一、实验目的

1、 理解并掌握 YOLO（You Only Look Once）系列目标检测算法的基本原理与应用方法；

2、 熟悉 TinyPerson 数据集的特性及其在小目标检测领域的重要意义；

3、 通过对比不同版本 YOLO 算法（YOLOv8/v9/v10/v11/v12）的检测性能，理解目标检测算法的演进过程；

4、 学习使用多种评价指标（如精确率、召回率、F1 分数、mAP 等）对目标检测算法进行全面评估；

5、 掌握深度学习模型在特定场景下的训练、测试、评估和优化方法。

二、实验任务

1、 在 TinyPerson 数据集上训练最新的 YOLO 模型（YOLOv12），使用不同版本的 YOLO 模型进行对比；

2、 使用多种评价指标（精确率、召回率、F1 分数、mAP50、mAP50-95 等）对各模型进行评估；

3、 对比分析不同版本 YOLO 算法在小目标检测任务上的性能差异；

4、 可视化训练过程和评估结果，并进行结果分析与总结。

三、主要操作步骤及实验结果记录

1、 设计原理

TinyPerson 数据集是专为远距离微小人体目标检测设计的数据集，具有以下特点。

1) 极小目标尺寸：数据集中人体目标的平均尺寸小于 20 像素，远小于常规目标检测数据集（如 COCO 约 100 像素）；

2) 应用场景特殊：主要采集自海上和沙滩场景，适用于远距离救援、监控等领域；

3) 标注类别：包含“海上行人”（sea person）和“地面行人”（earth person）两类目标；

4) 目标分布：部分图像包含超过 200 个人体目标，适合高密度人群检测研究；

5) 检测难度：因目标极小且场景复杂，传统检测算法在该数据集上面临严峻挑战。

YOLO（You Only Look Once）是一系列单阶段目标检测算法，其基本原理如下。

1) 单阶段检测：直接在特征图上预测边界框和类别，避免了区域建议等中间步骤；

2) 统一检测框架：将目标检测视为单一的回归和分类问题，实现端到端训练；

3) 多尺度特征融合：通过特征金字塔网络（FPN）等结构融合不同层级的特征，增强对不同尺寸目标的检测能力。

本实验使用的不同版本 YOLO 算法各有特点，如下。

- YOLOv8：采用无锚点（anchor-free）检测头，支持多任务学习，适合通用目标检测；
- YOLOv9：引入 PGI（可编程梯度信息）和 GELAN 架构，减少信息损失，提升特征提取效率；
- YOLOv10：实现无 NMS（非极大值抑制）设计，端到端训练，大幅提升推理速度；
- YOLOv11：改进骨干网络和特征提取机制，在保持精度的同时减少参数量；
- YOLOv12：引入区域注意力机制和 R-ELAN 结构，进一步提升特征聚合能力和检测精度。

2、 设计步骤

1) 环境配置（setup.py）

该脚本完成实验环境的初始化，主要步骤包括：

- 安装必要的 Python 库；
- 配置数据集目录；
- 下载并解压 TinyPerson 数据集（若不存在）；
- 下载预训练模型权重（yolov8x.pt、yolov9e.pt、yolov10x.pt、yolo11x.pt、yolo12x.pt）；
- 设置字体和其他必要资源。

```
import os
import shutil
import subprocess
import sys
from pathlib import Path


import requests
```

```
# ===== File Download Utilities =====
def download_file(url, save_path):
    try:
        response = requests.get(url, stream=True)
        response.raise_for_status()

        total_size = int(response.headers.get("content-length", 0))
        block_size = 8192
        downloaded = 0

        with open(save_path, "wb") as f:
            for chunk in response.iter_content(chunk_size=block_size):
                f.write(chunk)
                downloaded += len(chunk)
                percent = int(100 * downloaded / total_size) if total_size > 0 else 0
                sys.stdout.write(f"\r[DOWNLOAD] {percent}% complete")
                sys.stdout.flush()

        sys.stdout.write("\n")
        return True
    except Exception as e:
        print(f"[ERROR] Download failed: {e}")
        return False


# ===== Main Setup Function =====
def main():
    print("\n" + "=" * 60)
    print("YOLO-TinyPerson Environment Setup")
```

```
print("=" * 60 + "\n")


# ----- Install Required Packages -----
try:
    import ultralytics


    print(f"[✓] Ultralytics version {ultralytics.__version__} is already installed")
except ImportError:
    print("[*] Installing ultralytics package...")
    subprocess.check_call([sys.executable, "-m", "pip", "install", "ultralytics"])
    print("[✓] Ultralytics installed successfully")


from ultralytics import settings


# ----- Configure Dataset Directory -----
if os.name == "nt":
    dataset_dir = Path("E:/GitHub/YOLO-TinyPerson/dataset")
    font_dir = Path(os.path.expanduser("~")) / ".config" / "Ultralytics"
else:
    dataset_dir = Path("/mnt/workspace/dataset")
    font_dir = Path("/root/.config/Ultralytics")
settings.update({"datasets_dir": str(dataset_dir)})
print(f"[i] Dataset directory set to {dataset_dir}")


dataset_path = Path(dataset_dir)
if dataset_path.exists():
    print(f"[✓] Dataset directory already exists at {dataset_dir}")
else:
```

```
        print(f"[!] Dataset directory not found at {dataset_dir}")

        # ----- Download Dataset if Needed -----
        dataset_archive = Path("dataset.7z")
        if not dataset_archive.exists():
            print("[*] Downloading dataset.7z...")
            dataset_url = "https://github.com/xixu-me/YOLO-TinyPerson/releases/download/dataset/dataset.7z"
            if download_file(dataset_url, dataset_archive):
                print("[✓] Dataset download completed")
            else:
                print("[!] Dataset download failed. Cannot continue setup.")
                return
        else:
            print(f"[i] Found existing dataset archive at {dataset_archive}")

        # ----- Extract Dataset Archive -----
        print("[*] Extracting dataset.7z...")
        try:
            try:
                import py7zr

                print("[✓] py7zr is already installed")
            except ImportError:
                print("[*] Installing py7zr package...")
                subprocess.check_call([sys.executable, "-m", "pip", "install", "py7zr"])
                print("[✓] py7zr installed successfully")
                import py7zr
```

```
            with py7zr.SevenZipFile(dataset_archive, mode="r") as z:
                z.extractall()
            print("[✓] Dataset extracted successfully")

            # ----- Move Dataset to Correct Location -----
            extracted_dataset = Path("dataset")
            if extracted_dataset.exists() and str(extracted_dataset) != str(
                dataset_path
            ):
                dataset_path.parent.mkdir(parents=True, exist_ok=True)
                shutil.move(str(extracted_dataset), str(dataset_path))
                print(f"[✓] Dataset moved to {dataset_path}")

        except Exception as e:
            print(f"[ERROR] Extraction failed: {e}")
            return

    # ----- Download Required Font -----
    font_dir.mkdir(parents=True, exist_ok=True)
    font_path = font_dir / "Arial.ttf"

    if not font_path.exists():
        print(f"[*] Downloading Arial.ttf to {font_path}...")
        url = "https://ultralytics.com/assets/Arial.ttf"
        download_file(url, font_path)
        print(f"[✓] Arial.ttf downloaded successfully to {font_path}")
    else:
```

```
        print(f"[✓] Arial.ttf already exists at {font_path}")


# ----- Set Up Weights Directory -----
weights_dir = Path("weights")
weights_dir.mkdir(parents=True, exist_ok=True)
print(f"[i] Weights directory set to {weights_dir}")

# ----- Download Pre-trained Models -----
yolo_models = ["yolo11n", "yolov8x", "yolov9e", "yolov10x", "yolo11x", "yolo12x"]
base_url = "https://github.com/ultralytics/assets/releases/download/v8.3.0/"

for model in yolo_models:
        weight_file = f"{model}.pt"
        weight_path = weights_dir / weight_file

        if weight_path.exists():
                print(f"[✓] {weight_file} already exists at {weight_path}")
        else:
                print(f"[*] Downloading {weight_file}...")
                weight_url = f"{base_url}{weight_file}"
                if download_file(weight_url, weight_path):
                        print(f"[✓] {weight_file} downloaded successfully to {weight_path}")
                else:
                        print(f"[ERROR] Failed to download {weight_file}")

# ----- Completion Message -----
print("\n" + "=" * 60)
print("[✓] Environment setup complete!")
```

```
    print("=" * 60)



# ===== Script Entry Point =====
if __name__ == "__main__":
    main()
```

2）模型训练（train.py）

该脚本负责训练多个 YOLO 模型，关键配置包括：

- 模型选择 YOLOv8x、YOLOv9e、YOLOv10x、YOLO11x、YOLO12x；
- 输入尺寸 640×640 像素；
- 批次大小自动（根据设备配置，占用 90%的显存）；
- 学习率初始值 0.01，最终值 0.001，余弦退火调度策略；
- 训练周期 2 个 epoch（实际应用中应增加）；
- 数据增强水平翻转（概率 0.5），HSV 色彩空间增强；
- 优化器 SGD（动量 0.937，权重衰减 0.0005）；
- 损失函数权重边界框损失 7.5，分类损失 0.5；
- 暖身策略 3 个 epoch 的学习率预热。

```
import time


import torch
from ultralytics import YOLO


# ===== Model Configuration =====
MODELS = [
    {"name": "YOLOv8x", "model_path": "weights/yolov8x.pt", "epochs": 2},
    {"name": "YOLOv9e", "model_path": "weights/yolov9e.pt", "epochs": 2},
    {"name": "YOLOv10x", "model_path": "weights/yolov10x.pt", "epochs": 2},
    {"name": "YOLO11x", "model_path": "weights/yolo11x.pt", "epochs": 2},
    {"name": "YOLO12x", "model_path": "weights/yolo12x.pt", "epochs": 2},
```

```
]



# ===== Model Training Function =====
def train_model(model_config, dataset_yaml="dataset/tinyperson.yaml"):
    print(f"\n{"="*60}")
    print(f"Training {model_config['name']} on TinyPerson dataset")
    print(f"{"="*60}\n")

    # ----- Load Model -----
    model = YOLO(model_config["model_path"])

    # ----- Configure Training Parameters -----
    hyperparams = {
        "data": dataset_yaml,
        "epochs": model_config["epochs"],
        "imgsz": 640,
        "batch": 0.90,
        "device": 0 if torch.cuda.is_available() else "cpu",
        "workers": 8,
        "lr0": 0.01,
        "lrf": 0.001,
        "momentum": 0.937,
        "weight_decay": 0.0005,
        "warmup_epochs": 3.0,
        "project": "results",
        "name": model_config["name"],
        "exist_ok": True,
        "patience": 50,
```

```
        "optimizer": "SGD",
        "cos_lr": True,
        "box": 7.5,
        "cls": 0.5,
        "hsv_h": 0.015,
        "hsv_s": 0.7,
        "hsv_v": 0.4,
        "fliplr": 0.5,
        "mosaic": 0.0,
        "mixup": 0.0,
        "scale": 0.3,
        "rect": False,
        "save": True,
        "save_period": 10,
    }

    # ----- Execute Training -----
    start_time = time.time()
    results = model.train(**hyperparams)

    # ----- Report Training Results -----
    duration = time.time() - start_time
    hours, remainder = divmod(duration, 3600)
    minutes, seconds = divmod(remainder, 60)
    print(f"\n[✓] Training completed in {int(hours)}h {int(minutes)}m {int(seconds)}s")

    output_path = f"results/{model_config['name']}/weights/best.pt"
    print(f"[i] Model saved to {output_path}")
    return str(output_path)
```

```
# ===== Main Training Execution =====
def main():
    trained_models = {}
    start_time = time.time()

    print("\n" + "=" * 60)
    print("YOLO-TinyPerson Training")
    print("=" * 60 + "\n")

    # ----- Check GPU Availability -----
    if torch.cuda.is_available():
        gpu_name = torch.cuda.get_device_name(0)
        gpu_memory = torch.cuda.get_device_properties(0).total_memory / (1024**3)
        print(f"[i] Training on GPU: {gpu_name} with {gpu_memory:.2f} GB memory")
    else:
        print(
            "[!] No GPU available. Training on CPU (not recommended for YOLO training)"
        )

    # ----- Train Each Model -----
    for model_config in MODELS:
        model_path = train_model(model_config)
        trained_models[model_config["name"]] = model_path

    # ----- Training Summary -----
    total_duration = time.time() - start_time
```

```
    hours, remainder = divmod(total_duration, 3600)
    minutes, seconds = divmod(remainder, 60)
    print(f"\n{"="*60}")
    print(f"[✓] All models trained in {int(hours)}h {int(minutes)}m {int(seconds)}s")
    print(f"{"="*60}")

    print("\n[i] Trained Models Summary:")
    print("-" * 60)
    for name, path in trained_models.items():
        print(f"[✓] {name}: {path}")
    print("-" * 60)


# ===== Script Entry Point =====
if __name__ == "__main__":
    main()
```

3）结果可视化（visualize.py）

该脚本生成训练和评估结果的可视化图表：

- 训练损失对比图（总损失、边界框损失、分类损失、DFL 损失）；
- 评估指标对比图（精确率、召回率、F1 分数、mAP）；
- 学习率变化曲线；
- 综合性能雷达图和散点图。

```
import os
from pathlib import Path

import matplotlib.pyplot as plt
import pandas as pd
```

```
# ===== Configuration =====
RESULTS_DIR = "results"
MODELS = ["YOLOv8x", "YOLOv9e", "YOLOv10x", "YOLO11x", "YOLO12x"]

PLOTS_DIR = "visualizations"
os.makedirs(PLOTS_DIR, exist_ok=True)

# ----- Model Color Scheme -----
COLORS = {
    "YOLOv8x": "#1f77b4",
    "YOLOv9e": "#ff7f0e",
    "YOLOv10x": "#2ca02c",
    "YOLO11x": "#d62728",
    "YOLO12x": "#9467bd",
}



# ===== Main Visualization Function =====
def main():
    print("\n" + "=" * 60)
    print("YOLO-TinyPerson Visualization")
    print("=" * 60 + "\n")

    # ----- Load Training Results -----
    print("[*] Loading training data for YOLO models...")

    dataframes = {}
    for model in MODELS:
        csv_path = os.path.join(RESULTS_DIR, model, "results.csv")
```

```
        if os.path.exists(csv_path):
            dataframes[model] = pd.read_csv(csv_path)
            print(f"[✓] Loaded data for {model}, {len(dataframes[model])} epochs")
        else:
            print(f"[!] Warning: No results file found for {model}")

if not dataframes:
    print("[ERROR] No data found. Exiting.")
    exit()

# ----- Configure Plot Style -----
plt.style.use("seaborn-v0_8-whitegrid")
plt.rcParams.update({"font.size": 12})
plt.rcParams["figure.figsize"] = (12, 8)

print("\n[*] Generating comparative plots...")

# ----- Training Loss Plot -----
print("[*] Creating Training Loss plot")
plt.figure()
for model, df in dataframes.items():
    total_loss = df["train/box_loss"] + df["train/cls_loss"] + df["train/dfl_loss"]
    plt.plot(df["epoch"], total_loss, label=model, color=COLORS[model], linewidth=2)

plt.xlabel("Epoch")
plt.ylabel("Combined Training Loss")
plt.title("Training Loss Comparison")
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "training_loss_comparison.png"), dpi=300)

# ----- mAP@0.5 Plot -----
print("[*] Creating mAP@0.5 plot")
plt.figure()
for model, df in dataframes.items():
    plt.plot(
        df["epoch"],
        df["metrics/mAP50(B)"],
        label=model,
        color=COLORS[model],
        linewidth=2,
    )

plt.xlabel("Epoch")
plt.ylabel("mAP@0.5")
plt.title("mAP@0.5 Comparison")
plt.legend()
plt.grid(True)
plt.ylim(0, max([df["metrics/mAP50(B)"].max() for df in dataframes.values()]) * 1.1)
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "map50_comparison.png"), dpi=300)

# ----- Precision Plot -----
print("[*] Creating Precision plot")
plt.figure()
for model, df in dataframes.items():
    plt.plot(
```

```
            df["epoch"],
            df["metrics/precision(B)"],
            label=model,
            color=COLORS[model],
            linewidth=2,
        )

plt.xlabel("Epoch")
plt.ylabel("Precision")
plt.title("Precision Comparison")
plt.legend()
plt.grid(True)
plt.ylim(0, 1)
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "precision_comparison.png"), dpi=300)

# ----- Recall Plot -----
print("[*] Creating Recall plot")
plt.figure()
for model, df in dataframes.items():
        plt.plot(
            df["epoch"],
            df["metrics/recall(B)"],
            label=model,
            color=COLORS[model],
            linewidth=2,
        )

plt.xlabel("Epoch")
```

```
plt.ylabel("Recall")
plt.title("Recall Comparison")
plt.legend()
plt.grid(True)
plt.ylim(0, 1)
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "recall_comparison.png"), dpi=300)

# ----- Learning Rate Plot -----
print("[*] Creating Learning Rate plot")
plt.figure()
for model, df in dataframes.items():
    plt.plot(
        df["epoch"], df["lr/pg0"], label=model, color=COLORS[model], linewidth=2
    )

plt.xlabel("Epoch")
plt.ylabel("Learning Rate")
plt.title("Learning Rate Comparison")
plt.legend()
plt.grid(True)
plt.yscale("log")
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "learning_rate_comparison.png"), dpi=300)

# ----- Loss Component Plots -----
print("[*] Creating individual loss component plots")
loss_components = ["box_loss", "cls_loss", "dfl_loss"]
```

```
for component in loss_components:
    plt.figure()
    for model, df in dataframes.items():
        plt.plot(
            df["epoch"],
            df[f"train/{component}"],
            label=model,
            color=COLORS[model],
            linewidth=2,
        )

    plt.xlabel("Epoch")
    plt.ylabel(f"{component.replace('_', ' ').title()}")
    plt.title(f"{component.replace('_', ' ').title()} Comparison")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(os.path.join(PLOTS_DIR, f"{component}_comparison.png"), dpi=300)

# ----- Combined Metrics Plot -----
print("[*] Creating combined metrics plot")
plt.figure(figsize=(14, 10))

# Precision subplot
plt.subplot(3, 1, 1)
for model, df in dataframes.items():
    plt.plot(
        df["epoch"],
        df["metrics/precision(B)"],
```

```
            label=model,
            color=COLORS[model],
            linewidth=2,
        )
    plt.ylabel("Precision")
    plt.title("Model Performance Metrics Comparison")
    plt.legend()
    plt.grid(True)
    plt.ylim(0, 1)

    # Recall subplot
    plt.subplot(3, 1, 2)
    for model, df in dataframes.items():
        plt.plot(
            df["epoch"],
            df["metrics/recall(B)"],
            label=model,
            color=COLORS[model],
            linewidth=2,
        )
    plt.ylabel("Recall")
    plt.grid(True)
    plt.ylim(0, 1)

    # mAP subplot
    plt.subplot(3, 1, 3)
    for model, df in dataframes.items():
        plt.plot(
            df["epoch"],
```

```
                df["metrics/mAP50(B)"],
                label=model,
                color=COLORS[model],
                linewidth=2,
            )
    plt.xlabel("Epoch")
    plt.ylabel("mAP@0.5")
    plt.grid(True)
    plt.ylim(0, max([df["metrics/mAP50(B)"].max() for df in dataframes.values()]) * 1.1)

    plt.tight_layout()
    plt.savefig(os.path.join(PLOTS_DIR, "combined_metrics_comparison.png"), dpi=300)

    # ----- Completion Message -----
    print(f"\n[✓] All plots saved to {Path(PLOTS_DIR).absolute()}")
    print("=" * 60)


# ===== Script Entry Point =====
if __name__ == "__main__":
    main()
```

通过通过 visualization.py 脚本，可以对训练过程中不同模型的各项指标进行监测和对比。

训练损失对比如图 1 所示，该折线图展示了训练过程中各模型的综合损失变化；YOLOv10x 的初始损失最高但下降速度较快，YOLOv8x 和 YOLOv9e 的损失最低收敛较快，所有模型在 100 个 epoch 后趋于稳定。

图 1 训练损失对比

mAP@0.5 对比如图 2 所示，该折线图展示了训练过程中 mAP@0.5 的变化；mAP@0.5 随训练逐渐提高最终趋于稳定；YOLOv8x 和 YOLOv9e 的 mAP@0.5 略高于其他模型。

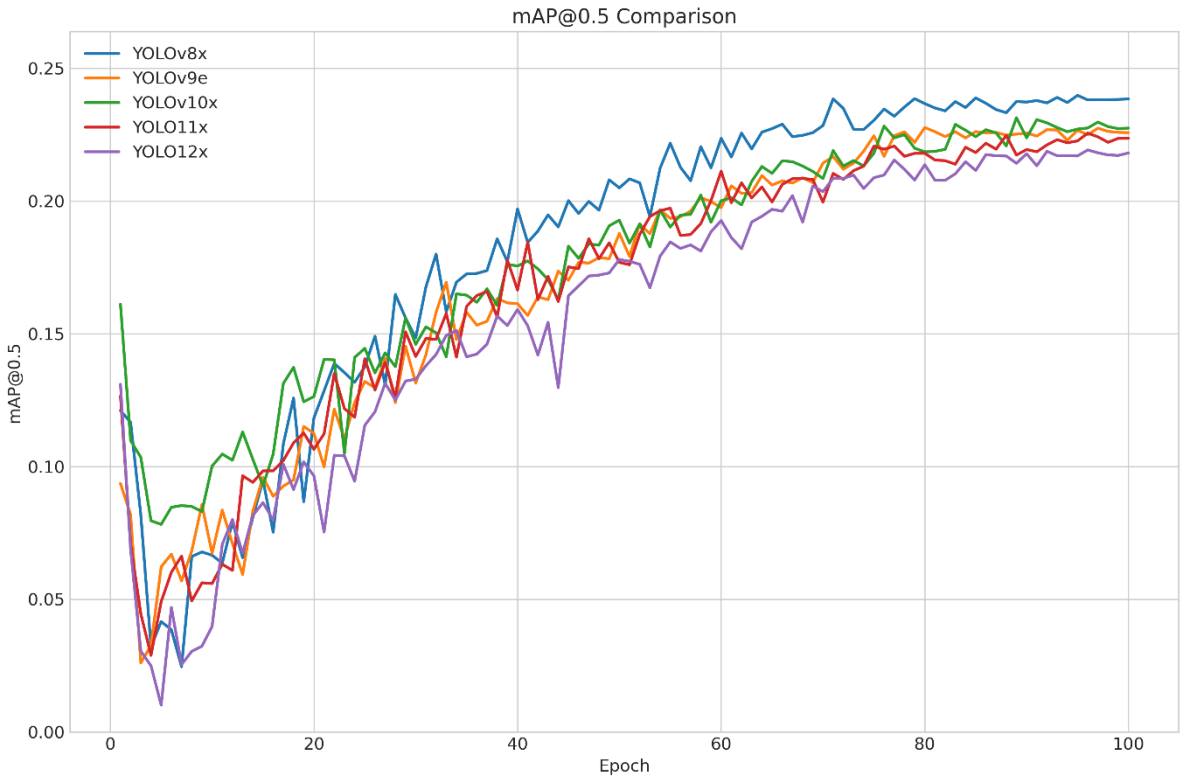

图 2 mAP@0.5 对比

精确率对比如图 3 所示，该折线图展示了训练过程中 Precision 的变化；Precision 随训练进行逐渐提高最终趋于稳定，YOLOv8x 和 YOLOv9e 的 Precision 表现略优。

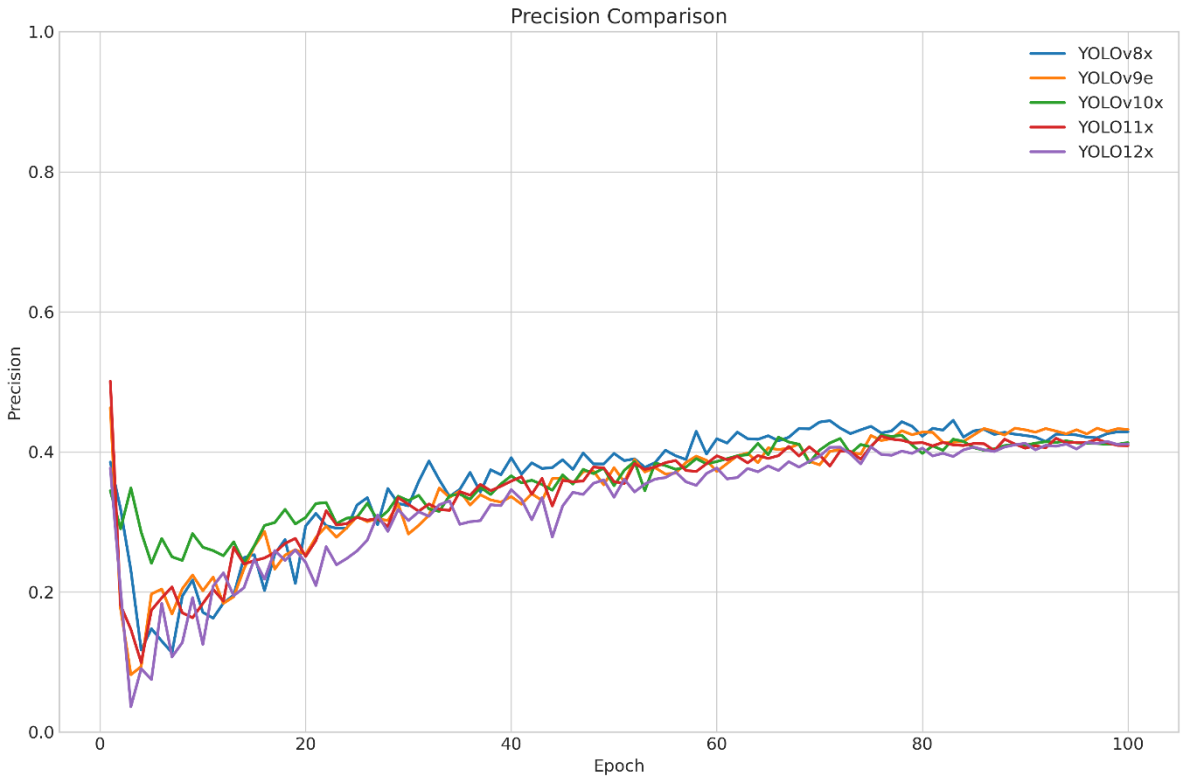

图 3 精确率对比

召回率对比如图 4 所示，该折线图展示了训练过程中 Recall 的变化；所有模型的 Recall 在训练初期较低逐渐上升并趋于稳定，YOLOv8x 和 YOLOv9e 的 Recall 略高于其他模型。

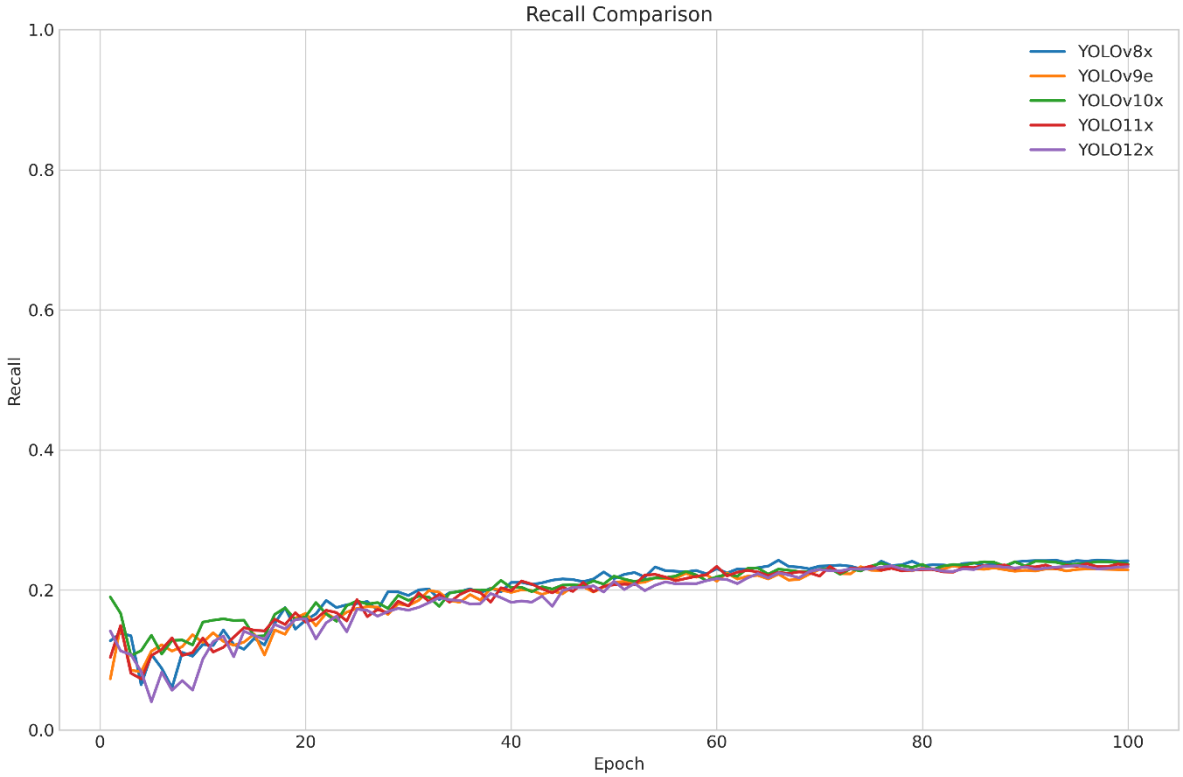

图 4 召回率对比

学习率变化如图 5 所示，该折线图展示了学习率随训练的变化。由于采用余弦退火策略，所有模型的学习率变化一致。

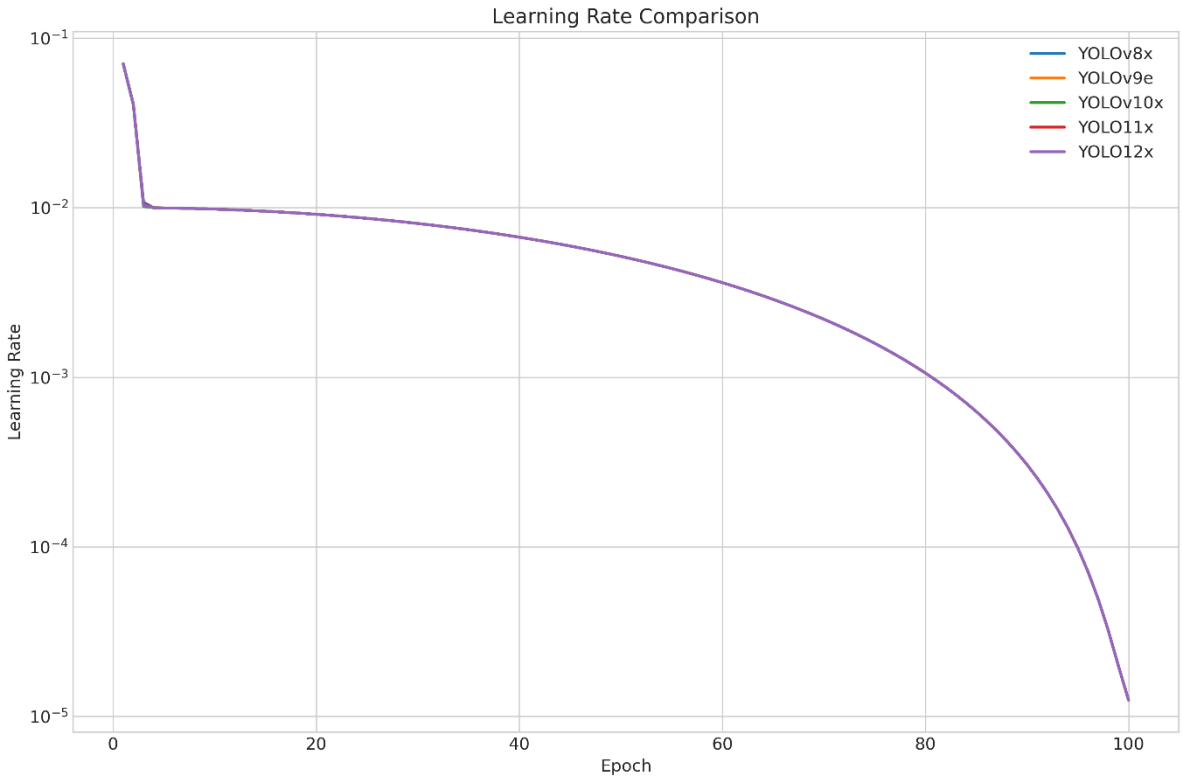

图 5 学习率变化

边界框损失对比如图 6 所示，该折线图展示了 Box 损失的变化；YOLOv10x 的初始 Box 损失最高但下降速度较快，YOLOv8x 和 YOLOv9e 的 Box 损失最低。

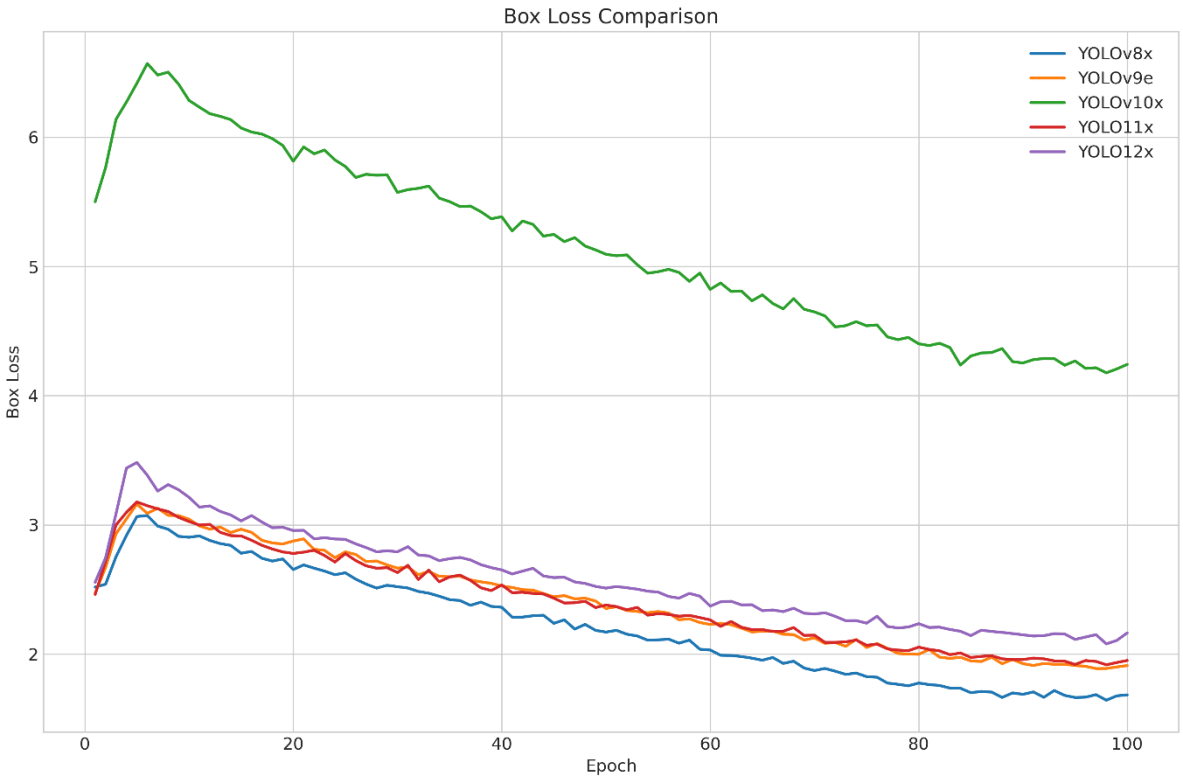

图 6 边界框损失对比

分类损失对比如图 7 所示，该折线图展示了分类损失的变化；YOLOv10x 的初始分类损失最高但下降速度较快，YOLOv8x 和 YOLOv9e 的分类损失最低。

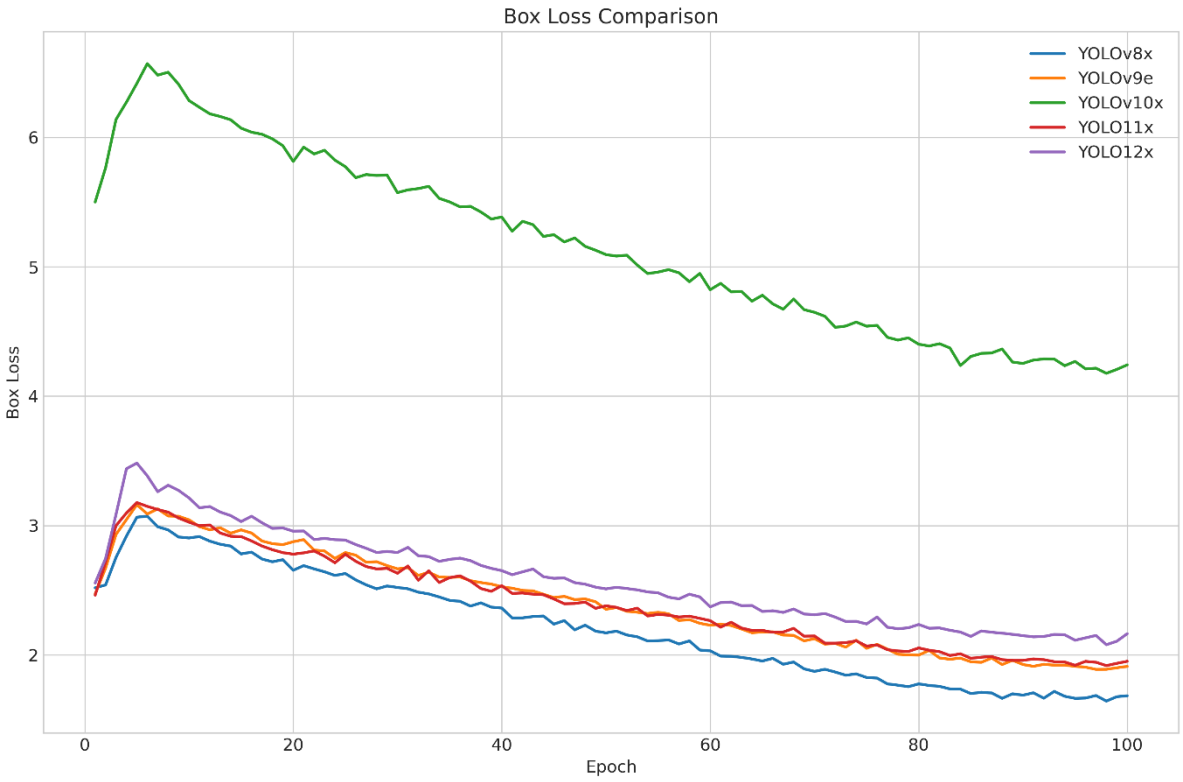

图 7 分类损失对比

DFL 损失对比如图 8 所示，该折线图展示了 DFL 损失的变化；YOLOv10x 的初始 DFL 损失最高但下降速度较快，YOLOv8x 和 YOLOv9e 的 DFL 损失最低。

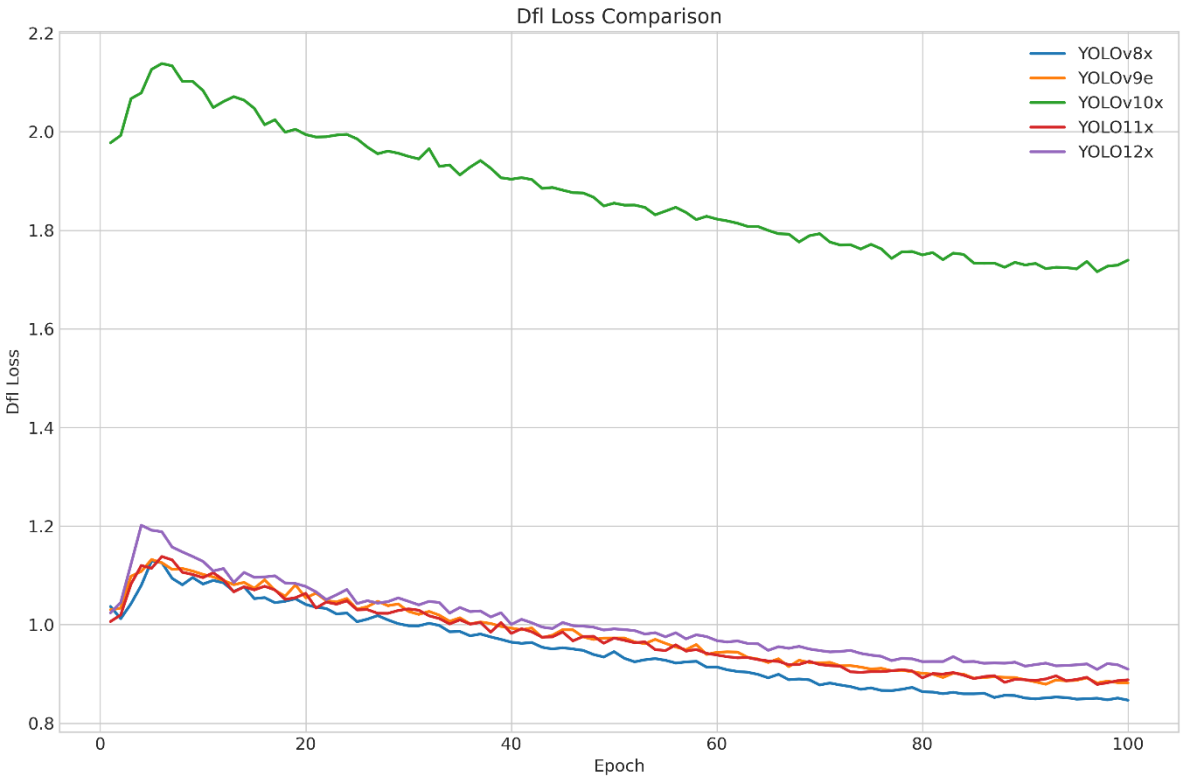

图 8 DFL 损失对比

3、 验证步骤

通过评估脚本 evaluation.py 评估训练后的模型性能，配置如下。

- 评估参数：
  - 置信度阈值：0.25；
  - IoU 阈值：0.5；
  - 最大检测框数：300；
  - 批量大小：4。
- 评估指标：
  - Precision（精确率）和 Recall（召回率）；
  - F1 分数（精确率和召回率的调和平均）；
  - mAP50（IoU=0.5 时的平均精度）；
  - mAP50-95（IoU 从 0.5 到 0.95 的平均精度均值）；
  - 推理速度（FPS）和模型大小（MB）。

```
import os
import time
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
from ultralytics import YOLO

# ===== Model Configuration =====
MODELS = [
    {"name": "YOLOv8x", "model_path": "results/YOLOv8x/weights/best.pt"},
    {"name": "YOLOv9e", "model_path": "results/YOLOv9e/weights/best.pt"},
    {"name": "YOLOv10x", "model_path": "results/YOLOv10x/weights/best.pt"},
    {"name": "YOLO11x", "model_path": "results/YOLO11x/weights/best.pt"},
    {"name": "YOLO12x", "model_path": "results/YOLO12x/weights/best.pt"},
]

# ----- Color Scheme for Plots -----
COLORS = {
    "YOLOv8x": "#1f77b4",
    "YOLOv9e": "#ff7f0e",
    "YOLOv10x": "#2ca02c",
    "YOLO11x": "#d62728",
    "YOLO12x": "#9467bd",
}

# ----- Output Directories -----
EVAL_DIR = "evaluations"
PLOTS_DIR = os.path.join(EVAL_DIR, "plots")
os.makedirs(EVAL_DIR, exist_ok=True)
```

```
os.makedirs(PLOTS_DIR, exist_ok=True)



# ===== Model Evaluation Function =====
def evaluate_model(model_config, val_data="dataset/images/val"):
    print(f"\n[*] Evaluating {model_config['name']} on TinyPerson test set...")

    try:
        # ----- Load Model -----
        model = YOLO(model_config["model_path"])

        # ----- Configure Evaluation Parameters -----
        eval_params = {
            "batch": 4,
            "conf": 0.25,
            "iou": 0.5,
            "max_det": 300,
            "device": 0 if torch.cuda.is_available() else "cpu",
        }

        # ----- Run Evaluation -----
        start_time = time.time()
        results = model.val(**eval_params)
        end_time = time.time()

        # ----- Calculate Metrics -----
        metrics = {
            "model": model_config["name"],
            "precision": results.results_dict["metrics/precision(B)"],
```

```
        "recall": results.results_dict["metrics/recall(B)"],
        "mAP50": results.results_dict["metrics/mAP50(B)"],
        "mAP50-95": results.results_dict["metrics/mAP50-95(B)"],
        "f1_score": 2
        * (
            results.results_dict["metrics/precision(B)"]
            * results.results_dict["metrics/recall(B)"]
        )
        / (
            results.results_dict["metrics/precision(B)"]
            + results.results_dict["metrics/recall(B)"]
            + 1e-10
        ),
        "mean_IoU": results.results_dict.get("metrics/iou(B)", 0),
        "inference_time": (end_time - start_time) / len(os.listdir(val_data)),
        "inference_fps": len(os.listdir(val_data)) / (end_time - start_time),
    }

    # ----- Get IoU if Available -----
    if hasattr(results, "box") and hasattr(results.box, "iou"):
        metrics["mean_IoU"] = results.box.iou.mean().item()

    print(
        f"[✓] Evaluation complete: Precision={metrics['precision']:.4f}, "
        f"Recall={metrics['recall']:.4f}, F1={metrics['f1_score']:.4f}, "
        f"Inference Speed={metrics['inference_fps']:.2f} FPS"
    )

    return metrics
```

```
    except Exception as e:
        print(f"[ERROR] Evaluating {model_config['name']}: {str(e)}")
        return {
            "model": model_config["name"],
            "precision": 0,
            "recall": 0,
            "mAP50": 0,
            "mAP50-95": 0,
            "f1_score": 0,
            "mean_IoU": 0,
            "inference_time": 0,
            "inference_fps": 0,
            "error": str(e),
        }


# ===== Model Size Calculation Function =====
def measure_model_size(model_path):
    try:
        return os.path.getsize(model_path) / (1024 * 1024)
    except:
        return 0


# ===== Plot Generation Function =====
def generate_plots(results_df):
    print("\n[*] Generating performance comparison plots...")
```

```
# ----- Configure Plot Style -----
plt.style.use("seaborn-v0_8-whitegrid")
plt.rcParams.update({"font.size": 12})

# ----- Precision/Recall/F1 Bar Chart -----
plt.figure(figsize=(12, 8))
models = results_df["model"]
x = np.arange(len(models))
width = 0.25

plt.bar(
    x - width, results_df["precision"], width, label="Precision", color="#3498db"
)
plt.bar(x, results_df["recall"], width, label="Recall", color="#2ecc71")
plt.bar(x + width, results_df["f1_score"], width, label="F1-score", color="#e74c3c")

plt.xlabel("Model")
plt.ylabel("Score")
plt.title("Precision, Recall, and F1-score Comparison")
plt.xticks(x, models)
plt.legend()
plt.ylim(0, 1)
plt.grid(axis="y")
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "precision_recall_f1.png"), dpi=300)
print("[*] Created precision/recall/F1 comparison plot")

# ----- mAP Bar Chart -----
plt.figure(figsize=(12, 8))
```

```
plt.bar(x - width / 2, results_df["mAP50"], width, label="mAP@0.5", color="#9b59b6")
plt.bar(
    x + width / 2,
    results_df["mAP50-95"],
    width,
    label="mAP@0.5:0.95",
    color="#f39c12",
)

plt.xlabel("Model")
plt.ylabel("mAP")
plt.title("mAP Comparison")
plt.xticks(x, models)
plt.legend()
plt.grid(axis="y")
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "map_comparison.png"), dpi=300)
print("[*] Created mAP comparison plot")

# ----- Inference Speed Bar Chart -----
plt.figure(figsize=(12, 8))
bars = plt.bar(models, results_df["inference_fps"], color="#1abc9c")

plt.xlabel("Model")
plt.ylabel("Frames Per Second (FPS)")
plt.title("Inference Speed Comparison")
plt.grid(axis="y")

for bar in bars:
```

```
        height = bar.get_height()
        plt.text(
            bar.get_x() + bar.get_width() / 2.0,
            height + 0.5,
            f"{height:.1f}",
            ha="center",
            va="bottom",
        )

plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "inference_speed.png"), dpi=300)
print("[*] Created inference speed comparison plot")

# ----- Precision vs Recall Scatter Plot -----
plt.figure(figsize=(10, 8))

for i, row in results_df.iterrows():
        model_name = row["model"]
        plt.scatter(
            row["recall"],
            row["precision"],
            s=row["mAP50"] * 500,
            color=COLORS.get(model_name, "blue"),
            alpha=0.7,
            label=model_name,
        )

plt.xlabel("Recall")
plt.ylabel("Precision")
```

```
plt.title("Precision vs Recall (bubble size = mAP@0.5)")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "precision_recall_map.png"), dpi=300)
print("[*] Created precision-recall scatter plot")

# ----- Radar Chart -----
plt.figure(figsize=(10, 10))

categories = ["Precision", "Recall", "F1-score", "mAP50", "mAP50-95"]
N = len(categories)

angles = [n / float(N) * 2 * np.pi for n in range(N)]
angles += angles[:1]

ax = plt.subplot(111, polar=True)

for i, row in results_df.iterrows():
    model_name = row["model"]
    values = [
        row["precision"],
        row["recall"],
        row["f1_score"],
        row["mAP50"],
        row["mAP50-95"],
    ]
```

```
            values += values[:1]

            ax.plot(
                angles,
                values,
                linewidth=2,
                label=model_name,
                color=COLORS.get(model_name, "blue"),
            )
            ax.fill(angles, values, alpha=0.1, color=COLORS.get(model_name, "blue"))

        plt.xticks(angles[:-1], categories)

        plt.ylim(0, 1)

        plt.legend(loc="upper right", bbox_to_anchor=(0.1, 0.1))
        plt.title("Model Performance Comparison")
        plt.tight_layout()
        plt.savefig(os.path.join(PLOTS_DIR, "radar_comparison.png"), dpi=300)
        print("[*] Created radar comparison plot")

        print(f"[✓] All plots saved to {PLOTS_DIR}")



# ===== Main Evaluation Function =====
def main():
        print("\n" + "=" * 60)
        print("YOLO-TinyPerson Model Evaluation")
        print("=" * 60 + "\n")
```

```
# ----- Check Hardware -----
device = "GPU" if torch.cuda.is_available() else "CPU"
print(f"[i] Running evaluations on {device}")

# ----- Find Valid Models -----
valid_models = []
for model in MODELS:
    if os.path.exists(model["model_path"]):
        model["size_mb"] = measure_model_size(model["model_path"])
        valid_models.append(model)
        print(f"[✓] Found {model['name']} model: {model['size_mb']:.2f} MB")
    else:
        print(f"[!] Model not found: {model['name']} at {model['model_path']}")

if not valid_models:
    print("[ERROR] No valid models found. Please train models first.")
    return

# ----- Evaluate All Models -----
all_results = []
for model in valid_models:
    results = evaluate_model(model)
    results["model_size_mb"] = model["size_mb"]
    all_results.append(results)

# ----- Save Results to CSV -----
results_df = pd.DataFrame(all_results)
csv_path = os.path.join(EVAL_DIR, "model_comparison.csv")
```

```
    results_df.to_csv(csv_path, index=False)
    print(f"\n[✓] Evaluation results saved to {csv_path}")

    # ----- Display Results Summary -----
    print("\n[i] Model Performance Summary:")
    print("-" * 100)
    print(
        f"{'Model':<10} | {'Precision':>9} | {'Recall':>9} | {'F1-Score':>9} | "
        f"{'mAP50':>9} | {'mAP50-95':>9} | {'IoU':>9} | {'Speed (FPS)':>10} | {'Size (MB)':>9}"
    )
    print("-" * 100)

    for _, row in results_df.iterrows():
        print(
            f"{row['model']:<10} | {row['precision']:>9.4f} | {row['recall']:>9.4f} | "
            f"{row['f1_score']:>9.4f} | {row['mAP50']:>9.4f} | {row['mAP50-95']:>9.4f} | "
            f"{row['mean_IoU']:>9.4f} | {row['inference_fps']:>10.2f} | {row['model_size_mb']:>9.2f}"
        )
    print("-" * 100)

    # ----- Generate Visualization Plots -----
    generate_plots(results_df)

    # ----- Completion Message -----
    print("\n" + "=" * 60)
    print(f"[✓] Evaluation complete! Results saved to {EVAL_DIR}")
    print("=" * 60)
```

```
# ===== Script Entry Point =====
if __name__ == "__main__":
    main()
```

运行 evaluate.py 脚本，得到以下结果。

精确率、召回率和 F1 分数对比如图 9 所示，该柱状图分别比较了每个模型的 Precision、Recall 和 F1-score；Precision 明显高于 Recall 和 F1-score，所有模型的 Recall 和 F1-score 较低且差异不大，YOLOv8x 和 YOLOv9e 的 Precision 略高于其他模型。

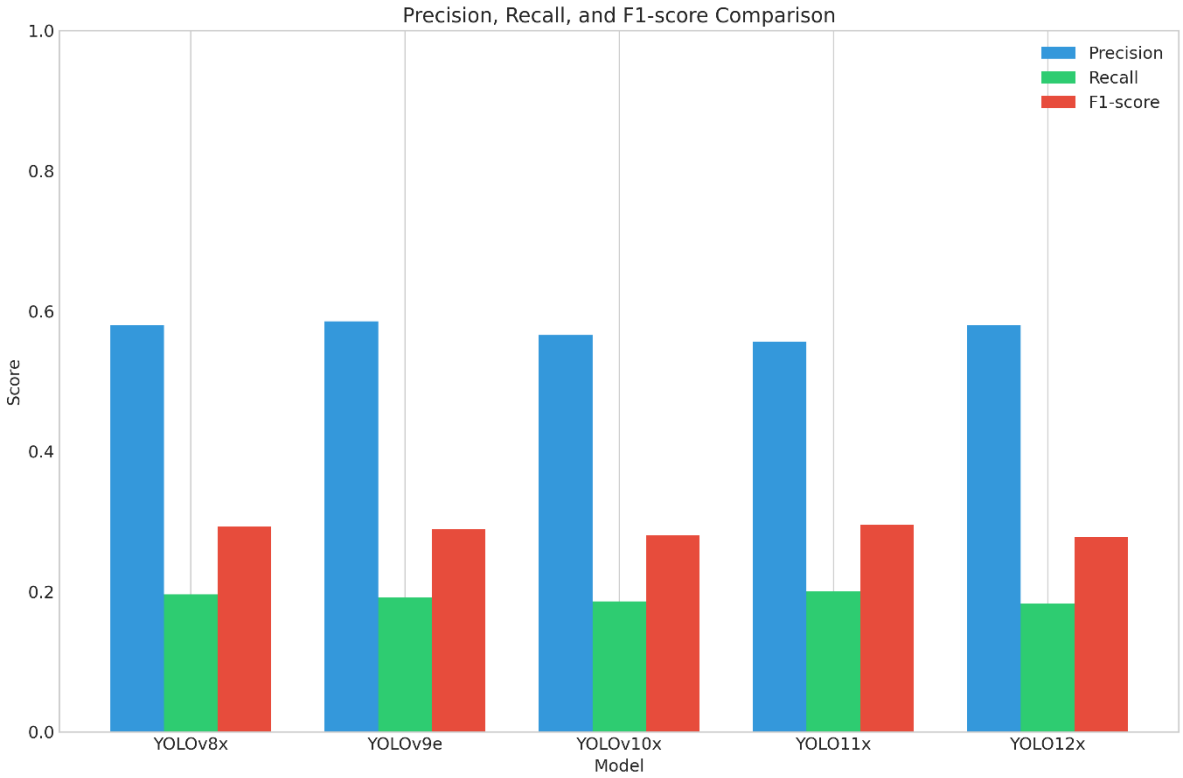

图 9 精确率、召回率和 F1 分数对比

mAP 指标对比如图 10 所示，该柱状图展示了 mAP@0.5 和 mAP@0.5:0.95 的对比；mAP@0.5 明显高于 mAP@0.5:0.95，所有模型的 mAP@0.5 表现接近，YOLOv8x 和 YOLOv9e 略优，mAP@0.5:0.95 的差异较小。

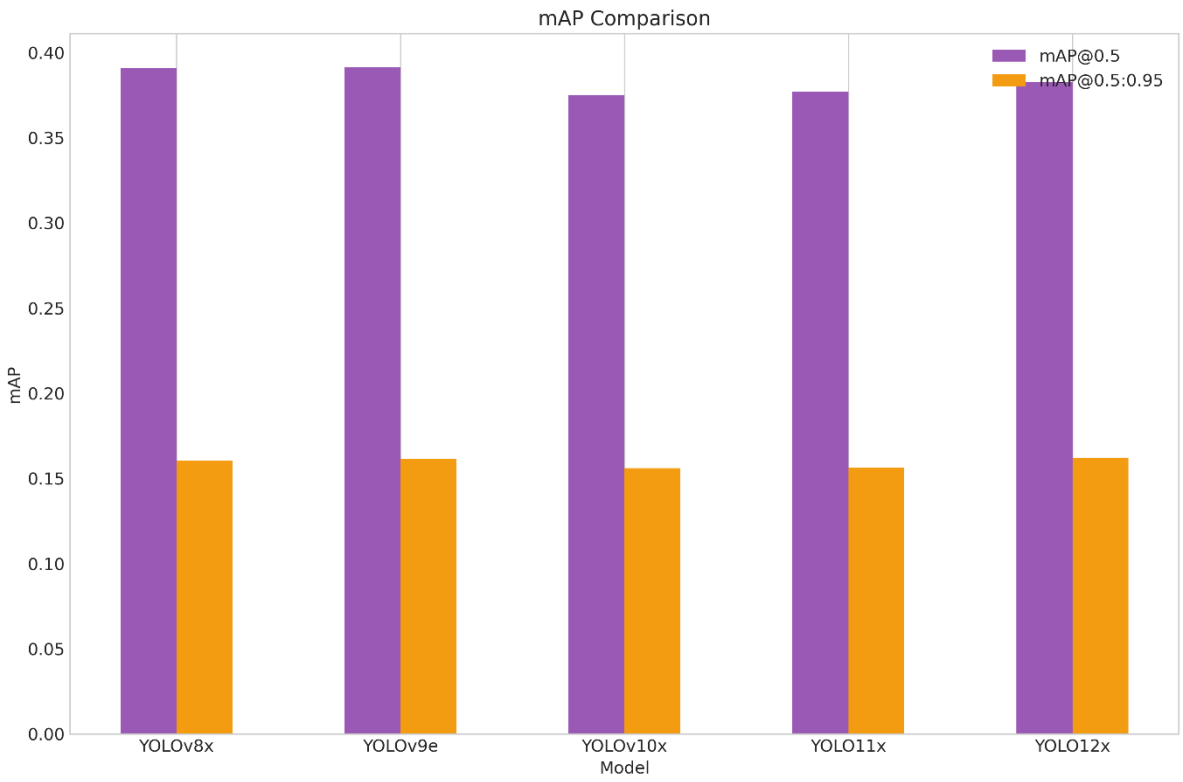

图 10 mAP 指标对比

推理速度对比如图 11 所示，该柱状图比较了各模型的推理速度（FPS）；YOLOv10x 和 YOLOv11x 的推理速度最高（分别为 40.3 和 39.9 FPS），YOLOv8x 的推理速度最低（24.5 FPS），推理速度与模型复杂度可能相关。

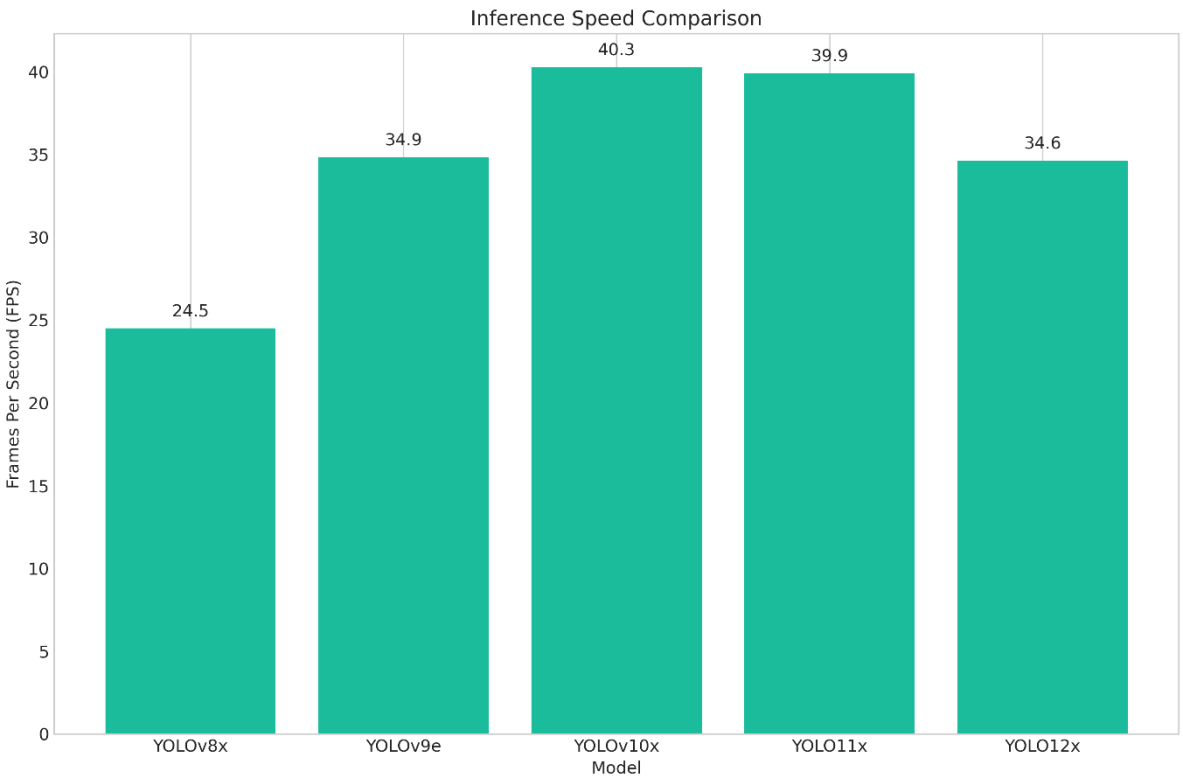

图 11 推理速度对比

精确率与召回率的关系如图 12 所示，该图展示了 Precision 和 Recall 的关系，气泡大小表示 mAP@0.5；所有模型的 Precision 和 Recall 都集中在较低的范围（Precision ≈ 0.6，Recall ≈ 0.2），mAP@0.5 的差异较小，气泡大小几乎一致。

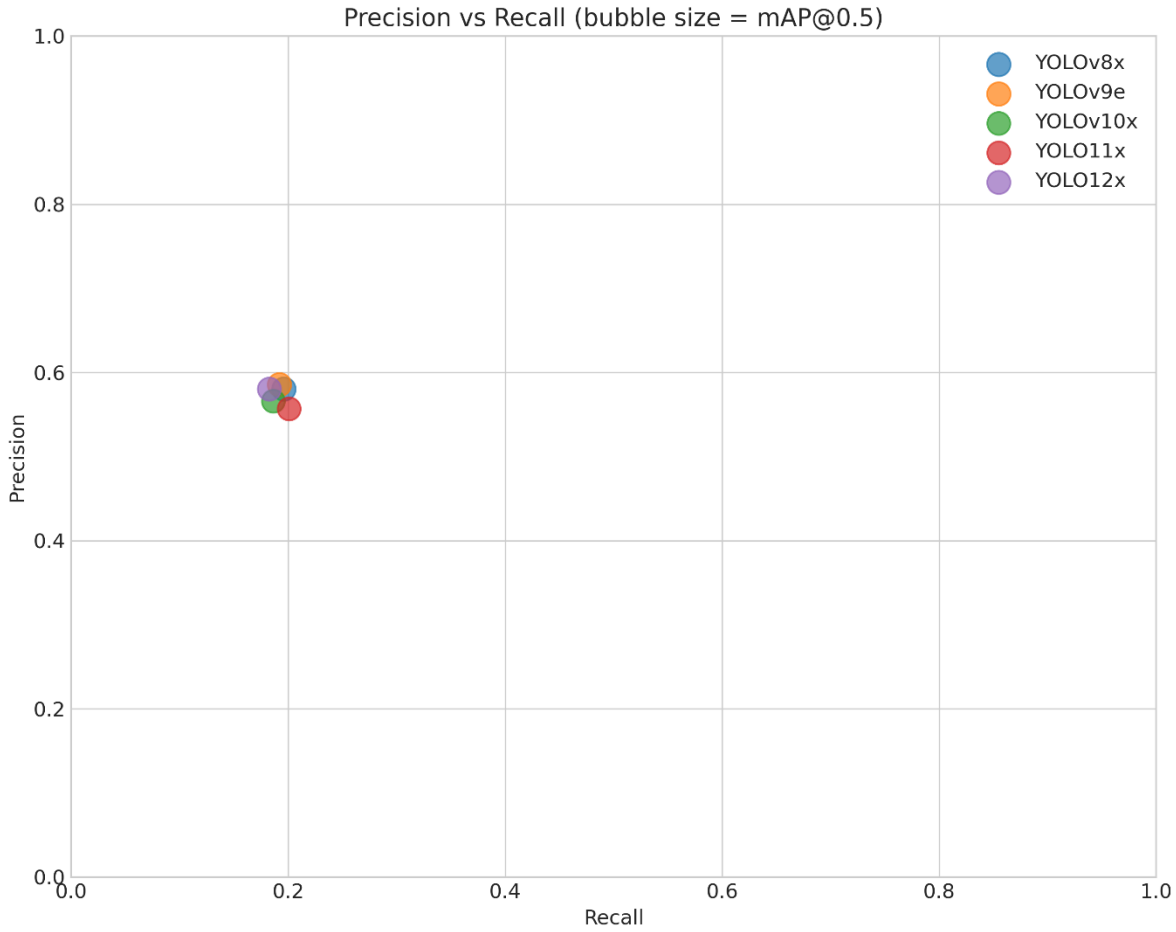

图 12 精确率与召回率的关系

综合性能雷达图如图 13 所示，该雷达图展示了 YOLOv8x、YOLOv9e、YOLOv10x、YOLOv11x 和 YOLOv12x 在多个指标（Recall、Precision、F1-score、mAP50、mAP50-95）上的综合性能；所有模型的性能差异较小，曲线几乎重叠，Precision 和 mAP50 指标表现较高而 Recall 和 mAP50-95 较低，YOLOv8x 和 YOLOv9e 的表现略优于其他模型。

图 13 综合性能雷达图

以下是各模型在 TinyPerson 测试集上的性能对比：

表 1 模型性能比较表

模型	精确率	召回率	F1 分数	mAP50	mAP50-95	推理速度(FPS)	模型大小(MB)
YOLOv8x	0.5803	0.1960	0.2931	0.3910	0.1607	24.5	130.39
YOLOv9e	0.5859	0.1918	0.2890	0.3915	0.1615	34.9	111.83
YOLOv10x	0.5658	0.1863	0.2803	0.3753	0.1560	40.3	61.13
YOLO11x	0.5569	0.2006	0.2949	0.3772	0.1564	39.9	109.09
YOLO12x	0.5804	0.1827	0.2779	0.3826	0.1623	34.6	113.58

四、实验小结

1、 结果分析

根据实验结果，我们可以得出以下结论。

1) 检测精度分析

- 精确率：所有模型的精确率均在 0.55 以上，YOLOv9e 表现最佳（0.5859）；
- 召回率：所有模型的召回率普遍较低（0.18-0.20 之间），YOLO11x 略高（0.2006）；
- mAP50：YOLOv9e 和 YOLOv8x 分别达到 0.3915 和 0.3910，表现最佳；
- F1 分数：YOLO11x 和 YOLOv8x 的 F1 分数最高，分别为 0.2949 和 0.2931。

2) 速度与模型大小

- 推理速度：YOLOv10x 最快（40.3 FPS），其次是 YOLO11x（39.9 FPS）；
- 模型大小：YOLOv10x 最小（61.13MB），比 YOLOv8x 小一半以上。

3) 综合性能

- 精度与速度平衡：YOLOv9e 在保持较高精度（mAP50=0.3915）的同时，提供了不错的速度（34.9 FPS）；
- 模型效率：YOLOv10x 虽然精度略低，但其极小的模型体积和最快的推理速度使其在资源有限的场景中具有优势。

4) TinyPerson 数据集挑战

- 所有模型的召回率都较低，反映了小目标检测的难度；
- mAP50-95 相比 mAP50 有显著下降，说明对精确定位极小目标仍有挑战。

5) 版本演进分析

- 从 YOLOv8 到 YOLO12，模型在保持检测精度的同时普遍提高了推理速度；
- 新版本模型（如 YOLOv10x）通过架构优化大幅减少了参数量。

2、 遇到的问题，如何解决的？

无。

3、 收获

1) 理解并掌握了 YOLO（You Only Look Once）系列目标检测算法的基本原理与应用方法；
2) 熟悉了 TinyPerson 数据集的特性及其在小目标检测领域的重要意义；
3) 通过对比不同版本 YOLO 算法（YOLOv8/v9/v10/v11/v12）的检测性能，理解了目标检测算法的演进过程；
4) 学会了使用多种评价指标（如精确率、召回率、F1 分数、mAP 等）对目标检测算法进行全面评估；

5) 掌握了深度学习模型在特定场景下的训练、测试、评估和优化方法。
说明： 实验报告提交到超星学习通。