交流变压器振动数据 无监督+半监督 故障诊断方案

一、问题分析与现状总结

1.1 你现有的资产

组件	描述	作用
Zerone代码	1200维特征提取 + 图像编码 + ResNet分类	有监督分类，已验证效果好
Hetero-VAE代码	ResNet18-VAE，CWT+STFT+Context三通道	无监督重构，但依赖"正常"标签筛选
数据	~30,000条无标签 + 数千条有标签	标签稀缺是核心瓶颈

1.2 核心矛盾

1. **Zerone方法**：需要有标签数据训练，但标签稀缺
2. **Hetero-VAE方法**：号称无监督，但 `only_normal=True` 实际上依赖了"正常"目录标签
3. **真正的无监督场景**：3万条数据完全无标签，不知道哪些是正常/异常

1.3 你的数据结构（从上传文件分析）

```json
```

```
{
  "data_time": "2023-12-04T15:09:34.000Z", // 时间戳
  "signal_value": "0.12662,0.10868,...",   // 8192点振动信号
  "sensor_id": "xxx"                       // 传感器ID
}
```

二、推荐方案架构：三阶段渐进式学习

三、阶段一：真正的无监督异常检测

3.1 为什么不能直接用你现在的VAE?

你的 `hetero_train.py` 有一个隐藏假设：

```python
# 问题代码
train_set = TransformerVibrationDataset(train_path, only_normal=False, mode="train")
```

虽然 `only_normal=False`，但数据目录结构本身就带有标签信息（如 `/正常/`、`/故障/`）。

真正无监督的定义：模型完全不知道任何样本的标签，只能依靠数据本身的统计规律。

3.2 推荐模型：Deep SVDD + CWT特征

为什么选择Deep SVDD而不是VAE?

方法	优点	缺点
VAE重构误差	直观、可解释	假设"大多数是正常"才有效，重构能力过强时异常也能重构
Deep SVDD	直接学习"紧凑球"，天然适合异常检测	需要预训练初始化，否则容易坍塌
对比学习	特征表达能力强	需要数据增强设计合理

推荐组合：

1. **主模型**：Deep SVDD（单类分类器，假设大多数是正常）
2. **特征提取**：CWT时频图 + Zerone统计特征（复用你现有代码）
3. **辅助模型**：VAE用于生成重构误差作为辅助得分

3.3 Deep SVDD 核心代码

python

```python
# deep_svdd.py

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models

class DeepSVDD(nn.Module):
    """
    Deep Support Vector Data Description
    核心思想：将所有"正常"样本映射到隐空间的一个紧凑超球内
    """
    def __init__(self, encoder_backbone='resnet18', latent_dim=128):
        super().__init__()

        # 编码器：复用你的ResNet结构
        resnet = models.resnet18(weights=None)
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 64, 7, stride=2, padding=3, bias=False),
            resnet.bn1, resnet.relu, resnet.maxpool,
            resnet.layer1, resnet.layer2, resnet.layer3, resnet.layer4,
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten()
        )

        # 投影头
        self.projection = nn.Sequential(
            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Linear(256, latent_dim)
        )
```

```
    # 超球中心（训练后固定）
    self.register_buffer('center', torch.zeros(latent_dim))
    self.R = 0.0  # 超球半径

def forward(self, x):
    h = self.encoder(x)
    z = self.projection(h)
    return z

def init_center(self, dataloader, device, eps=0.1):
    """用训练数据的平均表示初始化中心"""
    n_samples = 0
    c = torch.zeros(self.projection[-1].out_features, device=device)

    self.eval()
    with torch.no_grad():
        for x in dataloader:
            x = x.to(device)
            z = self.forward(x)
            c += z.sum(dim=0)
            n_samples += z.size(0)
    c /= n_samples

    # 避免中心在原点（会导致trivial solution）
    c[(abs(c) < eps) & (c < 0)] = -eps
    c[(abs(c) < eps) & (c > 0)] = eps

    self.center = c

def anomaly_score(self, x):
    """计算异常得分：到中心的距离"""
```

```
        z = self.forward(x)
        dist = torch.sum((z - self.center) ** 2, dim=1)
        return dist


def svdd_loss(z, center, nu=0.1, R=None):
    """
    Deep SVDD 损失函数
    soft-boundary: 允许一定比例(nu)的样本在球外
    """
    dist = torch.sum((z - center) ** 2, dim=1)

    if R is None:  # One-class模式
        return torch.mean(dist)
    else:  # Soft-boundary模式
        scores = dist - R ** 2
        loss = R ** 2 + (1 / nu) * torch.mean(F.relu(scores))
        return loss
```

3.4 训练流程（阶段一完整代码）

python

```python
# stage1_unsupervised.py

import os
import json
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader
from pathlib import Path
import pywt
import cv2

# ============ 配置 ============
class Stage1Config:
    # 数据
    DATA_ROOT = "/path/to/unlabeled_data"
    SIGNAL_LEN = 8192
    FS = 10000  # 采样率，需根据实际调整

    # 模型
    INPUT_SIZE = 224
    LATENT_DIM = 128

    # 训练
    BATCH_SIZE = 32
    EPOCHS = 100
    LR = 1e-4
    NU = 0.05  # 假设5%的数据可能是异常

    # 输出
    OUTPUT_DIR = Path("./outputs/stage1")
    OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
```

```python
# ============ 数据集（无标签） ============
class UnlabeledVibrationDataset(Dataset):
    """
    无监督数据集：不关心目录结构，只读取所有信号
    """
    def __init__(self, root_dir, cfg=Stage1Config):
        self.cfg = cfg
        self.samples = []  # [(file_path, time_key), ...]

        root = Path(root_dir)
        for fp in root.rglob("*.jsonl"):
            self._parse_file(fp)
        for fp in root.rglob("*.json"):
            self._parse_file(fp)

        print(f"[Stage1] 加载 {len(self.samples)} 个样本")

    def _parse_file(self, fp):
        try:
            text = fp.read_text(encoding='utf-8', errors='ignore')
            if fp.suffix == '.jsonl':
                for line in text.splitlines():
                    if line.strip():
                        rec = json.loads(line)
                        self.samples.append((fp, rec))
            else:
                data = json.loads(text)
                if isinstance(data, list):
                    for rec in data:
                        self.samples.append((fp, rec))
        except Exception as e:
```

```
        pass

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        fp, rec = self.samples[idx]

        # 解析信号
        sig_raw = rec.get('signal_value', '')
        if isinstance(sig_raw, str):
            parts = sig_raw.replace('[','').replace(']','').split(',')
            sig = np.array([float(p) for p in parts if p.strip()], dtype=np.float32)
        elif isinstance(sig_raw, list):
            sig = np.array(sig_raw, dtype=np.float32)
        else:
            sig = np.zeros(self.cfg.SIGNAL_LEN, dtype=np.float32)

        # 长度对齐
        if len(sig) < self.cfg.SIGNAL_LEN:
            sig = np.pad(sig, (0, self.cfg.SIGNAL_LEN - len(sig)))
        else:
            sig = sig[:self.cfg.SIGNAL_LEN]

        # Z-score
        sig = (sig - sig.mean()) / (sig.std() + 1e-8)

        # 转CWT图像
        img = self._to_cwt_image(sig)

        return torch.from_numpy(img), idx
```

```
    def _to_cwt_image(self, sig):
        """信号 -> 3通道CWT图像"""
        H = W = self.cfg.INPUT_SIZE

        # CWT
        scales = np.arange(1, 129)
        cwt_matrix, _ = pywt.cwt(sig, scales, 'morl',
                                 sampling_period=1.0/self.cfg.FS)
        cwt_abs = np.abs(cwt_matrix).astype(np.float32)

        # Resize & Normalize
        cwt_img = cv2.resize(cwt_abs, (W, H))
        cwt_img = (cwt_img - cwt_img.min()) / (cwt_img.max() - cwt_img.min() + 1e-8)

        # 复制到3通道（或可以加入其他特征通道）
        img = np.stack([cwt_img, cwt_img, cwt_img], axis=0)
        return img.astype(np.float32)


# ============ 训练 ============
def train_stage1():
    cfg = Stage1Config()
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    # 数据
    dataset = UnlabeledVibrationDataset(cfg.DATA_ROOT, cfg)
    loader = DataLoader(dataset, batch_size=cfg.BATCH_SIZE, shuffle=True,
                        num_workers=4, drop_last=True)

    # 模型
    model = DeepSVDD(latent_dim=cfg.LATENT_DIM).to(device)
```

```python
# 预训练初始化（用AutoEncoder预训练避免坍塌）
print("预训练编码器...")
pretrain_autoencoder(model.encoder, model.projection, loader, device, epochs=10)

# 初始化中心
print("初始化超球中心...")
model.init_center(loader, device)

# 正式训练
optimizer = torch.optim.Adam(model.parameters(), lr=cfg.LR, weight_decay=1e-5)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, cfg.EPOCHS)

best_loss = float('inf')
for epoch in range(cfg.EPOCHS):
    model.train()
    total_loss = 0

    for x, _ in loader:
        x = x.to(device)
        z = model(x)
        loss = svdd_loss(z, model.center, nu=cfg.NU)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / len(loader)
    scheduler.step()

    if avg_loss < best_loss:
```

```
            best_loss = avg_loss
            torch.save({
                'model': model.state_dict(),
                'center': model.center,
                'epoch': epoch
            }, cfg.OUTPUT_DIR / 'best_model.pth')

        if (epoch + 1) % 10 == 0:
            print(f"Epoch {epoch+1}/{cfg.EPOCHS} | Loss: {avg_loss:.6f}")

    print(f"阶段一完成，最佳Loss: {best_loss:.6f}")
    return model


def pretrain_autoencoder(encoder, projection, loader, device, epochs=10):
    """用自编码器预训练，避免SVDD坍塌到trivial solution"""
    # 简单的解码器
    decoder = nn.Sequential(
        nn.Linear(128, 256),
        nn.ReLU(),
        nn.Linear(256, 512),
        nn.ReLU(),
        nn.Linear(512, 224*224*3)
    ).to(device)

    params = list(encoder.parameters()) + list(projection.parameters()) + list(decoder.parameters())
    opt = torch.optim.Adam(params, lr=1e-3)

    for ep in range(epochs):
        for x, _ in loader:
            x = x.to(device)
            B = x.size(0)
```

```
        h = encoder(x)
        z = projection(h)
        recon = decoder(z).view(B, 3, 224, 224)

        loss = F.mse_loss(recon, x)
        opt.zero_grad()
        loss.backward()
        opt.step()
```

四、阶段二：伪标签生成

4.1 异常得分计算与阈值确定

```
python
```

```python
# stage2_pseudo_label.py

import numpy as np
import torch
from scipy import stats

def compute_anomaly_scores(model, dataloader, device):
    """
    对所有样本计算异常得分
    """
    model.eval()
    scores = []
    indices = []

    with torch.no_grad():
        for x, idx in dataloader:
            x = x.to(device)
            score = model.anomaly_score(x)
            scores.extend(score.cpu().numpy().tolist())
            indices.extend(idx.numpy().tolist())

    return np.array(scores), np.array(indices)


def determine_thresholds(scores, method='percentile'):
    """
    确定正常/异常阈值

    方法选择：
    1. percentile: 简单分位数（假设异常比例）
    2. mad: 中位数绝对偏差（更鲁棒）
    3. pot: Peaks Over Threshold（极值理论）
```

```
    """
    if method == 'percentile':
        # 假设 95% 是正常，5% 可能异常
        t_normal = np.percentile(scores, 5)     # 低于此 = 高置信正常
        t_anomaly = np.percentile(scores, 99)   # 高于此 = 高置信异常

    elif method == 'mad':
        # 基于MAD的鲁棒方法
        median = np.median(scores)
        mad = np.median(np.abs(scores - median))

        t_normal = median - 2 * mad * 1.4826  # ~5%分位
        t_anomaly = median + 4 * mad * 1.4826  # ~99.9%分位

    elif method == 'pot':
        # 极值理论 Peaks Over Threshold
        # 适用于异常是极端值的场景
        from scipy.stats import genpareto

        q90 = np.percentile(scores, 90)
        exceedances = scores[scores > q90] - q90

        if len(exceedances) > 10:
            shape, _, scale = genpareto.fit(exceedances)
            # 设定误报率 q = 0.001
            t_anomaly = q90 + scale / shape * ((0.001 * len(scores) / len(exceedances)) ** (-shape) - 1)
        else:
            t_anomaly = np.percentile(scores, 99)

        t_normal = np.percentile(scores, 5)

    return t_normal, t_anomaly
```

```python
def generate_pseudo_labels(scores, indices, t_normal, t_anomaly):
    """
    生成伪标签

    返回:
    - pseudo_normal: 高置信正常样本索引
    - pseudo_anomaly: 高置信异常样本索引
    - uncertain: 不确定样本索引
    """
    pseudo_normal = indices[scores <= t_normal]
    pseudo_anomaly = indices[scores >= t_anomaly]
    uncertain = indices[(scores > t_normal) & (scores < t_anomaly)]

    print(f"伪标签统计:")
    print(f"  高置信正常: {len(pseudo_normal)} ({100*len(pseudo_normal)/len(scores):.1f}%)")
    print(f"  高置信异常: {len(pseudo_anomaly)} ({100*len(pseudo_anomaly)/len(scores):.1f}%)")
    print(f"  不确定区域: {len(uncertain)} ({100*len(uncertain)/len(scores):.1f}%)")

    return pseudo_normal, pseudo_anomaly, uncertain


# ============ 执行阶段二 ============
def run_stage2(model, dataset, device, output_dir):
    """
    执行伪标签生成流程
    """
    loader = DataLoader(dataset, batch_size=64, shuffle=False, num_workers=4)

    # 计算得分
    scores, indices = compute_anomaly_scores(model, loader, device)
```

```python
# 确定阈值（推荐MAD方法，对异构数据更鲁棒）
t_normal, t_anomaly = determine_thresholds(scores, method='mad')
print(f"阈值: 正常 < {t_normal:.4f}, 异常 > {t_anomaly:.4f}")

# 生成伪标签
pseudo_normal, pseudo_anomaly, uncertain = generate_pseudo_labels(
    scores, indices, t_normal, t_anomaly
)

# 保存结果
np.savez(output_dir / 'pseudo_labels.npz',
         scores=scores,
         indices=indices,
         pseudo_normal=pseudo_normal,
         pseudo_anomaly=pseudo_anomaly,
         uncertain=uncertain,
         t_normal=t_normal,
         t_anomaly=t_anomaly)

# 可视化得分分布
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.hist(scores, bins=100, alpha=0.7, edgecolor='black')
plt.axvline(t_normal, color='green', linestyle='--', label=f'正常阈值 ({t_normal:.3f})')
plt.axvline(t_anomaly, color='red', linestyle='--', label=f'异常阈值 ({t_anomaly:.3f})')
plt.xlabel('异常得分')
plt.ylabel('样本数')
plt.title('异常得分分布 & 伪标签划分')
plt.legend()
plt.savefig(output_dir / 'score_distribution.png', dpi=150)
plt.close()
```

```
    return pseudo_normal, pseudo_anomaly, uncertain
```

五、阶段三：有监督微调

5.1 整合Zerone特征 + 迁移学习

```
python
```

```python
# stage3_supervised.py

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
import numpy as np

# 复用你的Zerone特征提取
from zerone_features import (
    compute_time_features, stft_segment_means, compute_psd,
    compute_high_frequency_ratios, normalize_features
)


class SupervisedVibrationDataset(Dataset):
    """
    有监督数据集：结合真实标签 + 伪标签
    """
    def __init__(self,
                 labeled_samples,     # [(signal, label), ...]
                 pseudo_normal_idx,    # 伪标签正常的索引
                 pseudo_anomaly_idx,   # 伪标签异常的索引
                 unlabeled_dataset,    # 阶段一的无标签数据集
                 cfg):
        self.cfg = cfg
        self.samples = []

        # 1. 真实标签数据
        for sig, label in labeled_samples:
            self.samples.append((sig, label, 'real'))

        # 2. 伪标签数据（可选权重降低）
```

```
        for idx in pseudo_normal_idx:
            sig, _ = unlabeled_dataset[idx]
            self.samples.append((sig.numpy(), 0, 'pseudo'))  # 0 = 正常

        for idx in pseudo_anomaly_idx:
            sig, _ = unlabeled_dataset[idx]
            self.samples.append((sig.numpy(), 1, 'pseudo'))  # 1 = 异常

        print(f"[Stage3] 真实标签: {len(labeled_samples)}, "
              f"伪标签: {len(pseudo_normal_idx) + len(pseudo_anomaly_idx)}")

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        sig, label, source = self.samples[idx]

        # 如果sig已经是图像格式，直接返回
        if isinstance(sig, np.ndarray) and sig.ndim == 3:
            img = torch.from_numpy(sig)
        else:
            # 需要转换
            img = self._signal_to_image(sig)

        # 权重：伪标签样本权重可以降低
        weight = 1.0 if source == 'real' else 0.5

        return img, label, weight

    def _signal_to_image(self, sig):
        """结合CWT + Zerone特征"""
        # 复用hetero_data.py中的_to_rgb3_from_1d逻辑
```

```
    # 或者直接用Zerone的vector_to_image
    pass  # 根据实际实现


class TransformerClassifier(nn.Module):
    """
    迁移学习分类器：使用Stage1预训练的编码器
    """
    def __init__(self, pretrained_encoder, num_classes=2, freeze_encoder=True):
        super().__init__()

        self.encoder = pretrained_encoder

        if freeze_encoder:
            for param in self.encoder.parameters():
                param.requires_grad = False

        # 分类头
        self.classifier = nn.Sequential(
            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 64),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(64, num_classes)
        )

    def forward(self, x):
        # 编码
        feat = self.encoder(x)  # (B, 512)
```

```
        # 分类
        logits = self.classifier(feat)
        return logits

    def unfreeze_encoder(self, unfreeze_layers=2):
        """渐进解冻：先解冻后几层"""
        # ResNet的层：layer1, layer2, layer3, layer4
        layers = ['layer4', 'layer3', 'layer2', 'layer1']

        for i, name in enumerate(layers[:unfreeze_layers]):
            for child_name, child in self.encoder.named_children():
                if name in str(child_name):
                    for param in child.parameters():
                        param.requires_grad = True


def train_stage3(pretrained_model, labeled_data, pseudo_labels, unlabeled_dataset, cfg):
    """
    阶段三训练流程
    """
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    # 数据
    pseudo_normal, pseudo_anomaly, _ = pseudo_labels
    dataset = SupervisedVibrationDataset(
        labeled_data, pseudo_normal, pseudo_anomaly, unlabeled_dataset, cfg
    )

    # 类别平衡采样
    labels = [s[1] for s in dataset.samples]
    class_counts = np.bincount(labels)
```

```
weights = 1.0 / class_counts[labels]
sampler = WeightedRandomSampler(weights, len(weights))

loader = DataLoader(dataset, batch_size=32, sampler=sampler, num_workers=4)

# 模型：迁移Stage1的编码器
classifier = TransformerClassifier(
    pretrained_model.encoder,
    num_classes=2,
    freeze_encoder=True
).to(device)

# 优化器
optimizer = torch.optim.AdamW(
    filter(lambda p: p.requires_grad, classifier.parameters()),
    lr=1e-3, weight_decay=1e-4
)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='max', patience=5, factor=0.5
)

criterion = nn.CrossEntropyLoss(reduction='none')  # 支持样本权重

# 训练
best_f1 = 0
for epoch in range(50):
    classifier.train()
    total_loss = 0

    for img, label, weight in loader:
        img = img.to(device)
        label = label.to(device)
```

```
            weight = weight.to(device)

            logits = classifier(img)
            loss = (criterion(logits, label) * weight).mean()

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

        # 验证（需要真实标签验证集）
        val_f1 = validate(classifier, val_loader, device)
        scheduler.step(val_f1)

        if val_f1 > best_f1:
            best_f1 = val_f1
            torch.save(classifier.state_dict(), cfg.OUTPUT_DIR / 'best_classifier.pth')

        # 渐进解冻
        if epoch == 20:
            classifier.unfreeze_encoder(unfreeze_layers=1)
            print("解冻 layer4")
        if epoch == 30:
            classifier.unfreeze_encoder(unfreeze_layers=2)
            print("解冻 layer3, layer4")

        print(f"Epoch {epoch+1} | Loss: {total_loss/len(loader):.4f} | Val F1: {val_f1:.4f}")

    return classifier
```

六、推荐的最佳实践

6.1 关于模型选择

场景	推荐方案
完全无标签，需要快速筛选	Deep SVDD + CWT
有少量正常样本可用	VAE（你现有的hetero代码）
有少量有标签数据	迁移学习 + Zerone特征
需要可解释性	VAE重构误差 + Grad-CAM

6.2 关于你现有代码的改进建议

hetero_data.py 改进：

```python
# 原来的问题：依赖目录结构判断标签
# 改进：完全不依赖目录名
class TrueUnlabeledDataset(Dataset):
    def __init__(self, root_dir):
        self.samples = []
        # 不管目录叫什么名字，全部读取
        for fp in Path(root_dir).rglob("*.jsonl"):
            self._parse(fp)
```

hetero_model.py 改进：

python

```python
# 增加异常得分计算
class SpatialResNetVAE(nn.Module):
    def anomaly_score(self, x):
        """基于重构误差 + KL散度的异常得分"""
        recon, mu, logvar = self.forward(x)

        # L1重构误差
        recon_loss = F.l1_loss(recon, x, reduction='none').mean(dim=[1,2,3])

        # KL散度
        kl_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp(), dim=[1,2,3])

        # 综合得分
        score = recon_loss + 0.1 * kl_loss
        return score
```

6.3 完整工作流

bash

```
# 1. 阶段一：无监督预训练
python stage1_unsupervised.py --data /path/to/30000_unlabeled

# 2. 阶段二：伪标签生成
python stage2_pseudo_label.py --model outputs/stage1/best_model.pth

# 3. 阶段三：有监督微调（结合少量真实标签）
python stage3_supervised.py \
  --pretrained outputs/stage1/best_model.pth \
  --pseudo outputs/stage2/pseudo_labels.npz \
  --labeled /path/to/labeled_data

# 4. 推理
python inference.py --model outputs/stage3/best_classifier.pth --input new_data.jsonl
```

七、与你现有代码的对接

7.1 特征提取复用

你的 `zerone_features.py` 已经非常完善，可以直接复用：

```
python
```

```python
from zerone_features import (
    compute_time_features,     # 15维时域特征
    stft_segment_means,        # 127维STFT
    compute_psd,               # 1050维PSD
    compute_high_frequency_ratios,  # 8维高频指标
    build_sample_from_multichannel  # 多通道聚合
)

def extract_zerone_features(signal, fs):
    """
    提取1200维Zerone特征
    """
    # 时域
    time_feat = compute_time_features(signal)  # 15维

    # 去直流
    x_dc = signal - np.mean(signal)

    # STFT
    stft_feat = stft_segment_means(x_dc, fs)  # 127维

    # PSD
    psd = compute_psd(x_dc, fs, fmin=1, fmax=2000)  # 2000维 -> 压缩到1050

    # 高频
    hf_feat = []
    for thr in [1000, 2000, 3000, 4000]:
        amp_r, pwr_r = compute_high_frequency_ratios(psd, fs, threshold_hz=thr)
        hf_feat.extend([amp_r, pwr_r])

    return np.concatenate([time_feat, stft_feat, psd[:1050], hf_feat])
```

7.2 VAE重构复用

你的 `hetero_model.py` 中的VAE可以作为辅助模型：

```python
# 融合SVDD + VAE的异常得分
def ensemble_anomaly_score(svdd_model, vae_model, x):
    """
    双模型集成异常检测
    """
    # SVDD得分
    svdd_score = svdd_model.anomaly_score(x)

    # VAE重构得分
    recon, mu, logvar = vae_model(x)
    vae_score = F.l1_loss(recon, x, reduction='none').mean(dim=[1,2,3])

    # 归一化后加权
    svdd_norm = (svdd_score - svdd_score.mean()) / (svdd_score.std() + 1e-8)
    vae_norm = (vae_score - vae_score.mean()) / (vae_score.std() + 1e-8)

    # 加权融合
    final_score = 0.6 * svdd_norm + 0.4 * vae_norm
    return final_score
```

八、总结

阶段	输入	方法	输出
一	30,000无标签	Deep SVDD/VAE	异常得分分布
二	异常得分	阈值筛选	伪标签(正常/异常/不确定)
三	伪标签+真实标签	迁移学习分类器	故障诊断模型

核心改进点：

1. 不再依赖目录结构作为隐式标签
2. 用Deep SVDD代替纯VAE，更适合异常检测
3. 渐进式伪标签扩充，充分利用无标签数据
4. 复用你现有的Zerone特征工程

下一步建议：

1. 先在你现有的有标签数据上验证Stage3的分类效果
2. 然后逐步在无标签数据上应用Stage1+Stage2
3. 迭代优化伪标签质量和阈值设定