# 改进yolo11-CA-HSFPN等200+全套创新点大全：护目镜佩戴与否检测系统源码＆数据集全套

## 1. 图片效果展示

注意：由于项目一直在更新迭代，上面“1.图片效果展示”和“2.视频效果展示”展示的系统图片或者视频可能为老版本，新版本在老版本的基础上升级如下：（实际效果以升级的新版本为准）  
  
 （1）适配了YOLOV11的“目标检测”模型和“实例分割”模型，通过加载相应的权重（.pt）文件即可自适应加载模型。  
  
 （2）支持“图片识别”、“视频识别”、“摄像头实时识别”三种识别模式。  
  
 （3）支持“图片识别”、“视频识别”、“摄像头实时识别”三种识别结果保存导出，解决手动导出（容易卡顿出现爆内存）存在的问题，识别完自动保存结果并导出到tempDir中。  
  
 （4）支持Web前端系统中的标题、背景图等自定义修改。  
  
 另外本项目提供训练的数据集和训练教程,暂不提供权重文件（best.pt）,需要您按照教程进行训练后实现图片演示和Web前端界面演示的效果。

## 2.视频效果展示

## 3.背景

研究背景与意义  
  
随着工业化进程的加快和科技的不断发展，安全生产问题日益受到重视。特别是在一些高风险行业，如建筑、制造和化学等领域，佩戴护目镜等个人防护装备（PPE）已成为保障工人安全的重要措施。然而，尽管相关法规和标准日益完善，护目镜的佩戴率仍然不尽如人意，导致工伤事故频发。因此，开发一种高效、智能的护目镜佩戴检测系统显得尤为重要。  
  
本研究旨在基于改进的YOLOv11模型，构建一个高效的护目镜佩戴与否检测系统。YOLO（You Only Look Once）系列模型因其高效的实时目标检测能力而广泛应用于各类视觉识别任务。通过对YOLOv11进行改进，我们期望在保持其高速度的同时，提升对护目镜佩戴状态的检测精度。为此，我们将利用一个包含6300张图像的数据集，该数据集分为三类：佩戴护目镜（Goggles）、未佩戴护目镜（NO-Goggles）以及其他类别。这一数据集的构建为模型的训练和验证提供了丰富的样本，有助于提高模型的泛化能力。  
  
通过实现护目镜佩戴状态的自动检测，不仅可以有效降低人工监测的成本和误差，还能实时监控工人的安全状态，及时发出警报，防止潜在的安全隐患。此外，该系统的应用还可以为企业提供数据支持，帮助其制定更为科学的安全管理措施，从而进一步提升整体安全生产水平。综上所述，本研究不仅具有重要的理论意义，也具备广泛的实际应用价值。

## 4. 数据集信息展示

本项目数据集信息介绍  
  
本项目所使用的数据集名为“Eye Protection”，旨在为改进YOLOv11的护目镜佩戴与否检测系统提供强有力的支持。该数据集包含两类主要目标，分别为“Goggles”和“NO-Goggles”，共计两个类别。通过对这两个类别的精确标注和分类，数据集为模型的训练和评估提供了丰富的样本，使其能够有效识别佩戴护目镜的人员与未佩戴护目镜的人员。  
  
“Eye Protection”数据集的构建过程中，特别注重样本的多样性和代表性，确保数据集能够涵盖不同环境、光照条件以及佩戴者的多样性。这种多样性不仅有助于提高模型的泛化能力，还能增强其在实际应用中的适应性。数据集中包含的图像均经过精心挑选，确保每个类别的样本数量相对均衡，以避免模型在训练过程中出现偏倚。  
  
此外，数据集还包括详细的标注信息，标注不仅涵盖了目标的类别，还包括目标在图像中的位置。这些信息对于YOLOv11模型的训练至关重要，能够帮助模型学习到不同类别目标的特征和空间分布。通过使用“Eye Protection”数据集，研究团队希望能够提升护目镜佩戴检测系统的准确性和效率，从而在实际应用中更好地保障人员的安全。  
  
综上所述，“Eye Protection”数据集为本项目提供了坚实的数据基础，通过对护目镜佩戴与否的精准检测，旨在推动安全防护领域的技术进步，确保在各种工作环境中，工作人员能够得到有效的保护。

## 5. 全套项目环境部署教程（零基础手把手教学）

5.1 所需软件PyCharm和Anaconda安装教程（第一步）

https://www.bilibili.com/video/BV1BoC1YCEKi/?spm\_id\_from=333.999.0.0&vd\_source=bc9aec86d164b67a7004b996143742dc

5.2 安装Python虚拟环境创建和依赖库安装视频教程（第二步）

https://www.bilibili.com/video/BV1ZoC1YCEBw?spm\_id\_from=333.788.videopod.sections&vd\_source=bc9aec86d164b67a7004b996143742dc

## 6. 改进YOLOv11训练教程和Web\_UI前端加载模型视频教程（零基础手把手教学）

https://www.bilibili.com/video/BV1BoC1YCEhR?spm\_id\_from=333.788.videopod.sections&vd\_source=bc9aec86d164b67a7004b996143742dc

## 7.原始YOLOv11算法讲解

## 8. 70+全套YOLOV8创新点原理讲解（非科班也可以轻松写刊发刊，V12版本正在科研待更新）

由于篇幅限制，每个创新点的具体原理讲解就不全部展开，具体见下列网址中的改进模块对应项目的技术原理博客网址【Blog】（创新点均为模块化搭建，原理适配YOLOv5~YOLOv11等各种版本）

https://gitee.com/qunmasj/good

## 9.系统功能展示（检测对象为举例，实际内容以本项目数据集为准）

图9.1.系统支持检测结果表格显示  
  
 图9.2.系统支持置信度和IOU阈值手动调节  
  
 图9.3.系统支持自定义加载权重文件best.pt(需要你通过步骤5中训练获得)  
  
 图9.4.系统支持摄像头实时识别  
  
 图9.5.系统支持图片识别  
  
 图9.6.系统支持视频识别  
  
 图9.7.系统支持识别结果文件自动保存  
  
 图9.8.系统支持Excel导出检测结果数据

## 10. YOLOv11核心改进模块源码讲解

```以下是对代码的核心部分进行提炼和详细注释的版本：  
  
```python  
import sys  
import subprocess  
  
def run\_script(script\_path):  
 """  
 使用当前 Python 环境运行指定的脚本。  
  
 Args:  
 script\_path (str): 要运行的脚本路径  
  
 Returns:  
 None  
 """  
 # 获取当前 Python 解释器的路径  
 python\_path = sys.executable  
  
 # 构建运行命令，使用 streamlit 运行指定的脚本  
 command = f'"{python\_path}" -m streamlit run "{script\_path}"'  
  
 # 执行命令并等待其完成  
 result = subprocess.run(command, shell=True)  
   
 # 检查命令执行结果，如果返回码不为0，表示执行出错  
 if result.returncode != 0:  
 print("脚本运行出错。")  
  
# 程序入口  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 指定要运行的脚本路径  
 script\_path = "web.py" # 这里可以直接指定脚本名称，假设它在当前目录下  
  
 # 调用函数运行脚本  
 run\_script(script\_path)  
```  
  
### 代码核心部分分析：  
1. \*\*导入模块\*\*：  
 - `sys`：用于获取当前 Python 解释器的路径。  
 - `subprocess`：用于执行外部命令。  
  
2. \*\*`run\_script` 函数\*\*：  
 - 该函数接收一个脚本路径作为参数，并在当前 Python 环境中运行该脚本。  
 - 使用 `sys.executable` 获取当前 Python 解释器的路径，以确保脚本在正确的环境中运行。  
 - 构建一个命令字符串，使用 `streamlit` 模块运行指定的脚本。  
 - 使用 `subprocess.run` 执行命令，并通过 `shell=True` 允许在 shell 中执行命令。  
 - 检查命令的返回码，如果不为0，表示脚本运行出错，打印错误信息。  
  
3. \*\*程序入口\*\*：  
 - 使用 `if \_\_name\_\_ == "\_\_main\_\_":` 确保只有在直接运行该脚本时才会执行以下代码。  
 - 指定要运行的脚本路径，并调用 `run\_script` 函数来执行该脚本。```

这个程序文件 `ui.py` 的主要功能是通过当前的 Python 环境来运行一个指定的脚本，具体来说是一个名为 `web.py` 的脚本。程序首先导入了必要的模块，包括 `sys`、`os` 和 `subprocess`，以及一个自定义的 `abs\_path` 函数，这个函数来自于 `QtFusion.path` 模块，可能用于获取文件的绝对路径。  
  
在 `run\_script` 函数中，程序接受一个参数 `script\_path`，这个参数是要运行的脚本的路径。函数内部首先获取当前 Python 解释器的路径，使用 `sys.executable` 来实现。接着，程序构建了一个命令字符串，这个命令使用了 `streamlit` 模块来运行指定的脚本。具体的命令格式是 `"{python\_path}" -m streamlit run "{script\_path}"`，其中 `{python\_path}` 和 `{script\_path}` 会被实际的路径替换。  
  
随后，程序使用 `subprocess.run` 方法来执行这个命令。`shell=True` 参数允许在 shell 中执行命令，这样可以处理更复杂的命令。如果脚本运行返回的状态码不为零，表示运行过程中出现了错误，程序会打印出 "脚本运行出错。" 的提示信息。  
  
在文件的最后部分，程序通过 `if \_\_name\_\_ == "\_\_main\_\_":` 语句来确保只有在直接运行这个文件时才会执行后面的代码。在这个部分，程序调用 `abs\_path` 函数来获取 `web.py` 的绝对路径，并将其存储在 `script\_path` 变量中。最后，调用 `run\_script` 函数来运行这个脚本。  
  
总的来说，这个程序的作用是提供一个简单的接口来运行一个 Streamlit 应用，具体是通过调用 `web.py` 脚本来实现的。

```以下是代码中最核心的部分，并附上详细的中文注释：  
  
```python  
import torch  
import torch.nn as nn  
from functools import partial  
  
# 导入自定义的归一化模块  
from .prepbn import RepBN, LinearNorm  
# 导入自定义的Transformer编码器层  
from ..modules.transformer import TransformerEncoderLayer  
  
# 定义一个包含AIFI\_RepBN的模块  
\_\_all\_\_ = ['AIFI\_RepBN']  
  
# 定义线性归一化和RepBN的组合  
ln = nn.LayerNorm  
linearnorm = partial(LinearNorm, norm1=ln, norm2=RepBN, step=60000)  
  
# 定义带有RepBN的Transformer编码器层  
class TransformerEncoderLayer\_RepBN(TransformerEncoderLayer):  
 def \_\_init\_\_(self, c1, cm=2048, num\_heads=8, dropout=0, act=..., normalize\_before=False):  
 # 初始化父类的构造函数  
 super().\_\_init\_\_(c1, cm, num\_heads, dropout, act, normalize\_before)  
   
 # 初始化两个归一化层  
 self.norm1 = linearnorm(c1)  
 self.norm2 = linearnorm(c1)  
  
# 定义AIFI transformer层，继承自TransformerEncoderLayer\_RepBN  
class AIFI\_RepBN(TransformerEncoderLayer\_RepBN):  
 """定义AIFI transformer层。"""  
  
 def \_\_init\_\_(self, c1, cm=2048, num\_heads=8, dropout=0, act=nn.GELU(), normalize\_before=False):  
 """使用指定参数初始化AIFI实例。"""  
 super().\_\_init\_\_(c1, cm, num\_heads, dropout, act, normalize\_before)  
  
 def forward(self, x):  
 """AIFI transformer层的前向传播。"""  
 c, h, w = x.shape[1:] # 获取输入的通道数、高度和宽度  
 pos\_embed = self.build\_2d\_sincos\_position\_embedding(w, h, c) # 构建位置嵌入  
 # 将输入从形状[B, C, H, W]展平为[B, HxW, C]  
 x = super().forward(x.flatten(2).permute(0, 2, 1), pos=pos\_embed.to(device=x.device, dtype=x.dtype))  
 # 将输出转换回原始形状并返回  
 return x.permute(0, 2, 1).view([-1, c, h, w]).contiguous()  
  
 @staticmethod  
 def build\_2d\_sincos\_position\_embedding(w, h, embed\_dim=256, temperature=10000.0):  
 """构建2D正弦-余弦位置嵌入。"""  
 assert embed\_dim % 4 == 0, "嵌入维度必须是4的倍数，以便进行2D正弦-余弦位置嵌入"  
 grid\_w = torch.arange(w, dtype=torch.float32) # 创建宽度的网格  
 grid\_h = torch.arange(h, dtype=torch.float32) # 创建高度的网格  
 grid\_w, grid\_h = torch.meshgrid(grid\_w, grid\_h, indexing="ij") # 生成网格坐标  
 pos\_dim = embed\_dim // 4 # 计算位置嵌入的维度  
 omega = torch.arange(pos\_dim, dtype=torch.float32) / pos\_dim # 计算频率  
 omega = 1.0 / (temperature\*\*omega) # 应用温度缩放  
  
 # 计算宽度和高度的正弦和余弦值  
 out\_w = grid\_w.flatten()[..., None] @ omega[None]  
 out\_h = grid\_h.flatten()[..., None] @ omega[None]  
  
 # 返回拼接后的正弦和余弦位置嵌入  
 return torch.cat([torch.sin(out\_w), torch.cos(out\_w), torch.sin(out\_h), torch.cos(out\_h)], 1)[None]  
```  
  
### 代码核心部分解释：  
1. \*\*导入必要的库\*\*：导入了PyTorch和自定义的模块。  
2. \*\*归一化层的定义\*\*：使用`partial`函数创建一个组合的归一化层，包含线性归一化和RepBN。  
3. \*\*Transformer编码器层的定义\*\*：创建一个新的编码器层类，继承自基础的Transformer编码器层，并在其中初始化两个归一化层。  
4. \*\*AIFI transformer层的实现\*\*：在AIFI\_RepBN类中实现了前向传播方法，包含位置嵌入的构建和输入形状的转换。  
5. \*\*位置嵌入的构建\*\*：静态方法`build\_2d\_sincos\_position\_embedding`用于生成2D正弦-余弦位置嵌入，确保嵌入维度为4的倍数，并使用温度缩放来调整频率。```

这个程序文件 `transformer.py` 实现了一个基于 Transformer 架构的编码器层，特别是结合了重参数化批归一化（RepBN）和线性归一化（LinearNorm）的特性。文件中主要定义了两个类：`TransformerEncoderLayer\_RepBN` 和 `AIFI\_RepBN`。  
  
首先，文件导入了必要的 PyTorch 库以及其他模块。`RepBN` 和 `LinearNorm` 是自定义的归一化模块，可能用于改进模型的训练效果和性能。`TransformerEncoderLayer` 是基础的 Transformer 编码器层，`AIFI` 可能是某种特定的 Transformer 变体。  
  
在 `TransformerEncoderLayer\_RepBN` 类中，构造函数接受多个参数，包括输入通道数 `c1`、中间层的通道数 `cm`、注意力头的数量 `num\_heads`、dropout 概率、激活函数和是否在归一化之前进行处理的标志。这个类调用了父类的构造函数，并初始化了两个归一化层 `norm1` 和 `norm2`，这两个层使用了之前定义的 `linearnorm` 函数。  
  
接下来，`AIFI\_RepBN` 类继承自 `TransformerEncoderLayer\_RepBN`，并定义了自己的构造函数。这个构造函数与父类相似，但默认激活函数设置为 GELU（高斯误差线性单元）。此外，它还实现了一个 `forward` 方法，负责前向传播。  
  
在 `forward` 方法中，输入张量 `x` 的形状被调整，以便与位置嵌入相结合。具体来说，输入的形状从 `[B, C, H, W]` 被展平为 `[B, HxW, C]`，然后通过调用父类的 `forward` 方法进行处理。位置嵌入是通过 `build\_2d\_sincos\_position\_embedding` 方法生成的，该方法使用二维正弦和余弦函数来创建位置编码，以增强模型对输入数据中空间信息的理解。  
  
`build\_2d\_sincos\_position\_embedding` 方法接收宽度 `w`、高度 `h` 和嵌入维度 `embed\_dim` 作为参数。它首先确保嵌入维度是4的倍数，以便能够正确地计算正弦和余弦值。然后，生成了宽度和高度的网格，并计算出对应的正弦和余弦位置编码，最后将这些编码合并并返回。  
  
总体来说，这个文件实现了一个结合了重参数化批归一化和位置编码的 Transformer 编码器层，旨在提高模型的性能和表达能力。

```以下是简化后的代码，保留了核心部分，并添加了详细的中文注释：  
  
```python  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
  
class GRNwithNHWC(nn.Module):  
 """   
 GRN (Global Response Normalization) 层  
 该层用于对输入进行全局响应归一化，输入假设为 (N, H, W, C) 格式。  
 """  
 def \_\_init\_\_(self, dim, use\_bias=True):  
 super().\_\_init\_\_()  
 self.use\_bias = use\_bias  
 self.gamma = nn.Parameter(torch.zeros(1, 1, 1, dim)) # 可学习参数  
 if self.use\_bias:  
 self.beta = nn.Parameter(torch.zeros(1, 1, 1, dim)) # 可学习偏置  
  
 def forward(self, x):  
 Gx = torch.norm(x, p=2, dim=(1, 2), keepdim=True) # 计算L2范数  
 Nx = Gx / (Gx.mean(dim=-1, keepdim=True) + 1e-6) # 归一化  
 if self.use\_bias:  
 return (self.gamma \* Nx + 1) \* x + self.beta # 应用归一化和偏置  
 else:  
 return (self.gamma \* Nx + 1) \* x # 仅应用归一化  
  
class UniRepLKNetBlock(nn.Module):  
 """  
 UniRepLKNet中的基本块  
 包含深度卷积、归一化、Squeeze-and-Excitation块和前馈网络。  
 """  
 def \_\_init\_\_(self, dim, kernel\_size, drop\_path=0., deploy=False):  
 super().\_\_init\_\_()  
 self.dwconv = nn.Conv2d(dim, dim, kernel\_size=kernel\_size, stride=1, padding=kernel\_size // 2, groups=dim) # 深度卷积  
 self.norm = nn.BatchNorm2d(dim) # 批归一化  
 self.se = SEBlock(dim, dim // 4) # Squeeze-and-Excitation块  
 self.pwconv1 = nn.Linear(dim, dim \* 4) # 前馈网络第一层  
 self.pwconv2 = nn.Linear(dim \* 4, dim) # 前馈网络第二层  
 self.drop\_path = nn.Identity() if drop\_path <= 0 else DropPath(drop\_path) # 随机深度  
  
 def forward(self, inputs):  
 x = self.dwconv(inputs) # 深度卷积  
 x = self.norm(x) # 归一化  
 x = self.se(x) # Squeeze-and-Excitation  
 x = F.gelu(self.pwconv1(x)) # 激活函数  
 x = self.pwconv2(x) # 前馈网络  
 return self.drop\_path(x) + inputs # 残差连接  
  
class UniRepLKNet(nn.Module):  
 """  
 UniRepLKNet模型的实现  
 包含多个UniRepLKNetBlock，构成深度网络。  
 """  
 def \_\_init\_\_(self, in\_chans=3, num\_classes=1000, depths=(3, 3, 27, 3), dims=(96, 192, 384, 768)):  
 super().\_\_init\_\_()  
 self.stages = nn.ModuleList() # 存储各个阶段的块  
 for i in range(len(depths)):  
 stage = nn.Sequential(  
 \*[UniRepLKNetBlock(dim=dims[i], kernel\_size=3) for \_ in range(depths[i])]  
 )  
 self.stages.append(stage)  
  
 def forward(self, x):  
 for stage in self.stages:  
 x = stage(x) # 通过每个阶段  
 return x  
  
# 示例代码：创建模型并进行前向传播  
if \_\_name\_\_ == '\_\_main\_\_':  
 inputs = torch.randn((1, 3, 640, 640)) # 随机输入  
 model = UniRepLKNet() # 创建模型  
 res = model(inputs) # 前向传播  
 print(res.shape) # 输出结果的形状  
```  
  
### 代码说明：  
1. \*\*GRNwithNHWC\*\*: 实现了全局响应归一化层，用于对输入特征进行归一化处理，增强模型的稳定性。  
2. \*\*UniRepLKNetBlock\*\*: 这是模型的基本构建块，包含深度卷积、归一化、Squeeze-and-Excitation模块和前馈网络，支持残差连接。  
3. \*\*UniRepLKNet\*\*: 整个模型的实现，包含多个阶段，每个阶段由多个基本块组成，最终输出特征。  
  
以上代码保留了模型的核心结构，并进行了简化和注释，便于理解其功能和结构。```

这个程序文件 `UniRepLKNet.py` 实现了一个名为 UniRepLKNet 的深度学习模型，主要用于音频、视频、点云、时间序列和图像识别。该模型基于多个已有的深度学习框架和模型，如 RepLKNet、ConvNeXt、DINO 和 DeiT。程序的结构主要由多个类和函数组成，以下是对其主要部分的说明。  
  
首先，程序导入了必要的库，包括 PyTorch 及其相关模块，以及一些用于深度学习模型构建的工具。接着，定义了一些核心组件，例如 GRN（全局响应归一化）层、NCHW 和 NHWC 格式转换的层、以及用于卷积操作的辅助函数。这些组件为后续的模型构建提供了基础。  
  
在 `GRNwithNHWC` 类中，实现了 GRN 层的功能，通过计算输入张量的 L2 范数并进行归一化，增强了模型的表达能力。`NCHWtoNHWC` 和 `NHWCtoNCHW` 类则负责在不同的张量格式之间进行转换，以适应不同的操作需求。  
  
接下来，`get\_conv2d` 函数根据输入参数决定使用哪种卷积实现，特别是针对大卷积核的情况，尝试使用 iGEMM 优化实现。`get\_bn` 函数则根据是否使用同步批归一化来返回相应的归一化层。  
  
`SEBlock` 类实现了 Squeeze-and-Excitation 模块，通过自适应平均池化和全连接层来增强特征通道的表示能力。`DilatedReparamBlock` 类则实现了膨胀卷积的重参数化块，支持多种卷积核大小和膨胀率的组合，提升了模型在不同尺度下的特征提取能力。  
  
`UniRepLKNetBlock` 类是 UniRepLKNet 的基本构建块，结合了深度卷积、归一化、Squeeze-and-Excitation 机制和前馈网络。该类支持在训练和推理模式之间切换，能够根据需要选择是否使用检查点来节省内存。  
  
`UniRepLKNet` 类是整个模型的核心，负责构建模型的不同阶段和下采样层。根据输入的参数，模型会初始化不同的卷积层和归一化层，并通过前向传播函数处理输入数据。该类还实现了权重初始化和模型的推理模式切换功能。  
  
最后，程序提供了一些函数，如 `unireplknet\_a`、`unireplknet\_f` 等，用于创建不同版本的 UniRepLKNet 模型，并可选择加载预训练权重。在主程序部分，创建了一个随机输入的张量，并实例化了模型进行前向传播，展示了模型的基本使用方式。  
  
整体来看，`UniRepLKNet.py` 文件实现了一个复杂而灵活的深度学习模型，能够处理多种输入类型，并在不同的任务中表现出色。通过模块化的设计，便于扩展和维护，同时也为研究人员提供了丰富的功能和灵活性。

### 程序整体功能和构架概括  
  
该程序是一个深度学习框架，主要用于实现和测试不同的模型架构，特别是针对选择性扫描和 Transformer 架构的模型。整体结构由多个模块组成，每个模块负责特定的功能，包括模型的构建、测试和用户界面等。以下是对各个文件功能的总结：  
  
- \*\*ui.py\*\*: 提供了一个用户界面功能，允许用户通过命令行运行一个 Streamlit 应用（`web.py`），用于可视化和交互。  
- \*\*test\_selective\_scan\_speed.py\*\*: 实现了选择性扫描算法的速度测试，比较不同实现的性能，帮助优化算法效率。  
- \*\*transformer.py\*\*: 实现了一个基于 Transformer 的编码器层，结合了重参数化批归一化和位置编码，增强了模型的表达能力。  
- \*\*UniRepLKNet.py\*\*: 构建了一个名为 UniRepLKNet 的深度学习模型，支持多种输入类型，并结合了多个先进的深度学习技术，旨在提高特征提取和表示能力。  
  
### 文件功能整理表  
  
| 文件名 | 功能描述 |  
|----------------------------------|-----------------------------------------------------------------------------------------------|  
| `ui.py` | 提供用户界面功能，通过命令行运行 Streamlit 应用（`web.py`），用于可视化和交互。 |  
| `test\_selective\_scan\_speed.py` | 实现选择性扫描算法的速度测试，比较不同实现的性能，帮助优化算法效率。 |  
| `transformer.py` | 实现基于 Transformer 的编码器层，结合重参数化批归一化和位置编码，增强模型的表达能力。 |  
| `UniRepLKNet.py` | 构建 UniRepLKNet 深度学习模型，支持多种输入类型，结合多个先进的深度学习技术，提高特征提取能力。 |  
  
这个结构使得程序具备良好的模块化特性，便于扩展和维护，同时也为研究和开发提供了灵活的工具和框架。

注意：由于此博客编辑较早，上面“10.YOLOv11核心改进源码讲解”中部分代码可能会优化升级，仅供参考学习，以“11.完整训练+Web前端界面+200+种全套创新点源码、数据集获取”的内容为准。