signal_data, errors='coerce')
        if np.isnan(signal_data).sum() > 0:
            mean_val = np.nanmean(signal_data)
            signal_data = np.nan_to_num(signal_data, nan=mean_val)
        return signal_data.astype(float)

    except Exception:
        return None

# (其余辅助函数与之前相同)

def resample_signal(signal_array: np.ndarray, original_fs: int, target_fs: int = 32000) -> np.ndarray:
    if original_fs == target_fs: return signal_array
    duration = len(signal_array) / original_fs
    num_target_samples = int(duration * target_fs)
    if num_target_samples == 0: return np.array([])
    return resample(signal_array, num_target_samples).astype(np.float32)

def angle_sync(signal_array: np.ndarray, rpm: float, fs: int = 32000, ppr: int = 720) -> np.ndarray | None:
    if signal_array.size == 0: return None
    if rpm <= 0 or not np.isfinite(rpm): return None
    fr = rpm / 60.0
    t_original = np.arange(len(signal_array)) / fs
    if len(t_original) == 0: return None
    total_revs = t_original[-1] * fr
    if total_revs < 1: return None
    num_full_revs = int(np.floor(total_revs))
    total_angle = 2 * np.pi * num_full_revs
    num_new_points = num_full_revs * ppr
    angle_new = np.linspace(0, total_angle, num=num_new_points, endpoint=False)
    angle_original = 2 * np.pi * fr * t_original
    angle_domain_signal = np.interp(angle_new, angle_original, signal_array)
    return angle_domain_signal.astype(np.float32)

def generate_stft_image(angle_domain_signal: np.ndarray, ppr: int = 720) -> np.ndarray:

```

```

orders, _, Zxx = stft(angle_domain_signal, fs=ppr, nperseg=256, nooverlap=192)
return np.log1p(np.abs(Zxx)).astype(np.float32)

def preprocess_single_file(file_path, original_fs, rpm):
    signal = select_channel_from_file(file_path)
    if signal is None: return None
    resampled = resample_signal(signal, original_fs=original_fs)
    if resampled is None or resampled.size==0: return None
    angle_synced = angle_sync(resampled, rpm=rpm)
    if angle_synced is None: return None
    stft_image = generate_stft_image(angle_synced)
    if stft_image is None: return None
    height, width = stft_image.shape
    if width > 4096: stft_image = stft_image[:, :4096]
    elif width < 4096:
        padding_needed = 4096 - width
        stft_image = np.pad(stft_image, ((0, 0), (0, padding_needed)), mode='constant', constant_values=0)
    return stft_image

# --- 2. 主执行流程 ---
print("--- 开始最终诊断流程 (已更新 NO 列逻辑) ---")

# a. 准备目标域数据
df_target = df_filtered[df_filtered['domain'] == 'target'].copy()
print(f'找到 {len(df_target)} 个目标域文件进行诊断。')

# b. 加载模型 (需要重新实例化模型, 以防之前的定义丢失)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
final_model = SimpleCNN(num_classes=len(source_dataset.classes),
input_shape=tuple(images_batch.shape[1:]))
MODEL_PATH = "./source_domain_model_oversampled.pth"
final_model.load_state_dict(torch.load(MODEL_PATH))
final_model.to(device)
final_model.eval()
print(f'成功加载模型: {MODEL_PATH}')

# c. 循环处理并预测
predictions = []
idx_to_class = {i: cls for cls, i in source_dataset.class_to_idx.items()}

with torch.no_grad():
    for index, row in tqdm(df_target.iterrows(), total=len(df_target), desc="正在诊断目标域文件"):
        file_path_str, original_fs, rpm = row['csv_path'], row['fs'], row['rpm']

```

```

image = preprocess_single_file(file_path_str, original_fs, rpm)
if image is None:
    predicted_label = "处理失败"
else:
    image_tensor = torch.from_numpy(image).unsqueeze(0).unsqueeze(0).float().to(device)
    output = final_model(image_tensor)
    _, predicted_idx = torch.max(output.data, 1)
    predicted_label = idx_to_class[predicted_idx.item()]
predictions.append({'target_file': Path(file_path_str).name, 'predicted_fault_type': predicted_label})
del image, image_tensor, output
torch.cuda.empty_cache() # 清理 GPU 内存
gc.collect() # 强制垃圾回收

# d. 展示结果
df_results = pd.DataFrame(predictions)
print("\n--- 目标域故障诊断结果 ---")
display(df_results)

# %%
# --- 配置 ---
TARGET_OUTPUT_DIR = Path("./processed_images/target")
TARGET_OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

# --- 批量处理目标域文件 ---
target_processed_metadata = []
for index, row in tqdm(df_target.iterrows(), total=len(df_target), desc="正在处理目标域文件"):
    file_path_str, original_fs, rpm = row['csv_path'], row['fs'], row['rpm']

    # 使用本研究之前定义好的完整预处理函数
    image = preprocess_single_file(file_path_str, original_fs, rpm)

    if image is not None:
        original_stem = Path(file_path_str).stem
        output_filename = f"{original_stem}.npy"
        output_path = TARGET_OUTPUT_DIR / output_filename
        np.save(output_path, image)

        target_processed_metadata.append({
            'image_path': str(output_path),
            'original_file': file_path_str
        })

# --- 创建目标域的元数据文件 ---

```

```

df_target_processed = pd.DataFrame(target_processed_metadata)
target_manifest_path = TARGET_OUTPUT_DIR.parent / "target_manifest.csv"
df_target_processed.to_csv(target_manifest_path, index=False, encoding='utf-8-sig')

print(f"\n🎉 目标域数据处理完成！")
print(f"成功处理了 {len(df_target_processed)} / {len(df_target)} 个文件。")
print(f"图像已保存至: {TARGET_OUTPUT_DIR}")
print(f"元数据文件已保存至: {target_manifest_path}")

# %%
# --- 源域加载器 (使用过采样) ---
source_manifest_path = Path("./processed_images/source_manifest.csv")
source_dataset = BearingDataset(manifest_path=source_manifest_path, fixed_width=FIXED_WIDTH)
# (这里的采样器代码来自之前的 Cell 7-A, 确保它已被执行)
source_labels_list = pd.read_csv(source_manifest_path)[['label']].tolist()
class_weights = compute_class_weight('balanced', classes=np.array(source_dataset.classes),
y=source_labels_list)
class_weights_dict = {cls: weight for cls, weight in zip(source_dataset.classes, class_weights)}
sample_weights = [class_weights_dict[label] for label in source_labels_list]
sampler = WeightedRandomSampler(weights=sample_weights, num_samples=len(sample_weights),
replacement=True)

source_loader = DataLoader(
    dataset=source_dataset,
    batch_size=BATCH_SIZE,
    sampler=sampler,
    num_workers=2
)

# --- 目标域加载器 (不需要标签和采样器) ---
class TargetDataset(Dataset): # 为目标域创建一个简单的数据集类
    def __init__(self, manifest_path: Path, fixed_width: int):
        self.manifest = pd.read_csv(manifest_path)
        self.fixed_width = fixed_width
    def __len__(self):
        return len(self.manifest)
    def __getitem__(self, idx):
        # 目标域数据没有标签
        image_path = self.manifest.iloc[idx]['image_path']
        image = np.load(image_path)
        # (这里的 pad/truncate 逻辑与 BearingDataset 中一致)
        height, width = image.shape
        if width > self.fixed_width: image = image[:, :self.fixed_width]

```

```

        elif width < self.fixed_width:
            padding_needed = self.fixed_width - width
            image = np.pad(image, ((0, 0), (0, padding_needed)), mode='constant', constant_values=0)
            return torch.from_numpy(image).unsqueeze(0).float()

target_dataset = TargetDataset(manifest_path=target_manifest_path, fixed_width=FIXED_WIDTH)
target_loader = DataLoader(
    dataset=target_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True, # 目标域数据可以打乱
    num_workers=2
)

print("✅ 源域和目标域的数据加载器均已准备就绪！")

# %%
import torch
import torch.nn as nn
import torch.optim as optim
from tqdm.notebook import tqdm

# --- 1. 定义 CORAL 损失函数 ---
def coral_loss(source, target):
    """
    计算源域和目标域特征之间的 CORAL 损失。
    Args:
        source: 源域的特征张量 (batch_size, num_features)
        target: 目标域的特征张量 (batch_size, num_features)
    Returns:
        coral_loss: 计算出的损失值
    """
    d = source.size(1) # 特征维度

    # a. 计算源域的协方差矩阵
    source_mean = torch.mean(source, 0, keepdim=True)
    source_centered = source - source_mean
    source_cov = (source_centered.t() @ source_centered) / (source.size(0) - 1)

    # b. 计算目标域的协方差矩阵
    target_mean = torch.mean(target, 0, keepdim=True)
    target_centered = target - target_mean
    target_cov = (target_centered.t() @ target_centered) / (target.size(0) - 1)

```

```

# c. 计算两者差异的 Frobenius 范数的平方
loss = torch.mean(torch.mul((source_cov - target_cov), (source_cov - target_cov)))

return loss

# --- 2. 训练配置 ---
NUM_EPOCHS = 100 # 域适应训练通常需要足够的轮次
LEARNING_RATE = 0.0001 # 使用稍低的学习率可能更稳定
LAMBDA_CORAL = 1.0 # CORAL 损失的权重超参数，可以调整

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 实例化一个全新的模型用于域适应训练
da_model = SimpleCNN(num_classes=len(source_dataset.classes), input_shape=tuple(images_batch.shape[1:]))
da_model.to(device)

# --- 3. 定义损失函数和优化器 ---
classification_criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(da_model.parameters(), lr=LEARNING_RATE)

# --- 4. 编写域适应训练循环 ---
print("\n--- 开始域适应训练 ---")

# 将目标域数据加载器变成一个可以无限循环的迭代器
target_iter = iter(target_loader)

for epoch in range(NUM_EPOCHS):
    da_model.train()
    total_loss, cls_loss_sum, coral_loss_sum = 0.0, 0.0, 0.0

    progress_bar = tqdm(source_loader, desc=f"Epoch {epoch+1}/{NUM_EPOCHS}")

    for source_images, source_labels in progress_bar:
        # a. 从目标域加载器中获取一批数据
        try:
            target_images = next(target_iter)
        except StopIteration:
            # 如果目标域数据用完了，就重新创建一个迭代器
            target_iter = iter(target_loader)
            target_images = next(target_iter)

        # b. 将数据移动到设备
        source_images, source_labels = source_images.to(device), source_labels.to(device)

        # 计算损失
        loss = torch.mean(torch.mul((source_cov - target_cov), (source_cov - target_cov)))

```

```

target_images = target_images.to(device)

# c. 清零梯度
optimizer.zero_grad()

# d. 分别通过特征提取器
source_features = da_model.features(source_images)
target_features = da_model.features(target_images)

# e. 源域数据通过分类器得到预测
source_preds = da_model.classifier(source_features.view(source_features.size(0), -1))

# f. 计算分类损失 (只在源域上)
classification_loss = classification_criterion(source_preds, source_labels)

# g. 计算 CORAL 损失 (在源域和目标域的特征上)
# 确保批次大小一致
min_batch_size = min(source_features.size(0), target_features.size(0))
c_loss = coral_loss(
    source_features.view(source_features.size(0), -1)[:min_batch_size],
    target_features.view(target_features.size(0), -1)[:min_batch_size]
)

# h. 计算总损失
total_loss_batch = classification_loss + LAMBDA_CORAL * c_loss

# i. 反向传播和优化
total_loss_batch.backward()
optimizer.step()

# 统计
total_loss += total_loss_batch.item()
cls_loss_sum += classification_loss.item()
coral_loss_sum += c_loss.item()
progress_bar.set_postfix({'cls_loss': f'{classification_loss.item():.4f}', 'coral_loss': f'{c_loss.item():.4f}'})

# 打印周期统计信息
avg_total_loss = total_loss / len(source_loader)
avg_cls_loss = cls_loss_sum / len(source_loader)
avg_coral_loss = coral_loss_sum / len(source_loader)
print(f"Epoch {epoch+1}/{NUM_EPOCHS} 完成，总损失：{avg_total_loss:.4f} (分类：{avg_cls_loss:.4f}, CORAL: {avg_coral_loss:.4f})")

```

```
print("\n--- 域适应训练完成 ---")

# 保存最终的域适应模型
DA_MODEL_SAVE_PATH = "./domain_adapted_model.pth"
torch.save(da_model.state_dict(), DA_MODEL_SAVE_PATH)
print(f'域适应模型已保存至: {DA_MODEL_SAVE_PATH}')
```