Course: Digital Image Processing

Project: Intelligent Proctoring Behavior Detection System Based on YOLOv8

Name:	Weihao Tao
Student ID:	1230016043
Major:	Computer Science
Date:	December 21, 2025

Abstract

The background of this project stems from an offline computer-based examination held on December 17th. In this scenario, the proctor explicitly prohibited the use of mobile communication devices and paper revision materials. However, relying solely on manual patrolling makes it difficult to effectively supervise all examinees throughout the entire session. Addressing this issue, this project developed a computer vision-based auxiliary proctoring system aimed at automatically detecting violations in the examination room.

The experiment simulated a real computer lab exam environment. A 90-second video was recorded covering four typical behavioral patterns: "Normal Answering," "Phone on Chest," "Hiding Phone," and "Checking Notes." Using a Python script, a uniform sampling strategy based on time steps was applied to construct a dataset containing 200 highly representative samples. The dataset was then used to train the YOLOv8 algorithm. Experiments show that the system achieved a Mean Average Precision (mAP50) of 98.7% on the validation set, demonstrating precise identification of phones and books. This provides an effective technical solution for cheating detection in offline computer-based exams.

Keywords: YOLOv8; Video Frame Extraction; Intelligent Proctoring; Behavior Analysis; Object Detection

Contents

1 Environment & Tools

This project adopts a ”Cloud Training + Local Deployment” hybrid architecture, with core algorithms implemented based on the PyTorch framework.

Table 1: Development Environment Configuration

Category	Item	Configuration Details
Training Platform	Hardware	Google Colab Pro (NVIDIA Tesla T4 GPU)
Inference Platform	Local Device	MacBook Pro (Apple M4 Pro, 12MP Camera)
Language	Programming	Python 3.12
Algorithm Framework	Deep Learning	Ultralytics YOLOv8 (v8.3.240)
Image Processing	Core Library	OpenCV (cv2)

2 Data Acquisition & Preprocessing

High-quality datasets are the prerequisite for training robust models. Unlike simple web crawling, this project adopts a **”Video Recording - Keyframe Extraction”** strategy to ensure scene consistency and behavioral diversity.

2.1 Video Scenario Design & Recording

To comprehensively cover typical behaviors during an exam, a 90-second simulation video was recorded. The video was performed strictly according to a predefined timeline, including positive behaviors (serious exam taking) and negative behaviors (violations). The timeline is designed as follows:

0s - 30s (Serious Exam): The examinee focuses on the paper, simulating a normal test-taking scenario. Used to train the model to recognize ”safe” scenes.

30s - 50s (Phone on Chest): The examinee holds a mobile phone openly at chest level. Simulates obvious violations in the exam room.

50s - 70s (Phone Hiding): The examinee attempts to hide the phone under the desk or behind books. Simulates covert cheating behaviors, increasing detection difficulty.

70s - 90s (Taking Notes): The examinee uses a pen and paper. Adds hand movement interference to prevent the model from misclassifying all hand actions as playing with a phone.

2.2 Python-Based Uniform Frame Sampling

To avoid image redundancy caused by direct video capturing (video frame rate is usually 30fps, where adjacent frames have minimal differences), a Python automation script was written. It employs a **uniform sampling strategy** to precisely extract 200 representative images from the 90-second video.

Sampling Algorithm Logic: Assuming the total number of frames in the video is N_{total} and the target number of images is $N_{target} = 200$, the sampling step size $Step$ is calculated as:

$$Step = \lfloor \frac{N_{total}}{N_{target}} \rfloor \tag{1}$$

The script automatically classifies and names the extracted images based on timestamps, greatly simplifying subsequent data cleaning.

```python
# Core sampling logic display
while True:
    ret, frame = cap.read()
    if not ret: break

    # Extract frame based on calculated step size
    if frame_count % step == 0:
        current_sec = frame_count / fps
        # Determine behavior label based on current second
        label = get_label_by_time(current_sec)
        img_name = f"{OUTPUT_FOLDER}/image_{saved_count:03d}.jpg"
        cv2.imwrite(img_name, frame)
```

Listing 1: Core Code for Automated Frame Extraction and Classification

2.3 Dataset Composition

After processing with the script, the composition of the original dataset is shown in the table below. This distribution ensures a relative balance between "normal behavior" and "violation behavior" samples.

Table 2: Original Dataset Category Distribution

Behavior Category	**Duration**	**Image Count**	**Percentage**
Serious Exam	30s	≈ 66	33%
Phone on Chest	20s	≈ 44	22%
Phone Hiding	20s	≈ 44	22%
Taking Notes	20s	≈ 44	22%
Total	**90s**	**200**	**100%**

2.4 Data Annotation

To build a highly robust detection model, this project simulated a real exam environment and collected 200 raw images, covering various typical postures such as normal answering, holding communication devices (phones), and consulting materials (books).

Face: Confirms the presence and identity of the examinee.

Phone: Core violation detection item.

Book: Auxiliary violation detection item.

Annotation Process: High-precision Bounding Box annotation was performed using the Roboflow platform.

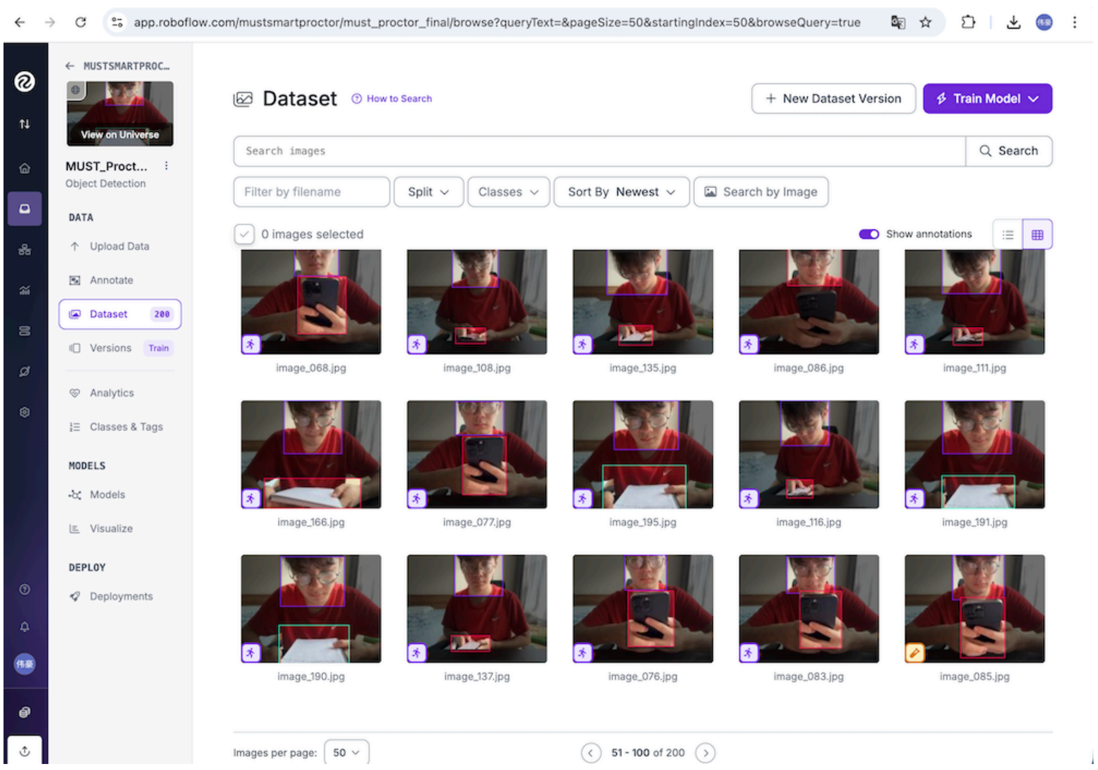


Figure 1: Roboflow Dataset Overview

2.5 Preprocessing & Augmentation

Addressing the potential overfitting risk due to the limited sample size (only 200 images) and the variable lighting and camera angles in real exam rooms, a standardized data preprocessing pipeline was designed. By introducing a series of image transformations and augmentation strategies, the project aims to enrich sample diversity, thereby improving the model's feature extraction capability and generalization level in complex environments. Specific implementation steps are as follows:

- **Resizing:** All raw video frames were uniformly resized to 640x640, the standard input size recommended by the YOLOv8 model.
- **Data Augmentation:** The Roboflow platform was used to perform a **3x Expansion** on the base dataset, extending the total sample size to 600 images. Specific augmentation methods include:
    1. **Brightness Perturbation:** Randomly applied ±15% brightness gain to simulate uneven lighting conditions between different seats in the computer room;
    2. **Noise Injection:** Added Gaussian noise to simulate the graininess of low-definition surveillance cameras in low-light environments;
    3. **Motion Blur:** Applied 2.5px Gaussian blur to simulate motion blur generated when examinees quickly hide phones or flip books.

- **Dataset Splitting:** Following standard evaluation protocols in deep learning, the augmented 600 images were strictly divided into Training, Validation, and Test sets based on a **7:2:1** ratio using random sampling.

2.6 Model Training

This project selected the YOLOv8n (Nano) model. This version has the smallest parameter count and the fastest inference speed, making it highly suitable for real-time deployment on ordinary PCs or mobile devices.

Training Parameter Settings:

Epochs: 50

Batch Size: 16

Image Size: 640

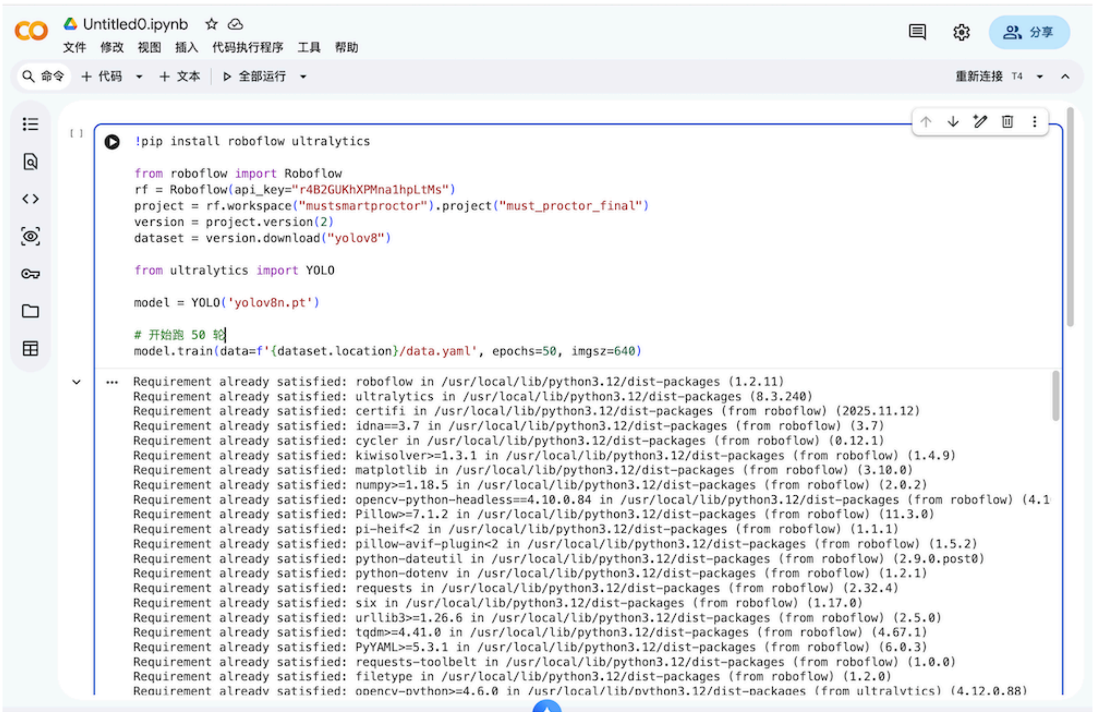
Optimizer: SGD

Core Training Code:

```python
from ultralytics import YOLO

model = YOLO('yolov8n.pt')

# Start Training
model.train(data=f'{dataset.location}/data.yaml', epochs=50, imgsz=640)
```



Untitled0.ipynb
文件 修改 视图 插入 代码执行程序 工具 帮助
分享
命令 + 代码 + 文本 全部运行
重新连接 T4

```
!pip install roboflow ultralytics

from roboflow import Roboflow
rf = Roboflow(api_key="r4B2GUKhXPMna1hpLtMs")
project = rf.workspace("mustsmartproctor").project("must_proctor_final")
version = project.version(2)
dataset = version.download("yolov8")

from ultralytics import YOLO

model = YOLO('yolov8n.pt')

# 开始跑 50 轮
model.train(data=f'{dataset.location}/data.yaml', epochs=50, imgsz=640)
```

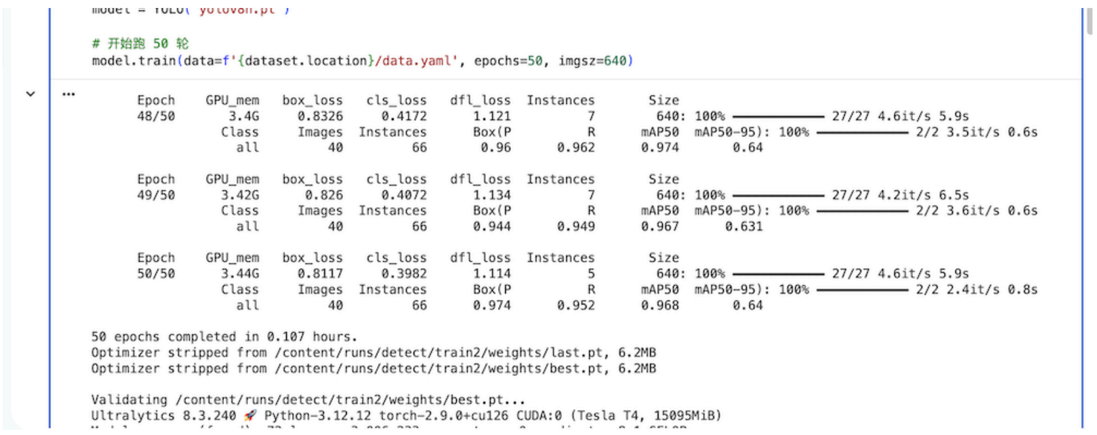
```
Requirement already satisfied: roboflow in /usr/local/lib/python3.12/dist-packages (1.2.11)
Requirement already satisfied: ultralytics in /usr/local/lib/python3.12/dist-packages (8.3.240)
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from roboflow) (2025.11.12)
Requirement already satisfied: idna==3.7 in /usr/local/lib/python3.12/dist-packages (from roboflow) (3.7)
Requirement already satisfied: cycler in /usr/local/lib/python3.12/dist-packages (from roboflow) (0.12.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from roboflow) (1.4.9)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (from roboflow) (3.10.0)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from roboflow) (2.0.2)
Requirement already satisfied: opencv-python-headless==4.10.0.84 in /usr/local/lib/python3.12/dist-packages (from roboflow) (4.1
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.12/dist-packages (from roboflow) (11.3.0)
Requirement already satisfied: pi-heif<2 in /usr/local/lib/python3.12/dist-packages (from roboflow) (1.1.1)
Requirement already satisfied: pillow-avif-plugin<2 in /usr/local/lib/python3.12/dist-packages (from roboflow) (1.5.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.12/dist-packages (from roboflow) (2.9.0.post0)
Requirement already satisfied: python-dotenv in /usr/local/lib/python3.12/dist-packages (from roboflow) (1.2.1)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from roboflow) (2.32.4)
Requirement already satisfied: six in /usr/local/lib/python3.12/dist-packages (from roboflow) (1.17.0)
Requirement already satisfied: urllib3>=1.26.6 in /usr/local/lib/python3.12/dist-packages (from roboflow) (2.5.0)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.12/dist-packages (from roboflow) (4.67.1)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.12/dist-packages (from roboflow) (6.0.3)
Requirement already satisfied: requests-toolbelt in /usr/local/lib/python3.12/dist-packages (from roboflow) (1.0.0)
Requirement already satisfied: filetype in /usr/local/lib/python3.12/dist-packages (from roboflow) (1.2.0)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.12/dist-packages (from ultralytics) (4.12.0.88)
```

Figure 2: Google Colab Training Environment Configuration

3 Results & Analysis

3.1 Training Convergence Analysis

The model training was conducted for 50 Epochs. Log data shows that by epochs 48-50, both Box Loss (localization loss) and Cls Loss (classification loss) showed a steady downward trend and converged. GPU memory usage stabilized at 3.4GB, indicating a stable and efficient training process.



```
# 开始跑 50 轮
model.train(data=f'{dataset.location}/data.yaml', epochs=50, imgsz=640)
```

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      48/50       3.4G     0.8326     0.4172      1.121          7        640: 100% ━━━━━━━━━━━━ 27/27 4.6it/s 5.9s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 2/2 3.5it/s 0.6s
                   all         40         66       0.96      0.962      0.974       0.64

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      49/50      3.42G      0.826     0.4072      1.134          7        640: 100% ━━━━━━━━━━━━ 27/27 4.2it/s 6.5s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 2/2 3.6it/s 0.6s
                   all         40         66      0.944      0.949      0.967      0.631

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      50/50      3.44G     0.8117     0.3982      1.114          5        640: 100% ━━━━━━━━━━━━ 27/27 4.6it/s 5.9s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 2/2 2.4it/s 0.8s
                   all         40         66      0.974      0.952      0.968       0.64

50 epochs completed in 0.107 hours.
Optimizer stripped from /content/runs/detect/train2/weights/last.pt, 6.2MB
Optimizer stripped from /content/runs/detect/train2/weights/best.pt, 6.2MB

Validating /content/runs/detect/train2/weights/best.pt...
Ultralytics 8.3.240 🚀 Python-3.12.12 torch-2.9.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
```

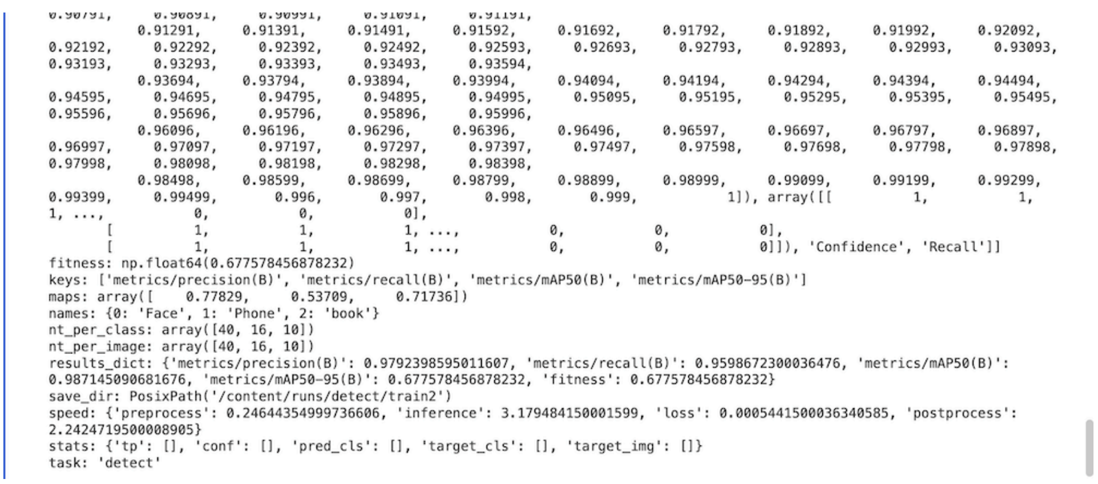
Figure 3: Training Logs and Loss Function Convergence

3.2 Evaluation Metrics

The model performed excellently on the validation set. The core metrics are shown in the table below:

Table 3: Model Evaluation Metrics

Class	mAP50	Evaluation Conclusion
All	**98.7%**	Excellent overall performance
Face	99.5%	Almost no missed detections
Phone	99.5%	Precise identification of key cheating items
Book	97.2%	Good multi-scale detection capability



```
0.90791,    0.90891,    0.90991,    0.91091,    0.91191,
        0.91291,    0.91391,    0.91491,    0.91592,    0.91692,    0.91792,    0.91892,    0.91992,    0.92092,
0.92192,    0.92292,    0.92392,    0.92492,    0.92593,    0.92693,    0.92793,    0.92893,    0.92993,    0.93093,
0.93193,    0.93293,    0.93393,    0.93493,    0.93594,
        0.93694,    0.93794,    0.93894,    0.93994,    0.94094,    0.94194,    0.94294,    0.94394,    0.94494,
0.94595,    0.94695,    0.94795,    0.94895,    0.94995,    0.95095,    0.95195,    0.95295,    0.95395,    0.95495,
0.95596,    0.95696,    0.95796,    0.95896,    0.95996,
        0.96096,    0.96196,    0.96296,    0.96396,    0.96496,    0.96597,    0.96697,    0.96797,    0.96897,
0.96997,    0.97097,    0.97197,    0.97297,    0.97397,    0.97497,    0.97598,    0.97698,    0.97798,    0.97898,
0.97998,    0.98098,    0.98198,    0.98298,    0.98398,
        0.98498,    0.98599,    0.98699,    0.98799,    0.98899,    0.98999,    0.99099,    0.99199,    0.99299,
0.99399,    0.99499,      0.996,      0.997,      0.998,      0.999,          1]), array([[          1,          1,
1, ...,           0,          0,          0],
      [           1,          1,          1, ...,          0,          0,          0],
      [           1,          1,          1, ...,          0,          0,          0]]), 'Confidence', 'Recall']]
fitness: np.float64(0.677578456878232)
keys: ['metrics/precision(B)', 'metrics/recall(B)', 'metrics/mAP50(B)', 'metrics/mAP50-95(B)']
maps: array([    0.77829,     0.53709,     0.71736])
names: {0: 'Face', 1: 'Phone', 2: 'book'}
nt_per_class: array([40, 16, 10])
nt_per_image: array([40, 16, 10])
results_dict: {'metrics/precision(B)': 0.9792398595011607, 'metrics/recall(B)': 0.9598672300036476, 'metrics/mAP50(B)':
0.987145090681676, 'metrics/mAP50-95(B)': 0.677578456878232, 'fitness': 0.677578456878232}
save_dir: PosixPath('/content/runs/detect/train2')
speed: {'preprocess': 0.24644354999736606, 'inference': 3.179484150001599, 'loss': 0.0005441500036340585, 'postprocess':
2.2424719500008905}
stats: {'tp': [], 'conf': [], 'pred_cls': [], 'target_cls': [], 'target_img': []}
task: 'detect'
```

Figure 4: Detailed Evaluation Metrics on Validation Set

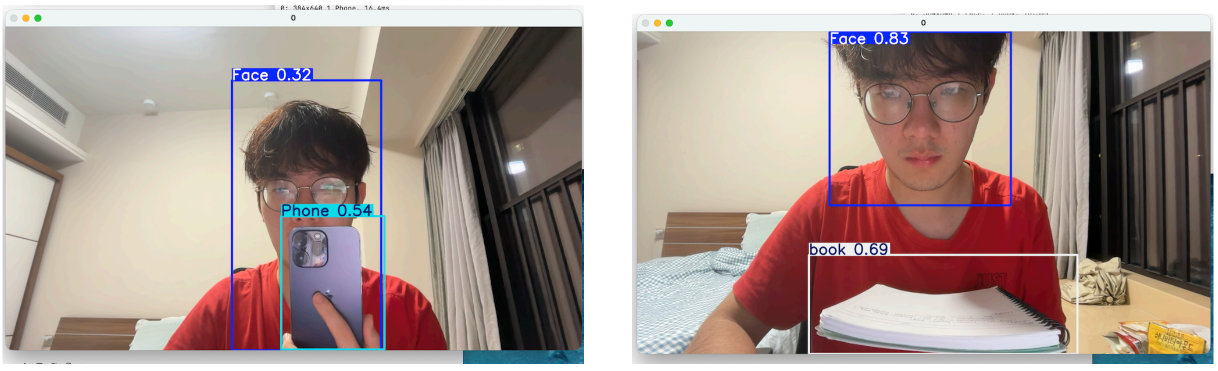
3.3 Real-time Inference Test

The best weight file (best.pt) generated from training was deployed to the local MacBook environment, invoking the camera for real-time stream detection.

Test Command:

```
yolo predict model=best.pt source=0 show=True
```

Test Results: As shown in the figure below, the model can accurately bound faces, phones, and books under complex backgrounds and different lighting conditions. The confidence scores are generally higher than 0.8, and the display is smooth, meeting the real-time requirements of online proctoring.



(a) Scenario A: Handheld Phone Detection (b) Scenario B: Checking Book Detection

Figure 5: Local Real-time Detection Demonstration

4 Project Innovations

This project successfully built a lightweight intelligent proctoring system based on YOLOv8. Through data augmentation and model fine-tuning, high-precision recognition of cheating behaviors such as Phone (99.5%) and Book (97.2%) usage was achieved.

Full-Process Closed Loop:: Independently completed the full development link from data acquisition, cleaning, and annotation to model training and deployment.

Scenario-Driven Dataset Construction: Directly addressing the ”blind spots” in manual proctoring during the recent offline computer-based exams (e.g., covertly using phones under desks), this project constructed a specific dataset. Unlike generic object detection datasets, this dataset focuses on capturing ”hard-to-detect” violation behaviors specific to the computer lab environment, significantly improving the model’s practical utility in real-world scenarios.

Environment-Aware Robustness Enhancement: Targeting the specific challenges of offline computer labs (e.g., dim lighting, low-resolution webcams), a specific data augmentation pipeline was constructed. By injecting Gaussian noise and motion blur, the model’s robustness against environmental interference and motion artifacts was effectively improved.

Feasibility Verification of Edge Deployment: The project successfully validated the performance of the lightweight YOLOv8n model on consumer-grade hardware (Apple M4 Pro). It demonstrated that high-precision detection (mAP 98.7%) can be achieved with low computational cost, proving the system’s potential for large-scale, low-cost deployment in educational institutions.