1. Background

OpenCV 是一个用于图像处理和计算机视觉的开源库，常用于人脸识别、目标检测、图像变换等任务。

RKNN（*Rockchip Neural Network*）是瑞芯微（*Rockchip*）推出的神经网络推理引擎，专门为其 AI 芯片（如 RK3399Pro、RK1808、RK3588 等）优化。RKNN 支持多种主流深度学习框架（如 TensorFlow、Caffe、ONNX、PyTorch）模型的转换和高效推理，常用于边缘 AI 设备的本地推理加速。

二者关系与区别：

- *OpenCV* 主要用于传统图像处理和部分深度学习推理（如 DNN 模块），但在嵌入式 AI 芯片上速度有限。
- *RKNN* 专注于在 Rockchip 芯片上高效运行深度学习模型，通常用于加速神经网络推理（如人脸识别、目标检测等 AI 功能）。
- 在 *Rockchip* 平台上，常见做法是用 OpenCV 采集和预处理图像，然后用 RKNN 进行神经网络推理，最后再用 OpenCV 进行后处理和显示。

总结：
OpenCV 负责“图像处理”，RKNN 负责“AI 推理加速”，两者可以结合使用，尤其适合 RK3588 这类带有 NPU 的国产 AI 芯片平台。

2. 项目流程

要在 *RK3588* 等瑞芯微平台上用 **RKNN + OpenCV** 实现人脸识别，常见流程如下：

- **1. 用 OpenCV 采集摄像头图像**
- **2. 用 OpenCV 进行人脸检测(可选，或用 RKNN 检测)**
- **3. 将检测到的人脸区域送入 RKNN 进行人脸识别推理**
- **4. 根据 RKNN 输出结果判断身份**
- **5. 用 OpenCV 显示结果或做后处理**

下面是一个简化版的流程代码（假设你已经有 RKNN 模型文件和 RKNN API 环境，且模型为人脸识别模型）：

```python
import cv2
from rknnlite.api import RKNN

# 1. 加载 RKNN 模型
rknn = RKNN()
rknn.load_rknn('your_face_recognition_model.rknn')
rknn.init_runtime()

# 2. 打开摄像头
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # 3. (可选) OpenCV 检测人脸区域
    # 这里假设你用 OpenCV 的 haarcascade 或 dnn 检测人脸
    # face = frame[y:y+h, x:x+w]  # 截取人脸区域

    # 4. 预处理（如 resize、BGR2RGB、归一化等，按你的模型要求）
    input_img = cv2.resize(frame, (112, 112))  # 以112x112为例
    input_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2RGB)
    input_img = input_img.astype('float32') / 255.0
    input_img = input_img[np.newaxis, :]  # batch 1

    # 5. RKNN 推理
    outputs = rknn.inference(inputs=[input_img])

    # 6. 根据 outputs 做人脸识别（如比对特征向量、分类等）
    # result = your_postprocess(outputs)

    # 7. 显示
    cv2.imshow('face', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
rknn.release()
```

注意事项

- **1. *RKNN* 模型需提前用 *rknn-toolkit/lite* 转换好（如 *ArcFace*、*MobileFaceNet* 等）。**
- **2. *OpenCV* 只做图像采集、预处理和显示，推理用 *RKNN*。**
- **3. 你可以用 *OpenCV* 检测人脸框，也可以用 *RKNN* 检测+识别一体模型。**
- **4. 代码需根据你的模型输入尺寸、预处理方式、输出格式调整。**
- **5. 如需完整工程模板或和 *PyQt/QML* 结合的界面方案，请告知你的具体需求！**

3. 项目模板

你需要一个“**RKNN+OpenCV**人脸识别”完整工程模板，并且界面用QML实现，Python负责业务逻辑和推理。下面是推荐的工程结构和关键代码模板，适合RK3588等瑞芯微平台：

3.1 工程结构建议

```
project_root/
│
├─ page/
│   ├─ main.py               # PyQt/PySide主入口，加载QML
│   ├─ face_infer.py         # RKNN推理与OpenCV采集逻辑
│   ├─ shared_data.py        # 全局状态（如 operation_status）
│   └─ LogOn.qml             # QML登录界面
│
├─ picture/
│   └─ background.jpg        # 背景图片等
│
├─ model/
│   └─ face_model.rknn       # RKNN模型
│
└─ requirements.txt
```

3.2 QML 登录界面（page/LogOn.qml）

已为你生成，核心点如下：

- operationStatus 通过 Python 传递给 QML，控制 Loader 区域显示“人脸识别”或“登录成功”。
- 可扩展为密码登录等。

3.3 Python 端：RKNN+OpenCV推理与QML交互

shared_data.py

```
# page/shared_data.py
operation_status = "face"  # "face" or "success"
```

face_infer.py

```python
# page/face_infer.py
import cv2
import numpy as np
from rknnlite.api import RKNN
import shared_data

class FaceRecognizer:
    def __init__(self, model_path):
        self.rknn = RKNN()
        self.rknn.load_rknn(model_path)
        self.rknn.init_runtime()
        self.cap = cv2.VideoCapture(0)

    def infer_once(self):
        ret, frame = self.cap.read()
        if not ret:
            return None
        # 预处理
        img = cv2.resize(frame, (112, 112))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = img.astype(np.float32) / 255.0
        img = np.expand_dims(img, 0)
        # 推理
        outputs = self.rknn.inference(inputs=[img])
        # 后处理（假设输出为特征向量，实际需根据模型调整）
        # result = your_postprocess(outputs)
        # 假设识别成功
        shared_data.operation_status = "success"
        return frame

    def release(self):
        self.cap.release()
        self.rknn.release()
```

main.py (加载QML并与QML通信)

```python
# page/main.py
from PyQt5.QtCore import QObject, pyqtSignal, pyqtProperty, QTimer
from PyQt5.QtQml import QQmlApplicationEngine
from PyQt5.QtWidgets import QApplication
import sys
import shared_data
from face_infer import FaceRecognizer

class Backend(QObject):
    operationStatusChanged = pyqtSignal()

    def __init__(self):
        super().__init__()
        self._operation_status = shared_data.operation_status

    @pyqtProperty(str, notify=operationStatusChanged)
    def operationStatus(self):
        return shared_data.operation_status

    def update_status(self):
        if self._operation_status != shared_data.operation_status:
            self._operation_status = shared_data.operation_status
            self.operationStatusChanged.emit()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    engine = QQmlApplicationEngine()
    backend = Backend()
    engine.rootContext().setContextProperty("backend", backend)
    engine.load("page/LogOn.qml")

    # 启动定时器不断推理并更新状态
    recognizer = FaceRecognizer("model/face_model.rknn")
    timer = QTimer()
    timer.timeout.connect(lambda: [recognizer.infer_once(), backend.update_status()])
    timer.start(100)  # 10帧每秒

    sys.exit(app.exec_())
```

3.4 QML 端与 Python 交互（LogOn.qml 关键片段）

```
// page/LogOn.qml
Item {
    // ...
    Loader {
        id: faceLoader
        anchors.fill: parent
        sourceComponent: backend.operationStatus !== "success" ? videoComponent : labelComponent
    }
    // ...
}
```

通过 setContextProperty("backend", backend) 实现 QML 访问 Python 的 operationStatus 属性。

3.5 requirements.txt

```
PyQt5
rknnlite
opencv-python
```

3.6 说明

你只需补充模型推理的后处理部分（如特征比对、阈值判断等）。
QML 只负责界面，所有业务逻辑和状态切换都在 Python。
若需密码登录等功能，可在 QML 里扩展 Loader 区域和状态切换。
如需更详细的某一部分代码或功能扩展，请告知！