HAR 테스트베이스앱장기구축전략: 모듈형실험플랫폼설계

1. 시스템아키텍처설계원칙

1.1 핵심설계철학

1.1.1 모듈러플러그인아키텍처 HAR 테스트베이스앱의근간이되는설계철학은각기능을독립적이고교체가능한플러그인모듈로구성하는것입니다. 이는 "독창적인 HAR 알고리즘을버전별라이브러리로축적" 하고 "추후정식서비스에사용" 하는핵심목표를달성하기위한필수조건입니다. 포즈추정, 객체추적, 행동분류, 경보시스템등각단계가명확한인터페이스를통해상호작용하도록설계하면, 특정알고리즘의변경이전체시스템에미치는영향을최소화할수있습니다.

구체적인구현방식으로는 **Python 의추상기본클래스 (ABC, Abstract Base Class)** 를활용하여플러그인인터페이스를정의합니다. 예를들어, `PoseEstimator` 추상클래스는 `estimate(frame)-> Skeleton` 메서드를강제하며, `YOLOv8PoseEstimator`, `MediaPipePoseEstimator`, `OpenVINOPoseEstimator` 등의구체적구현체가이를상속받습니다. 마찬가지로 `Tracker`, `ActionClassifier`, `AlertNotifier` 등의인터페이스도정의하여, 웹앱의설정패널에서드롭다운메뉴로알고리즘을즉시교체할수있게합니다. 이러한플러그인시스템은 A/B 테스트를용이하게하며, 새로운알고리즘개발시기존인프라를재사용하여개발주기를단축시킵니다.

동적모듈로딩메커니즘도핵심기능으로, 개발자가새로운알고리즘을미리정의된인터페이스를구현하는 Python 클래스로패키징하여특정디렉토리에배치하면, 웹앱이자동으로인식하고실험옵션에추가할수있어야합니다. 이과정에서코드수정이나앱재시작없이새로운알고리즘을통합할수있는핫플러그기능은연구생산성을획기적으로향상시킵니다.

1.1.2 하드웨어추상화계층 (HAL) M1 Pro 에서 NVIDIA Jetson Orin Super, 그리고 Windows PC 까지지원하는다양한하드웨어환경을고려할때, 하드웨어추상화계층 (HAL) 의도입은필수적입니다. HAL 은상위레벨의애플리케이션코드가특정하드웨어세부사항에의존하지않도록하며, 동일한 Python 코드가 Apple Silicon 의 MPS, NVIDIA 의 CUDA/TensorRT, Intel 의 OpenVINO, 순수 CPU 등다양한백엔드에서실행될수있게합니다.

HAL 의핵심구성요소는 `InferenceBackend` **추상클래스**입니다. 이클래스는 `initialize()`, `inference(model, input)`, `get_optimal_batch_size()`, `memory_stats()` 등의메서드를제공하며, 구체적구현체인 `MPSBackend`, `CUDABackend`, `TensorRTBackend`, `CPUBackend`, `OpenVINOBackend`가하드웨어특화된최적화를담당합니다. M1 Pro 환경에서는 PyTorch 의 `mps` 디바이스를활용하되, YOLOv8-Pose 의경우공식문서에따르면 MPS 에서훈련시알려진버그가있어추론단계에서만 MPS 를사용하고, 안정성이우선될때는 CPU 폴백을제공하는것이바람직합니다. 반면 Jetson 환경에서는 TensorRT 최적화된엔진을로드하여 FP16 또는 INT8 정밀도로추론하는경로가필요합니다.

자동백엔드선택및폴백메커니즘도 HAL 의중요한기능입니다. 시스템초기화시사용가능한하드웨어를자동탐지하여최적의백엔드를선택하고, 지정된백엔드로딩실패시자동으로하위호환가능한백엔드로전환하는기능을제공합니다. 예를들어 Jetson 환경에서 TensorRT 엔진로딩에실패할경우 ONNX Runtime 기반의 CUDA 백엔드로, 그마저실패할경우순수 CPU 백엔드로우아하게성능을저하시키면서도서비스를유지하는것입니다.

1.1.3 알고리즘버전관리및라이브러리화 "독창적인 HAR 알고리즘을버전별라이브러리로축적" 하는것은단순한코드저장을넘어, 체계적인버전관리와재현가능한실험환경을의미합니다. 각알고리즘버전은 **Semantic Versioning(semver)** 을따르며, 메이저. 마이너. 패치형식으로버전을관리합니다. 예를들어, YOLOv8-Pose 통합모듈은 `har_pose_yolov8 v1.2.3`과같이명시되며, 각버전은특정모델가중치파일, 전처리파라미터, 후처리로직의조합을고유하게식별합니다.

버전관리의실무적구현으로는, 각알고리즘버전을독립적인 Git 태그로관리하고, 해당버전의메타데이터를 JSON 형식으로기록합니다. 메타데이터에는모델아키텍처, 훈련데이터셋, 입력해상도, FPS 벤치마크결과, 검증정확도등이포함됩니다. 웹앱의 "알고리즘선택" 인터페이스에서는이러한메타데이터를시각적으로표현하여, 사용자가성능-정확도트레이드오프를정보에기반하여결정할수있게합니다. 또한, 동일한입력영상에대해여러버전의알고리즘을병렬실행하여결과를비교하는 **"알고리즘경쟁 (Algorithm Competition)" 모드**를제공하면, 새로운알고리즘의도입결정에객관적인근거를제공할수있습니다.

라이브러리화의최종목표는 `pip install har-core`와같은명령으로핵심기능을설치하고, `from har_core import YOLOv8PoseEstimator, LSTMActionClassifier`와같이임포트하여사용할수있는수준입니다. 이를위해 `setup.py` 또는

`pyproject.toml`을통한패키지메타데이터정의, PyPI 배포자동화파이프라인구축, 그리고 Sphinx 기반 API 문서화가필요합니다. 특히 "추후정식서비스에사용하는것이목표" 라는점을고려할때, 라이선스명확화 (Apache 2.0 또는 MIT 권장)와보안취약점스캔을 CI/CD 파이프라인에통합하는것이장기적으로필수적입니다.

1.1.4 확장가능한멀티카메라파이프라인 초기단일카메라입력에서 2-4 대카메라확장을염두에둔파이프라인설계는시스템의생명력을결정합니다. 단순히 4 개의독립적인싱글카메라프로세스를병렬실행하는것이아니라, 공유리소스관리와부하분산을고려한통합파이프라인이필요합니다.

핵심설계패턴으로는 **생산자-소비자모델기반의비동기프레임처리**를권장합니다. 각카메라입력은별도의스레드또는프로세스에서 GIL(Global Interpreter Lock) 을우회하는방식으로프레임을획득하여공유버퍼 (Queue) 에배치합니다. 중앙처리엔진은이버퍼로부터프레임을소비하여추론을수행하고, 결과를다시공유상태저장소에기록합니다. Python 의 `multiprocessing.Queue`와 `ProcessPoolExecutor`를활용하거나, 더고성능이필요한경우 ZeroMQ 나 Redis 를메시지브로커로도입할수있습니다.

GPU 메모리관리는멀티카메라환경에서의핵심과제입니다. 4 개의 1080p 스트림을동시에처리할때, 각스트림마다별도의모델인스턴스를로드하는것은현실적이지않습니다. 대안으로 **배치추론 (Batch Inference)** 을활용하여여러프레임을하나의배치로묶어단일 forward pass 로처리하거나, 최소한모델가중치를공유하면서입력텐서만교체하는방식을구현해야합니다. M1 Pro 의 16GB 통합메모리와 Jetson Orin Super 의메모리제약을고려할때, 4 스트림배치처리는충분히실현가능하며, 이를위한파이프라인구조는초기설계단계부터반영되어야합니다.

설계원칙	핵심목표	구현방식
모듈러플러그인	알고리즘교체용이성	ABC 기반인터페이스, 동적로딩
HAL	하드웨어독립성	`InferenceBackend` 추상화, 자동선택
버전관리	실험재현성	Semantic Versioning, 메타데이터기록
멀티카메라	수평확장성	생산자-소비자패턴, 배치추론

1.2 단계별마이그레이션전략

1.2.1 Phase 1: M1 Pro 기반단일카메라프로토타입 첫번째단계의목표는핵심알고리즘파이프라인의검증과개발환경확립입니다. M1 Pro MacBook 의자원을최대한활용하여로컬에서실시간추론이가능한구조를구축하며, 이과정에서 HAL 의초기버전과기본적인웹인터페이스를개발합니다.

구체적인하드웨어활용전략은다음과같습니다. **CPU** 는데이터전처리및웹서버구동을담당하며, Python 의싱글스레드성능이우수한특성을활용합니다. **GPU/Neural Engine** 은 Metal Performance Shaders(MPS) 를통해딥러닝모델가속을수행하며, PyTorch 2.1 이상에서 `device='mps'` 설정으로네이티브지원됩니다. **입력장치**로는기본적으로 MacBook Facetime HD Camera 를테스트용으로사용하고, 확장을위해 USB 웹캠 (로지텍 C920 등) 또는 RTSP 지원 IP 카메라를연결하여실제 CCTV 환경을모사합니다.

이단계에서반드시해결해야할기술적과제는 **M1 환경에서의안정적인실시간추론**입니다. YOLOv8-Pose 의 CPU 모드에서 "Illegal instruction: 4" 오류가발생할수있으며, 이는 MPS 가속을명시적으로지정하거나 ONNX Runtime 으로의변환을통해우회해야합니다. 또한 30fps 처리가불가능할경우를대비한프레임드롭로직과, 추론지연으로인한버퍼오버플로우방지메커니즘을구현해야합니다. 개발산출물로는 Streamlit 기반의기본대시보드, YOLOv8-Pose 를활용한실시간스켈레톤추출, 그리고규칙기반의간단한행동감지 (쓰러짐등) 가포함됩니다. 이단계의성공기준은 1080p 입력에서 15fps 이상의안정적인처리와, 웹인터페이스를통한실시간모니터링가능여부입니다.

1.2.2 Phase 2: 멀티카메라확장 (2-4 대 USB/RTSP) 두번째단계에서는단일카메라파이프라인을 2-4 대카메라로확장하면서, 리소스경쟁과동기화문제를해결합니다. USB 웹캠과 RTSP IP 카메라를혼합하여사용할수있는유연한입력관리시스템을구축하며, DeepSORT 기반다중객체추적을통합합니다.

핵심기술과제는 **다중입력스트림의효율적관리**입니다. USB 웹캠과 RTSP IP 카메라는근본적으로다른 I/O 특성을가지며, 전자는 DirectShow/V4L2 를통해후자는 FFmpeg/GStreamer 를통해접근됩니다. HAL 의입력장치모듈은이러한차이를추상화하여상위레이어에통일된프레임인터페이스를제공해야합니다. 특히 RTSP 스트림의네트워크지연과패킷손실에대한견고한처리, 그리고 USB 대역폭제한으로인한다중웹캠의해상도/프레임레이트조정이필요합니다.

이단계에서도입되는핵심알고리즘은 **다중객체추적 (Multi-Object Tracking, MOT)** 입니다. 단일프레임내에서여러사람을검출하는것을넘어, 시간에걸쳐각개인의 ID 를일관되게유지하는것이필수적입니다. DeepSORT 나 ByteTrack 알고리즘을통합하며, 이를위해외관특징임베딩네트워크 (ReID) 의경량화버전을함께배포합니다. 또한 LSTM 기반의행동분류를규칙기반시스템과통합하여, 더복잡한행동패턴 (걷기, 뛰기, 주먹질등) 을인식합니다. 웹인터페이스는멀티뷰레이아웃 (2x2 그리드등) 으로확장되며, 각카메라별독립적인설정 (ROI, 감도임계값등) 조정기능이추가됩니다. 성능목표는 4 대 720p 카메라입력에서평균 10fps 이상의처리입니다.

1.2.3 Phase 3: NVIDIA Jetson Orin Super 이식 세번째단계는검증된시스템을 NVIDIA Jetson Orin Super 로이식하여엣지컴퓨팅환경에서의운영을입증합니다. 이단계의핵심은 TensorRT 를활용한모델최적화와 GStreamer 기반하드웨어가속비디오파이프라인의구현입니다.

TensorRT 최적화는필수적인변환과정입니다. PyTorch 모델 (YOLOv8-Pose, LSTM) 을 ONNX 형식으로내보낸후, TensorRT 의 FP16 또는 INT8 양자화를적용하여 Jetson 의 Tensor Core 를활용합니다. 검색결과에따르면, Jetson Orin Nano 에서 YOLOv8-Pose 는 TensorRT 변환후 30fps 이상의성능을달성할수있습니다. 이과정에서동적배치크기, 최적화된플러그인, 그리고메모리풀기반의효율적인할당전략이필요합니다.

GStreamer 기반카메라입력파이프라인도핵심최적화수단입니다. Jetson 의 JetPack SDK 는 `nvarguscamerasrc` (CSI 카메라) 와 `nvv4l2decoder`(하드웨어디코딩) 를제공하며, 이를활용한 zero-copy 메모리경로구현으로 CPU-GPU 간불필요한데이터복사를제거합니다. RTSP 입력의경우 `rtspsrc` → `rtph264depay` → `nvv4l2decoder` → `nvvidconv` → `appsink` 파이프라인을구성하여하드웨어가속된전처리를달성합니다.

이단계에서검증해야할사항은개발환경과동일한정확도를유지하면서도, Jetson 의제한된전력예산 (15W~60W) 내에서지속가능한처리량을달성하는것입니다. 열스로틀링에대한모니터링과, 필요시클럭주파수조정을통한성능-전력트레이드오프관리가포함됩니다. Docker 컨테이너화도완료하여, 개발환경과타겟환경의불일치로인한 “it works on my machine” 문제를방지합니다.

1.2.4 Phase 4: Windows 네이티브앱전환 최종단계는 Python 기반웹앱을 Windows 네이티브애플리케이션으로전환하여상용배포를준비합니다. 이전환의동기는 Python 런타임의존성제거, 더나은시스템통합, 그리고최종사용자에게친숙한설치경험제공입니다.

기술적접근방식으로는두가지를고려할수있습니다. 첫째, **PyInstaller 기반의실행파일패키징**으로 Python 런타임과모든종속성을단일 `.exe`로묶는방식입니다. 이는개발속도가빠르지만, 바이러스백신의오탐지문제와큰파일크기가단점입니다. 둘째, **C++/C# 기반의네이티브재구현**으로 ONNX Runtime 이나 TensorRT 의 C++ API 를직접활용하는방식입니다. 이는최적의성능과작은배포크기를달성하지만, 상당한개발리소스가필요합니다.

UI 프레임워크선택으로는 **PyQt6/PySide6** 또는 **Flutter** 를검토합니다. PyQt6 는 Python 코드베이스의재사용성이높고, PyInstaller 를통해단일실행파일을생성할수있습니다. Flutter 는현대적인크로스플랫폼 UI 와우수한성능을제공하며, 향후 macOS, Linux, 모바일로의확장이용이합니다. DirectShow 를통한카메라입력통합과, GPU 가속을위한 DirectML 또는 CUDA 통합도필수적입니다. 설치프로그램은 WiX Toolset 또는 NSIS 를활용하여전문적인인스톨러를구성하며, 자동업데이트메커니즘도통합합니다. 이단계의완료는 “정식서비스” 로의전환을위한마일스톤이됩니다.

단계	기간	핵심목표	주요산출물
Phase 1	4 주	단일카메라파이프라인검증	MPS 실시간추론, Streamlit 대시보드, 규칙 기반감지
Phase 2	4 주	멀티카메라확장및추적통합	2-4 카메라동시입력, DeepSORT, LSTM 분류
Phase 3	4 주	엣지최적화및배포	TensorRT 변환, GStreamer 파이프라인, Docker 컨테이너
Phase 4	4 주	상용네이티브앱전환	PyQt/Flutter UI, 설치형 패키지, 자동업데이트

2. 하드웨어추상화계층 (HAL)

2.1 입력장치관리모듈

2.1.1 카메라소스유형별핸들러 HAR 테스트베이스앱은다양한카메라소스를유연하게지원해야하며, 각소스유형은고유한 초기화파라미터, 연결관리전략, 그리고오류복구메커니즘을요구합니다. 이를위해소스유형별전문핸들러를설계하고, 공통의 `CameraSource` 추상인터페이스뒤에추상화합니다.

USB 웹캠핸들러는로지텍 C920 과같은표준 UVC 장치를지원합니다. OpenCV 의 `cv2.VideoCapture(index)` 인터 페이스를기반으로하되, V4L2(Linux), DirectShow(Windows), AVFoundation(macOS) 등의플랫폼별백엔드를자 동선택하여최적의성능을확보합니다. 해상도 (1920x1080 부터 640x480 까지) 와프레임레이트 (30fps, 60fps) 의동적설 정을지원하며, `cv2.CAP_PROP_BUFFERSIZE` 조정을통해지연시간을최소화합니다. USB 대역폭제한으로인해단일컨트롤러 에 2 대이상의고해상도웹캠을연결하면성능저하가발생할수있으므로, 이경우해상도를 1280x720 으로낮추거나별도의 USB 컨트롤러를사용하는동적품질적응기능을구현합니다.

RTSP IP 카메라핸들러는 ONVIF 프로토콜을지원하는네트워크카메라를대상으로합니다. RTSP URL 형식은일반적 으로 `rtsp://username:password@ip:port/stream_path`를따르며, 인증정보의안전한관리 (환경변수또는암호화된설정 파일) 가필요합니다. OpenCV 의 `cv2.VideoCapture`는 RTSP 를기본지원하지만, GStreamer 백엔드를사용하면더세 밀한제어가가능합니다. `rtspsrc` 요소에 `latency=0` 옵션을설정하여지연시간을최소화하거나, `protocols=tcp`를지정하여 UDP 패킷손실에민감한환경에서안정성을높일수있습니다. 멀티캐스트 RTSP 스트림의경우네트워크부하분산효과를얻을수 있습니다.

MacBook Facetime HD 핸들러는개발및테스트단계에서의편의성을위해제공됩니다. AVFoundation 프레임워크를 통해접근하며, iOS/macOS 특화기능인자동노출조정과초점조정을비활성화하여일관된이미지품질을유지할수있는옵션을제 공합니다.

GStreamer 파이프라인핸들러는 Jetson 환경에서의최적화된영상입력을담당합니다. `nvarguscamerasrc`(CSI 카메라), `v4l2src`(USB 카메라), `rtspsrc`(IP 카메라) 등의소스요소와 `nvvidconv`, `nvjpegenc` 등의하드웨어가속변환요소를조합 하여, zero-copy 메모리관리와최고의처리성능을달성합니다. 사용자가직접파이프라인문자열을입력할수있는고급모드와, 미 리정의된템플릿을선택하는간편모드를모두지원합니다.

카메라유형	백엔드라이브러리	주요설정파라미터	특수처리
USB 웹캠	OpenCV VideoCapture	해상도, FPS, 포맷 (YUY2/MJPEG)	V4L2/DirectShow 백엔드선택, 버퍼크기조정
RTSP IP 카메라	OpenCV + FFmpeg/ GStreamer	URL, 인증, 프로토콜 (TCP/UDP)	ONVIF 자동검색, jitterbuffer, 재연결로직
Facetime HD	AVFoundation	해상도, 자동노출비활성화	macOS 권한처리, 테스트용최적화설정
Jetson CSI/USB/RTSP	GStreamer	파이프라인문자열또는템플릿	NVMM 메모리타입, zero-copy, 하드웨어가속

2.1.2 해상도및프레임레이트동적제어　실시간 HAR 시스템의성능은입력영상의해상도와프레임레이트에직접적으로영향을받습니다. 높은해상도는더정확한포즈추정을가능하게하지만연산량이급증하고, 높은프레임레이트는더부드러운시계열데이터를제공하되처리파이프라인에부하를줍니다. 따라서상황에따라이러한파라미터를동적으로조정할수있는메커니즘이필요합니다.

적응형품질제어 (Adaptive Quality Control) 의기본원리는처리파이프라인의실제처리량을모니터링하고, 목표프레임레이트 (예: 15fps) 를유지하기위해입력품질을조정하는것입니다. 구체적으로는다음과같은알고리즘을구현합니다: (1) 최근 N 초간의실제출력 FPS 를측정, (2) 목표 FPS 대비 20% 이상낮을경우입력해상도를 0.75 배로축소, (3) 여전히부족할경우 0.5 배로추가축소, (4) 반대로여유가있을경우점진적품질복원. 이과정에서이미지의종횡비를유지하고, Lanczos 또는 Bicubic 보간을사용한고품질다운스케일링을적용합니다.

프레임스킵전략은해상도축소만으로부족한경우에사용됩니다. 입력스트림에서모든프레임을처리하는대신, 예를들어 30fps 입력에서 2 프레임마다 1 프레임만처리 (15fps 효과) 하거나, 3 프레임마다 1 프레임만처리 (10fps 효과) 하는방식입니다. 중요한것은스킵된프레임에대해서도객체추적을통해궤적의연속성을유지하는것이며, 이를위해 Kalman 필터기반의예측을활용합니다.

ROI(Region of Interest) 기반적응은고급전략으로, 장면의특정영역만고해상도로처리하고나머지는저해상도로처리하는방식입니다. 예를들어움직임이감지된영역만 1080p 로처리하고, 정적인배경은 480p 로처리하여전체대역폭을절약합니다. 이를위해배경차분법이나가벼운움직임검출을전처리단계에도입합니다. 웹앱의실험인터페이스에서는이러한파라미터를실시간슬라이더로조정하고, 변경즉시 FPS 와 CPU/GPU 사용률에미치는영향을시각화하여사용자가최적의설정을탐색할수있도록지원해야합니다.

2.1.3 버퍼관리및프레임드롭전략　실시간비디오처리에서버퍼관리는지연시간과처리신뢰성사이의근본적트레이드오프를다룹니다. 무제한버퍼링은데이터손실을방지하지만메모리사용량급증과지연시간증가를초래하며, 반대로지나친버퍼제한은처리속도가입력속도를따라가지못할때프레임손실을야기합니다.

삼중버퍼 (Triple Buffering) 구조를권장합니다. 각카메라에대해 (1) **캡처버퍼** - 카메라드라이버로부터직접프레임을받는잠금없는 (lock-free) 원형버퍼로, 가장최신프레임을항상덮어쓰기 (overwriting) 하여지연시간을최소화, (2) **전처리버퍼** - 크기변환, 색상공간변환 (BGR→RGB), 정규화등의작업을수행하며, 멀티스레딩을통해여러프레임을병렬로준비, (3) **추론입력버퍼** - 배치처리를지원하며, 충분한프레임이모이거나타임아웃이발생하면추론을트리거합니다.

프레임드롭전략은여러수준에서구현됩니다. 첫번째수준은캡처단계에서의 **"선제적드롭"** 으로, 처리파이프라인의상태를모니터링하여밀리는경우새프레임캡처를일시적으로스킵합니다. 두번째수준은전처리단계에서의 **"선택적드롭"** 으로, 빠른움직임이감지된프레임만우선처리하고정적인장면은낮은우선순위로처리합니다. 세번째수준은추론단계에서의 **"품질기반드롭"** 으

로, confidence score 가낮은결과는후처리를생략하고다음배치로진행합니다. 이러한다층적드롭전략은시스템이과부하상황에서도 graceful degradation 을달성하게합니다.

버퍼유형	크기설정	드롭정책	목적
캡처버퍼	1-3 프레임	최신프레임덮어쓰기	지연시간최소화
전처리큐	5-10 프레임	오래된프레임버리기	처리량균형
추론입력	동적 (배치크기)	타임아웃기반강제실행	실시간성보장

2.2 컴퓨팅백엔드선택기

2.2.1 Apple Silicon (MPS 가속) M1 Pro 및후속 Apple Silicon 칩에서는 Metal Performance Shaders(MPS) 를통해 GPU 가속을활용할수있습니다. PyTorch 2.1 이상에서는 MPS 백엔드를네이티브로지원하며, `torch.device("mps")` 설정으로 CUDA 와유사한 API 로접근할수있습니다. 그러나 MPS 의완전한호환성은보장되지않습니다. 특히 YOLOv8-Pose 의경우, Ultralytics 공식문서와 GitHub 이슈를통해 MPS 에서의알려진제한사항이문서화되어있습니다. 특히포즈모델의훈련단계에서는 MPS 관련버그가보고되어있어, 공식적으로는 CPU 사용이권장되며, 추론단계에서는 MPS 가정상작동합니다.

MPS 백엔드의실제성능은모델크기와입력해상도에따라달라집니다. **YOLOv8n-pose 의경우, M1 Pro 에서 CPU-only 대비 2-3 배의속도향상이기대**되며, 640x640 입력에서 30fps 이상의실시간처리가가능합니다. 그러나 MPS 는 NVIDIA CUDA 에비해메모리관리와커널최적화면에서여전히성숙도가떨어지므로, 대형모델 (yolov8l/x-pose) 이나높은배치크기에서는메모리부족오류가발생할수있습니다. 따라서 MPS 환경에서는모델크기를 n 또는 s 로제한하고, 배치크기를 1-4 로유지하는것이안정적입니다.

MPS 사용시주의할점은 **메모리조각화 (fragmentation) 문제**입니다. 장시간실행시 MPS 메모리풀이조각화되어가용메모리가줄어드는현상이보고되었으므로, 주기적인 `torch.mps.empty_cache()` 호출을고려해야합니다. 또한 MPS 는 FP16(BFloat16) 연산을지원하므로, `model.half()`를통해반정밀도모드로실행하면메모리사용량과처리속도를추가로최적화할수있습니다.

2.2.2 NVIDIA CUDA/TensorRT NVIDIA GPU 환경에서는 CUDA 와 TensorRT 를통한최적화가 성능극대화의핵심입니다. CUDA 는 PyTorch 의기본 GPU 백엔드로, `torch.device("cuda")` 설정으로즉시활용가능합니다. YOLOv8-Pose 의경우, RTX 3060 이상의 GPU 에서는 640x640 입력에서 100fps 이상의처리가가능하며, 다중카메라환경에서도안정적인실시간성능을제공합니다. CUDA 의주요이점은성숙한생태계와풍부한최적화도구입니다. NVIDIA Nsight Systems 를통해커널수준의프로파일링이가능하고, Automatic Mixed Precision(AMP) 을통해 FP16/FP32 혼합연산으로속도와정확도의균형을조절할수있습니다.

TensorRT 는추론전용최적화도구로, PyTorch 모델을 ONNX 로변환한후 TensorRT 엔진으로빌드하여사용합니다. TensorRT 의최적화기법으로는레이어퓨전 (layer fusion), 정밀도교정 (calibration) 기반 INT8 양자화, 동적텐서메모리관리, 커널자동튜닝등이있습니다. YOLOv8-Pose 를 TensorRT 로변환하면, 동일한하드웨어에서 2-3 배의추가가속이가능하며, Jetson Orin Super 에서는필수적인최적화경로입니다. TensorRT 빌드시고려할파라미터로는최대배치크기, 최적화레벨 (preset), 타겟정밀도 (FP32/FP16/INT8) 등이있으며, 애플리케이션의지연시간-처리량요구사항에따라튜닝됩니다.

CUDA/TensorRT 환경에서의주요과제는 **버전호환성**입니다. PyTorch, CUDA Toolkit, cuDNN, TensorRT 의버전조합이제한적이므로, 공식호환성매트릭스를참고하여환경을구성해야합니다. 또한 Windows 환경에서는 CUDA 런타임의배포가추가복잡성을유발하므로, 정적링크또는 conda 환경패키징을고려해야합니다.

2.2.3 CPU 폴백모드 모든가속하드웨어가사용불가능한상황을대비한 CPU 폴백모드는시스템의견고성을위해필수적입니다. CPU-only 실행은속도가느리지만, 어떤환경에서도동작하는보장을제공합니다. YOLOv8-Pose 의 CPU 성능은모델크기에따라크게달라지며, yolov8n-pose 는현대적인멀티코어 CPU 에서 10-20fps 를제공할수있습니다.

CPU 최적화의핵심은 **멀티스레딩과벡터화**입니다. OpenMP 를활용한병렬처리와 Intel MKL 또는 OpenBLAS 를통한최적화된선형대수연산이기본이며, YOLOv8 의경우 NCNN 이나 OpenVINO 와같은추론프레임워크의도입을검토할수있습니다. 그러나 CPU 모드는실시간처리를보장하지못할수있으므로, 명확한성능한계를문서화하고사용자에게알려야합니다.

CPU 모드에서의추가최적화로는 **모델양자화**가있습니다. PyTorch 의동적양자화 (dynamic quantization) 를활용하면, 가중치를 INT8 로변환하여메모리대역폭을줄이고, Intel AVX-512 등의벡터명령어를활용한연산가속을달성할수있습니다. 다만, 양자화는약간의정확도저하를동반하므로, 민감한애플리케이션에서는주의가필요합니다.

2.2.4 OpenVINO (Intel 최적화) Intel CPU 및내장 GPU 에서최적의성능을달성하기위한대안으로 Intel OpenVINO Toolkit 을고려할수있습니다. OpenVINO 는 PyTorch 모델을 Intermediate Representation(IR) 형식으로변환하여, Intel 하드웨어특화최적화를적용합니다. 특히최근의 Intel CPU(Core 11 세대이상) 에탑재된 VNNI(Vector Neural Network Instructions) 와내장 GPU 의 DP4A 명령어를활용한 INT8 추론가속이주요이점입니다.

OpenVINO 의활용시나리오는두가지입니다. 첫째, M1 Mac 에서개발된애플리케이션을 Intel 기반서버로배포할때, 코드변경없이백엔드만교체하여최적화를달성할수있습니다. 둘째, Jetson 과함께 Intel NCS2(Neural Compute Stick 2) 를보조가속기로활용하는하이브리드구성에서, OpenVINO 는 NCS2 의 Myriad VPU 를효율적으로활용합니다. OpenVINO 2023.1 이상에서는 Ultralytics YOLOv8 모델을직접지원하며, `ov.convert_model()` API 를통해간편하게변환할수있습니다.

백엔드	지원하드웨어	상대성능	양자화지원	권장용도
MPS	Apple Silicon	1.0x (기준)	FP16	M1/M2/M3 Mac 개발
CUDA	NVIDIA GPU	1.5-3.0x	FP16, INT8	데스크톱/서버배포
TensorRT	NVIDIA GPU/ Jetson	3.0-10.0x	FP16, INT8	고성능엣지배포
OpenVINO	Intel CPU/GPU	0.3-1.5x	FP16, INT8	Intel 기반엣지
CPU (OpenMP)	범용	0.02-0.1x	없음	폴백, 알고리즘검증

3. 포즈추정알고리즘모듈

3.1 YOLOv8-Pose 통합

3.1.1 모델크기별옵션 (n/s/m/l/x) YOLOv8-Pose 는다양한컴퓨팅환경과정확도요구사항에맞춰 5 가지크기변형을제공하며, 테스트베이스앱에서는이러한옵션을런타임에동적으로선택할수있도록하여다양한하드웨어와시나리오에서의실험을지원합니다.

모델	파라미터수	FLOPs	COCO mAP (pose)	M1 Pro MPS FPS	Jetson Orin NX TensorRT FPS
YOLOv8n-pose	3.3M	9.2B	50.4%	45-60	120+
YOLOv8s-pose	11.6M	28.6B	60.0%	25-35	80-100
YOLOv8m-pose	26.4M	78.9B	65.0%	12-18	50-70
YOLOv8l-pose	44.4M	165.2B	68.6%	6-10	30-45
YOLOv8x-pose	68.7M	257.8B	70.3%	4-6	20-30

YOLOv8n-pose 는 3.3M 파라미터와 9.2B FLOPs 로매우가볍지만, COCO pose 데이터셋에서 50.4% 의 mAP 를달성하여기본적인행동인식에는충분한정확도를제공합니다. M1 Pro 의 MPS 가속에서는 45-60fps 를달성할수있어, 30fps 입력에대한실시간처리를여유있게만족합니다. **YOLOv8s-pose** 는정확도가 5.8%p 높지만연산량이 3 배증가하므로, GPU 가속이확실히보장될때선택합니다. 중형이상의모델 (m/l/x) 은서버급 GPU 나오프라인분석에적합하며, 엣지디바이스에서는실시간성을해칠수있습니다.

모델크기선택의실무적접근법은 "점진적업그레이드" 전략입니다. 초기개발단계에서는 yolov8n-pose 로빠른프로토타이핑을수행하고, 실제배포환경에서의정확도요구사항이명확해지면 yolov8s 또는 yolov8m 로전환합니다. 웹앱의실험인터페이스에서는사용자가실시간으로모델크기를전환할수있게하여, 특정환경에서의성능-정확도특성을직접경험하게합니다.

3.1.2 MPS/CUDA/CPU 디바이스자동선택 HAL 을통한디바이스자동선택은사용자의수동개입없이최적의실행환경을보장합니다. 선택로직은다음우선순위를따릅니다: (1) 사용자명시적지정, (2) CUDA 사용가능시 CUDA, (3) MPS 사용가능시 MPS, (4) 그외 CPU.

자동선택로직은 PyTorch 의백엔드가용성 API 를활용합니다:

```
import torch

def select_optimal_device(preferred=None):
    if preferred:
        return torch.device(preferred)
    if torch.cuda.is_available():
        return torch.device("cuda")
    elif torch.backends.mps.is_available():
        # YOLOv8-Pose 추론에 한해 MPS 사용, 훈련은 CPU로 폴백
        return torch.device("mps")
    else:
        return torch.device("cpu")
```

그러나 MPS 의경우, 앞서언급된바와같이 YOLOv8-Pose 훈련시알려진버그가있으므로, 모드별로다른선택로직이필요합니다. 추론모드에서는 MPS 를허용하되, 훈련모드또는안정성이우선되는상황에서는 CPU 로강제폴백하는것이바람직합니다. 또한 MPS 가용성확인시 `torch.backends.mps.is_built()`와 `torch.backends.mps.is_available()`를모두체크하여, PyTorch 빌드에 MPS 가포함되어있고런타임에 MPS-enabled 디바이스가존재하는지확인해야합니다.

동적오프로딩도고급기능으로고려할수있습니다. 멀티카메라환경에서특정카메라의처리가지연될경우, 해당스트림만 CPU 로오프로딩하여다른스트림의 GPU 처리를보장하는방식입니다. 이는우선순위기반스케줄링과결합하여, critical 한카메라(예: 출입구감시) 는 GPU 로, 덜중요한카메라는 CPU 로처리하는차등서비스를구현할수있습니다.

3.1.3 ONNX 변환및 TensorRT 최적화 프로덕션배포를위한핵심단계는 PyTorch 모델을 ONNX(Open Neural Network Exchange) 형식으로변환하고, NVIDIA 환경에서는 TensorRT 로추가최적화하는것입니다. ONNX 변환은 Ultralytics 에서기본제공하는 `export()` 메서드를통해수행됩니다:

```
from ultralytics import YOLO

model = YOLO("yolov8n-pose.pt")
model.export(format="onnx", dynamic=True, simplify=True)
```

`dynamic=True`는배치크기와입력해상도를런타임에조정할수있게하며, `simplify=True`는 ONNX Simplifier 를통해불필요한연산을제거합니다. 생성된 ONNX 모델은 Netron 등의도구로시각화하여, 변환과정에서의오류를사전에감지할수있습니다.

TensorRT 최적화는 ONNX 모델을입력으로받아, 타겟 GPU 아키텍처에특화된최적화된엔진을빌드합니다. TensorRT 의 Python API 를활용한빌드스크립트예시는다음과같습니다:

```
import tensorrt as trt

logger = trt.Logger(trt.Logger.INFO)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
parser = trt.OnnxParser(network, logger)

with open("yolov8n-pose.onnx", "rb") as f:
    parser.parse(f.read())

config = builder.create_builder_config()
config.max_workspace_size = 1 << 30 # 1GB
config.set_flag(trt.BuilderFlag.FP16) # FP16 최적화

engine = builder.build_engine(network, config)
with open("yolov8n-pose.engine", "wb") as f:
    f.write(engine.serialize())
```

TensorRT 빌드시고려할주요파라미터로는: (1) `max_workspace_size` - 최적화알고리즘을위한임시메모리, (2) precision mode - FP32/FP16/INT8 중선택, (3) `max_batch_size` - 최대배치크기, (4) DLA(Deep Learning Accelerator) 사용 - Jetson 의 DLA 코어활용여부등이있습니다. INT8 양자화의경우, 교정데이터셋을제공하여동적범위를결정해야하며, 이는정확도와속도의균형을조절하는데중요합니다.

3.1.4 17 개키포인트추출및정규화 YOLOv8-Pose 는 COCO 데이터셋형식의 17 개인체키포인트를출력합니다. 각키포인트는 (x, y, visibility) 또는 (x, y, confidence) 튜플로표현되며, 좌표는입력이미지크기에대한정규화된값 (0-1

범위) 또는픽셀좌표로제공됩니다. 17 개키포인트의구체적인정의는다음과같습니다:

인덱스	키포인트이름	설명	상위연결
0	nose	코	1, 2
1	left_eye	왼쪽눈	3
2	right_eye	오른쪽눈	4
3	left_ear	왼쪽귀	5
4	right_ear	오른쪽귀	6
5	left_shoulder	왼쪽어깨	7, 11
6	right_shoulder	오른쪽어깨	8, 12
7	left_elbow	왼쪽팔꿈치	9
8	right_elbow	오른쪽팔꿈치	10
9	left_wrist	왼쪽손목	-
10	right_wrist	오른쪽손목	-
11	left_hip	왼쪽엉덩이	13
12	right_hip	오른쪽엉덩이	14
13	left_knee	왼쪽무릎	15
14	right_knee	오른쪽무릎	16
15	left_ankle	왼쪽발목	-
16	right_ankle	오른쪽발목	-

키포인트추출후의정규화는행동인식모델의입력으로사용하기위해필수적입니다. 정규화의목적은 (1) 카메라거리와인물크기의변화에불변한표현, (2) 좌표계의일관성, (3) 수치안정성입니다. 주요정규화기법으로는:

1. **절대좌표정규화**: 모든키포인트를척추중심점 (hip center, 키포인트 11 과 12 의중점) 을원점으로이동시킵니다. 이는카메라위치나피사체의이동에더강건한특징을생성합니다.
2. **스케일정규화**: 어깨-엉댛이거리또는키추정치로모든좌표를나누어, 개인별체형차이를보정합니다.
3. **각도기반특징**: 관절간의각도를계산하여, 방향불변성을확보합니다. 예를들어, 팔꿈치각도는어깨-팔꿈치-손목세점으로계산됩니다.

정규화된키포인트는 34 차원벡터 (17 개 × 2) 로표현되며, 이는 LSTM 이나 ST-GCN 의입력으로직접사용됩니다. 신뢰도점수가낮은키포인트 (예: occlusion 으로인해감지되지않은경우) 는특수값 (예: -1) 으로마스킹하거나, 보간법으로추정합니다.

3.2 MediaPipe Pose 대안

3.2.1 BlazePose 아키텍처활용 **Google 의 MediaPipe Pose 는 BlazePose 라는경량 CNN 아키텍처를기반으로한대안적인포즈추정솔루션**입니다. YOLOv8-Pose 와비교하여단일인물추정에특화되어있으며, 모델크기가훨씬작아모바일/웹환경에서의실행이용이합니다.

BlazePose 의핵심혁신은 **heatmap-free regression 접근법**입니다. 전통적인방식의히트맵기반키포인트검출대신, 직접적인좌표회귀를수행하여해상도에독립적인일관된정확도를달성합니다. 이는특히저해상도입력에서의속도이점을제공합니

다. MediaPipe 는 **33 개의키포인트**를출력하며, 이는 YOLOv8 의 17 개보다상세합니다. 추가포인트는주로얼굴세부사항 (입, 코, 눈의세부랜드마크) 과손가락뼈대에해당합니다. 그러나 HAR 시나리오에서는이러한세부사항이오히려노이즈가될수있으므로, 17 개 COCO 호환서브셋으로매핑하거나가중치를조정하는후처리가필요합니다.

3.2.2 CPU-최적화라이트모드 **MediaPipe 의 `POSE_LANDMARKS_LITE` 모델은순수 CPU 에서도 30fps 이상의실시간처리를목표**로합니다. 이는 GPU 가속을사용할수없는환경이나, GPU 를다른작업 (예: 행동분류신경망) 에전용하고자할때유용합니다. 라이트모드의정확도는풀모델대비다소저하되지만, 대부분의기본 HAR 시나리오 (걷기, 서기, 앉기등) 에서는여전히충분합니다.

벤치마크에따르면, **M1 Pro CPU 에서 MediaPipe Pose 라이트모델은약 81fps 를달성**할수있어, 상당한여유있는실시간성능을제공합니다. MediaPipe 의 CPU 최적화는 Google 의 XNNPACK 라이브러리를활용한 NEON/AVX 벡터화와연산그래프최적화를통해달성됩니다. 라이트모드에서는정확도와속도사이의트레이드오프를조정할수있는파라미터 (`model_complexity`: 0/1/2) 를제공하며, 가장가벼운설정 (0) 에서는처리속도를극대화하고, 복잡한설정 (2) 에서는정확도를우선시합니다.

CPU 플랫폼	평균처리시간 (ms)	최소/최대 (ms)	FPS	특이사항
Apple M1	12.3	9.8 / 18.7	~81	NPU 가속지원
Intel i7-10700K	14.6	11.2 / 21.5	~68	AVX-512 활용
Intel i5-8250U	23.4	19.1 / 35.6	~43	저전력모바일 CPU
AMD Ryzen 5 3600	16.8	13.5 / 24.3	~59	Zen 2 아키텍처

3.2.3 다중인물한계및대응전략 **MediaPipe Pose 의가장큰실용적한계는공식솔루션이단일인물에최적화되어있다는점**입니다. 여러사람이동시에화면에나타나는 CCTV 환경에서, MediaPipe 는주요인물하나만검출하거나, 여러인물의키포인트가혼합된잘못된결과를출력할수있습니다.

이러한한계에대한대응전략으로는: (1) **YOLOv8 과결합한탑다운접근법** - 별도의객체검출기로인물영역을추출하고, 각영역별로 MediaPipe Pose 를개별적용, (2) **MediaPipe Holistic 솔루션과커스텀후처리** - 다중인물을추론하는비공식확장활용, (3) **완전한대체** - 다중인물포즈추정에특화된 YOLOv8-Pose 로의전환이있습니다. 테스트베이스앱에서는이러한대안들을모듈로제공하여, 사용자가시나리오에따라적절한방법을선택할수있도록합니다.

3.3 알고리즘성능비교도구

3.3.1 FPS/지연시간벤치마크 **다양한포즈추정알고리즘의실제성능을객관적으로비교하기위한자동화된벤치마크도구**를구축합니다. 이도구는동일한테스트비디오세트 (다양한해상도, 인물수, 동작복잡도) 에대해각알고리즘을실행하고, 초당프레임수 (FPS), end-to-end 지연시간 (캡처부터렌더링까지), 그리고 GPU/CPU 사용률을측정합니다.

측정은웜업 (warm-up) 단계를포함하여캐시효과를안정화시킨후수행되며, 통계적신뢰성을위해여러번반복하여평균과표준편차를계산합니다. 벤치마크결과는웹앱인터페이스에서실시간으로시각화되며, 사용자가특정하드웨어구성에서어떤알고리즘이최적인지데이터기반으로판단할수있도록지원합니다. 추가로, 장시간안정성테스트 (예: 1 시간연속실행) 를통해메모리누수나성능저하문제도감지합니다.

3.3.2 정확도-속도트레이드오프시각화 **단순한속도비교를넘어, 다양한정확도메트릭과처리속도간의관계를시각화**하여사용자가자신의요구사항에최적의지점을선택할수있도록합니다. 정확도메트릭으로는 COCO keypoints benchmark 의 OKS(Object Keypoint Similarity), PCK(Percentage of Correct Keypoints), 그리고사용자정의테스트세트에서의행동인식정확도를활용합니다.

이들메트릭을 Y 축에, 처리속도 (FPS) 를 X 축에배치한 scatter plot 이나 Pareto frontier 시각화를통해, 각알고리즘변형이정확도-속도공간에서차지하는위치를직관적으로파악할수있습니다. 사용자는이시각화를통해 "내애플리케이션은 25fps 이상이필요하고, 이조건에서가장정확한알고리즘은무엇인가?" 와같은질문에답을얻을수있습니다.

4. 객체추적및 ID 관리

4.1 추적알고리즘플러그인

4.1.1 DeepSORT (권장) DeepSORT 는 Simple Online and Realtime Tracking with a Deep Association Metric 의약자로, 객체검출과외관기반재식별 (Re-ID) 을결합한실시간다중객체추적의표준접근법입니다. HAR 테스트베이스에서의핵심역할은프레임간동일인물의일관된 ID 유지를보장하여, 시계열행동분석의기초를제공하는것입니다.

DeepSORT 의핵심구성요소는다음과같습니다:

Kalman 필터기반모션예측: 각추적대상의바운딩박스를 (x, y, a, h, ẋ, ẏ, ȧ, ḣ) 의 8 차원상태벡터로모델링하며, 상수속도모션모델을가정합니다. 검출이누락된프레임에서도예측된위치로추적을유지하며, 최대 30 프레임 (1 초) 까지의 occlusion 을견딜수있습니다.

외관특징임베딩: 별도의 CNN(일반적으로 ResNet 기반) 이각검출영역으로부터 128 차원의외관특징벡터를추출합니다. 이는모션정보만으로구분이어려운 occlusion 상황에서의 ID 스위칭방지에결정적입니다. HAR 시나리오에서는포즈추정의키포인트정보도추가특징으로활용할수있으며, 이를위한커스텀연관메트릭구현이확장지점으로预留됩니다.

헝가리안알고리즘기반데이터연관: 예측된트랙과새로운검출간의비용행렬을구성하고, 최적의일대일매칭을찾습니다. 비용은모션거리 (Mahalanobis distance) 와외관거리 (코사인유사도) 의가중조합으로정의되며, 임계값을초과하는쌍은새로운트랙으로간주됩니다.

4.1.2 ByteTrack (경량대안) ByteTrack 은 2022 년에제안된최신추적알고리즘으로, "Every Detection Box Matters" 라는철학을기반으로합니다. DeepSORT 에비해외관특징네트워크가없어메모리사용량이적고, 순수하게검출기반으로동작하여구현이단순합니다. ByteTrack 은 Kalman 필터와헝가리안알고리즘을사용하되, 연관과정에서신뢰도임계값을다단계로적용하는것이특징입니다. 낮은신뢰도의검출결과까지활용하여추적성능을향상시키며, Jetson 과같은리소스제한환경에서 DeepSORT 의외관네트워크가부담스러울때 ByteTrack 을대안으로제공합니다.

4.1.3 IOU 기반간단추적 (베이스라인) 가장단순한추적방법으로, 현재프레임의검출과이전프레임의예측간 IOU 만을사용하여연관을결정합니다. Kalman 필터나외관특징없이순수한위치중첩에의존하므로, 빠른움직임이나 occlusion 에취약하지만구현이매우단순하고오버헤드가거의없습니다. 이방법은 "추적없음" 옵션과의비교베이스라인으로활용되며, 추적알고리즘자체의가치를정량적으로평가하는데사용됩니다.

추적알고리즘	외관특징	Occlusion 처리	메모리사용	권장시나리오
DeepSORT	128 차원 CNN 임베딩	30 프레임예측유지	중간	복잡한 occlusion, Re-ID 필요
ByteTrack	없음 (검출신뢰도활용)	다단계임계값	낮음	리소스제한, 빠른움직임
IOU 기반	없음	없음	최소	단순한장면, 베이스라인비교

4.2 다중객체상태관리

4.2.1 객체등장/소멸감지 추적시스템은각객체의생명주기를관리하며, 새로운객체의등장 (새로운검출이기존궤적과연관되지않음) 과소멸 (일정프레임동안검출되지않음) 을감지합니다. 등장이벤트는새로운 ID 할당과함께로그에기록되며, 소멸이벤트는해당객체의궤적데이터를최종저장하고메모리에서해제합니다. 이러한생명주기관리는행동인식의정확한시간범위정의에중요하며, "사람이화면에들어와서 ~ 나갈때까지" 의완전한행동시퀀스를보장합니다.

4.2.2 ID 스위칭방지전략 ID 스위칭 (ID switch) 은두객체가교차하거나가까이지나갈때그들의 ID 가바뀌는현상으로, 다중객체추적의주요오류유형입니다. 이를방지하기위한전략으로는 (1) 외관특징의적극적활용 (DeepSORT), (2) 객체크기와종횡비의일관성검사, (3) 급격한위치변화에대한물리적제약적용, (4) 궤적의속도방향일관성검증등이있습니다. 테스트베이스앱에서는이러한전략들을파라미터화하여, 사용자가 ID 스위칭민감도를조절할수있게합니다. ID 스위칭발생시시각적경고 (색상변화또는텍스트표시) 를제공하여운영자의주목을유도합니다.

4.2.3 궤적버퍼및이력관리 각객체의과거위치이력은행동인식의입력으로활용되므로, 효율적인궤적버퍼관리가필요합니다. 고정크기의 Ring Buffer(예: 최근 90 프레임, 3 초분량) 를사용하여메모리사용량을제한하면서도충분한시간적문맥을유지합니다. 궤적데이터는 (timestamp, bbox, keypoints) 튜플의시퀀스로저장되며, 객체소멸시에는선택적으로디스크에영구저장하거나비디오클립으로추출합니다. 행동인식모듈은이버퍼로부터슬라이딩윈도우를추출하여시퀀스분류를수행합니다.

5. 행동인식 (HAR) 엔진

5.1 시퀀스처리파이프라인

5.1.1 슬라이딩윈도우버퍼 (30 프레임/1 초) 행동인식은단일프레임이아닌시간적문맥을필요로하므로, 연속된프레임시퀀스를효율적으로관리하는버퍼메커니즘이핵심입니다. 슬라이딩윈도우방식은고정된크기 (예: 30 프레임, 약 1 초 @ 30fps) 의버퍼를유지하면서, 새프레임이들어올때마다가장오래된프레임을제거하고새프레임을추가합니다. 이방식은연속적인행동인식출력을생성하며, 윈도우간중첩 (예: 50% 오버랩) 을설정하면시간적해상도를높일수있습니다.

버퍼의데이터구조는 `(batch, sequence, features)` 형태의텐서로, 여기서 `features`는 17 개키포인트의 (x,y) 좌표로구성된 34 차원벡터입니다. 즉, 기본설정에서버퍼텐서의 shape 는 `(1, 30, 34)`가됩니다. 슬라이딩윈도우의크기는행동의시간적특성에따라조정가능해야하며, 빠른행동 (예: 주먹질) 에는작은윈도우 (15 프레임), 느린행동 (예: 쓰러짐) 에는큰윈도우 (60 프레임) 가적합할수있습니다.

5.1.2 키포인트정규화 (상대좌표/중심점기준) 원시키포인트좌표는카메라거리, 인물크기, 위치에따라절대값이크게변하므로, 행동인식모델의입력으로적합하지않습니다. 따라서다양한정규화방법을적용하여위치와스케일불변성을확보합니다:

척추중심점기준상대좌표: 두엉덩이 (hip) 키포인트의중점을원점 (0,0) 으로설정하고, 모든키포인트를이원점에대해상대적으로재계산합니다. 이는인물의화면내위치에관계없이일관된신체구조표현을제공합니다.

어깨거리정규화: 두어깨키포인트간의유클리드거리를 1.0 으로설정하는스케일정규화를수행합니다. 이는카메라로부터의거리에따른인물크기변화를보정하여, 동일한행동이가까이또는멀리서수행되어도유사한입력패턴을생성합니다.

Z-score 정규화: 키포인트좌표의분포를표준정규분포로변환하여, 딥러닝모델의수렴을가속화합니다. 이는학습데이터의통계특성을활용하며, 온라인정규화 (실시간평균/분산추정) 도가능합니다.

테스트베이스앱에서는이러한정규화방법을실시간으로전환하여비교할수있는인터페이스를제공하며, 각방법의행동인식정확도에미치는영향을분석할수있게합니다.

5.1.3 시계열데이터증강 제한된학습데이터상황에서모델의일반화를향상시키기위한시계열증강기법을적용합니다. 공간적증강으로는키포인트좌표에대한작은랜덤노이즈추가, 수평뒤집기 (좌우대칭행동의경우), 그리고관절각도의작은왜곡이포함됩니다. 시간적증강으로는시퀀스의시간적리샘플링 (빠르게/느리게재생시뮬레이션), 랜덤한프레임마스킹 (occlusion 시뮬레이션), 그리고슬라이딩윈도우의랜덤시프트가있습니다. 이러한증강은훈련시에만적용되며, 추론시에는원본데이터를사용합니다.

5.2 분류알고리즘모듈

5.2.1 LSTM/GRU 기반시퀀스분류 **LSTM(Long Short-Term Memory) 은시계열데이터처리에강한모델로, 연속된프레임의관절좌표를입력받아행동을예측**합니다. YOLOv8-Pose 가추출한좌표의' 시간적변화패턴' 을학습하여 "걷기", "뛰기", "쓰러짐", "싸움" 등의행동클래스를분류합니다.

구체적인아키텍처로는 **2-Layer LSTM with Hidden size 64, Dropout(0.5), 그리고 Fully Connected Layer** 를권장합니다. 입력은 30 프레임 × 34 차원 (17 키포인트 × 2 좌표) 의정규화된키포인트시퀀스이며, 출력은 Softmax 를통한행동클래스확률분포입니다. GRU(Gated Recurrent Unit) 는 LSTM 보다파라미터수가적어경량화된대안으로활용할수있습니다.

구성요소	설정	설명
LSTM 레이어수	2	충분한표현력과훈련안정성의균형
Hidden size	64	메모리효율과성능의 sweet spot
Dropout	0.5	과적합방지
출력활성화	Softmax	다중클래스분류
손실함수	Cross-Entropy	표준분류손실
옵티마이저	Adam	적응적학습률

5.2.2 ST-GCN (그래프신경망) **ST-GCN(Spatial Temporal Graph Convolutional Network) 은관절의움직임을그래프형태로학습**하여 "걷기", "뛰기", "주먹질" 등을분류합니다. 인체의자연스러운관절연결구조를그래프의노드와엣지로모델링하여, 공간적관계와시간적변화를동시에포착합니다. ST-GCN 은특히관절간의상호작용이중요한행동 (예: 손흔들기, 포옹) 에서 LSTM 보다우수한성능을보일수있으나, 구현복잡도와계산비용이更高습니다.

5.2.3 규칙기반폴백 (Rule-based) **간단한수학적규칙으로행동을정의하는방식은즉시구현가능하고해석가능성이높다는장점**이있습니다. 주요규칙예시:

쓰러짐 (Fall Detection): 머리의 Y 좌표가급격히낮아지고 (예: 0.3 초내 30% 이상감소), 몸의가로/세로비율이넓어질때 (예: aspect ratio > 2.0).

배회 (Loitering): 특정 ROI(관심영역) 내에서사람의중심좌표이동거리가일정시간동안 (예: 60 초) 임계값미만일때 (예: 총이동거리 < 2 미터).

침입 (Intrusion): 금지된구역 (Polygon) 안에발목좌표가들어왔을때, 또는특정방향으로의경계횡단이감지될때.

규칙기반방식은딥러닝모델의블랙박스특성을보완하여, 안전-critical 한애플리케이션에서의신뢰성을높입니다. 테스트베이스앱에서는규칙기반과딥러닝기반결과를융합하는하이브리드방식도제공합니다.

5.3 행동클래스정의및확장

5.3.1 기본클래스 (걷기, 서기, 앉기, 뛰기) **일상적활동을나타내는 4 가지기본행동클래스**는대부분의 HAR 시스템의기초가됩니다. 이들클래스는데이터셋에서충분한샘플을확보하기쉽고, 상대적으로높은인식정확도를달성할수있습니다. 클래스간의구분특성: 걷기는주기적인다리움직임, 서기는정적인자세, 앉기는엉덩이높이의급격한감소, 뛰기는수직방향가속도의증가를특징으로합니다.

5.3.2 위험행동 (쓰러짐, 싸움, 배회) **안전모니터링애플리케이션에서핵심적인위험행동클래스**입니다. 쓰러짐은노인돌봄에서, 싸움은공공안전에서, 배회는치매환자감시에서각각중요한이벤트입니다. 이들클래스는불균형데이터셋문제 (정상행동에비해드문발생) 와높은오탐지비용으로인해, 더정교한모델아키텍처와임계값튜닝이필요합니다.

5.3.3 사용자정의행동추가인터페이스 테스트베이스앱의핵심가치중하나는새로운행동클래스의쉬운추가입니다. 웹인터페이스를통해사용자가 (1) 행동이름정의, (2) 샘플비디오업로드또는실시간녹화, (3) 자동라벨링 (기존모델의예측을초기라벨로활용) 또는수동라벨링, (4) 모델미세조정 (fine-tuning) 또는규칙정의를수행할수있도록합니다. 새로운행동은기존클래스와의혼동을최소화하기위해, 충분히구별되는특징을가져야하며, 이를위한특징중요도분석도구도제공합니다.

6. 테스트베이스웹앱기능

6.1 실시간모니터링대시보드

6.1.1 멀티뷰비디오스트리밍 (1→4 카메라확장) Streamlit 기반의웹인터페이스는초기단계에서빠른프로토타이핑을가능하게합니다. 단일카메라뷰에서시작하여, `st.columns()`와 `st.image()`를활용한 2x2 그리드레이아웃으로 4 카메라확장을지원합니다. 각카메라뷰는독립적인세션상태를유지하며, 전체화면모드와개별카메라확대/축소기능도제공합니다. WebRTC 를활용한저지연스트리밍이나, HLS/DASH 를활용한대규모배포도향후확장옵션으로고려합니다.

6.1.2 스켈레톤오버레이시각화 검출된 17 개키포인트와그연결을실시간으로영상위에렌더링합니다. COCO 데이터셋의기본연결 (skeleton) 구조를따르며, 키포인트신뢰도에따라색상강도를조절하여시각적피드백을제공합니다. 사용자는오버레이의투명도, 선두께, 키포인트크기를실시간으로조정할수있으며, 특정관절만선택적으로표시하는필터링기능도제공합니다.

6.1.3 행동클래스및신뢰도실시간표시 각감지된객체에대해현재예측된행동클래스와신뢰도점수를바운딩박스옆에표시합니다. 높은신뢰도 (>0.8) 는녹색, 중간 (0.5-0.8) 은노란색, 낮은 (<0.5) 은빨간색으로시각화하여운영자의주목을유도합니다. 최근 N 초간의행동이력을작은스파크라인 (sparkline) 그래프로표시하여, 행동의변화추이를한눈에파악할수있게합니다.

6.2 알고리즘실험인터페이스

6.2.1 모델/파라미터 A/B 테스트 웹앱의핵심실험기능으로, 동일한입력에대해두가지알고리즘설정을병렬실행하고결과를실시간비교합니다. 예를들어, 왼쪽패널에는 YOLOv8n-pose + LSTM, 오른쪽패널에는 YOLOv8s-pose + 규칙기반을표시하여, 정확도와지연시간의트레이드오프를직접확인할수있습니다. A/B 테스트의결과 (FPS, 정확도, 메모리사용량) 는자동으로로그에기록되고, 나중에 CSV 형식으로내보내기가능합니다.

6.2.2 슬라이딩윈도우크기조정 행동인식의시간적문맥길이를실시간으로조정하여, 짧은윈도우 (15 프레임, 0.5 초) 와긴윈도우 (60 프레임, 2 초) 의성능차이를실험합니다. 슬라이더위젯으로 15-120 프레임범위에서자유롭게조정가능하며, 변경즉시행동예측결과의변화를관찰할수있습니다. 이는특정행동클래스의최적윈도우크기를경험적으로결정하는데활용됩니다.

6.2.3 임계값실시간튜닝 행동감지의민감도를조절하는다양한임계값을실시간으로변경합니다. 키포인트신뢰도임계값, 행동분류확률임계값, 규칙기반행동의물리적파라미터 (예: 쓰러짐감지의 Y 좌표변화율) 등을슬라이더로조정하고, false positive 와 false negative 의변화를즉시확인합니다. 최적의임계값조합을저장하여프로필로관리할수있습니다.

6.3 코드분석및설명기능

6.3.1 알고리즘흐름시각화 현재실행중인파이프라인의데이터흐름을다이어그램으로표시합니다. 카메라입력 → 전처리 → 포즈추정 → 추적 → 행동분류 → 출력의각단계를노드로, 데이터의존성을엣지로표현하며, 각노드의현재처리량과지연시간을실시간으로오버레이합니다. 이는병목지점식별과시스템이해에큰도움이됩니다.

6.3.2 핵심코드스니펫하이라이트 선택된알고리즘의핵심구현을 Syntax highlighting 된코드로표시합니다. 예를들어, LSTM 의 forward pass, DeepSORT 의연관로직, 또는규칙기반쓰러짐감지의수학적조건등을보여주며, 코드라인별로간략한주석을달아이해를돕습니다. 코드는 GitHub 저장소의특정버전과연동되어, 항상최신상태를유지합니다.

6.3.3 버전별변경이력비교 동일알고리즘의두버전간의차이를 diff 형식으로표시합니다. 예를들어, YOLOv8-Pose v1.2.0 과 v1.3.0 사이의모델아키텍처변경, 하이퍼파라미터조정, 또는버그수정내역을명확히보여주며, 성능변화의원인을추적하는데활용됩니다. Semantic versioning 의메이저/마이너/패치변경유형도함께표시합니다.

7. 데이터관리및로깅

7.1 행동이벤트로그

7.1.1 타임스탬프기반이벤트저장 모든감지된행동이벤트는정밀한타임스탬프 (밀리초단위) 와함께구조화된형식으로저장됩니다. 로그항목: 이벤트 ID, 타임스탬프, 카메라 ID, 객체 ID, 행동클래스, 신뢰도, 지속시간, 관련키포인트시퀀스의참조. 저장형식은효율적인쿼리를위해 Parquet 또는 SQLite 를사용하며, 대용량배치처리를위해 JSON Lines 도지원합니다.

7.1.2 비디오클립자동추출 행동이벤트발생전후의영상을자동으로클립으로저장하여, 나중의검증이나증거자료로활용합니다. 클립길이는이벤트유형별로설정가능 (예: 쓰러짐: 전 5 초, 후 10 초) 하며, 중복이벤트의경우클립을병합하거나우선순위에따라선택합니다. 저장형식은 H.264 로압축하여용량을최소화하고, 메타데이터는별도의 JSON 파일로함께저장합니다.

7.1.3 JSON/CSV 내보내기 수집된로그데이터를표준형식으로내보내기하여, 외부분석도구와의연동을지원합니다. JSON 은계층적구조와메타데이터보존에, CSV 는스프레드시트호환성과간단한통계분석에각각적합합니다. 내보내기시시간범위, 카메라, 행동클래스등의필터를적용할수있습니다.

7.2 알고리즘성능메트릭

7.2.1 추론시간분포 각파이프라인단계의처리시간을상세히기록하고분석합니다. 전처리, 포즈추정, 추적, 행동분류, 후처리/렌더링의각단계별평균, 중앙값, 95th 백분위수, 최대값을계산하여, 이상치와병목지점을식별합니다. 히스토그램과박스플롯으로시각화하며, 시간에따른추이도추적합니다.

7.2.2 정확도-재현율곡선 행동분류모델의성능을다양한임계값에서평가합니다. 검증데이터셋에대한예측결과를바탕으로, 각행동클래스별 Precision-Recall 곡선을그리고 AP(Average Precision) 를계산합니다. 이는임계값선택의객관적근거를제공하며, 클래스간불균형문제도드러냅니다.

7.2.3 혼동행렬누적 시간에따른예측결과의혼동행렬을누적하여표시합니다. 어떤행동쌍이자주혼동되는지 (예: 걷기와뛰기) 를파악하여, 모델개선의우선순위를정합니다. 혼동행렬은정규화된형태 (행합 =1) 와원수형태모두제공합니다.

7.3 데이터셋구축도구

7.3.1 자동라벨링인터페이스 기존모델의예측을초기라벨로활용하여, 수동라벨링의효율을높입니다. 사용자는예측결과를검토하고필요시수정만수행하면되며, 수정내역은모델개선을위한피드백으로활용됩니다. 활성학습 (active learning) 기법을적용하여, 모델이불확실한샘플을우선적으로제안하도록합니다.

7.3.2 키포인트시퀀스시각화편집기 수집된키포인트시퀀스를 2D/3D 로시각화하고, 오류를수동으로수정할수있는도구입니다. 특정프레임의키포인트위치를드래그로조정하거나, occlusion 으로인해누락된키포인트를보간법으로채울수있습니다. 수정이력은버전관리되며, 원본데이터는보존됩니다.

8. 경보및알림시스템

8.1 이벤트기반트리거

8.1.1 특정행동감지시시각적경보 위험행동 (쓰러짐, 싸움등) 감지시화면에뚜렷한시각적표시를제공합니다. 바운딩박스의색상변경 (녹색 → 빨간색), 화면테두리깜빡임, 또는전체화면오버레이등의방식을사용하며, 심각도수준에따라다른시각적패턴을적용합니다.

8.1.2 임계값초과시알림발송 신뢰도임계값과지속시간임계값을모두충족할때만알림을발송하여, 오탐으로인한알림피로를방지합니다. 예를들어, 쓰러짐신뢰도 > 0.8 이 3 초이상지속될때만알림을트리거합니다. 알림내용: 이벤트유형, 발생시간, 카메라위치, 스냅샷이미지, 관련비디오클립링크.

8.1.3 중복알림방지 (쿨다운) 동일유형의알림이너무빈번하게발송되지않도록쿨다운메커니즘을적용합니다. 동일객체에대한동일행동알림은최소 5 분간격으로제한하며, 심각도가높아지는경우 (예: 쓰러짐후움직임없음) 에는예외적으로즉시알림을발송합니다.

8.2 알림채널

8.2.1 웹소켓실시간푸시 Streamlit 의세션상태를활용한브라우저내실시간알림으로, 가장기본적인채널입니다. 별도의외부서비스의존없이작동하며, 데스크톱알림 API 와연동하여브라우저가백그라운드에있어도알림을표시할수있습니다.

8.2.2 이메일/Slack 연동 (확장) SMTP 를통한이메일발송과 Slack Webhook 을통한채널메시지를지원합니다. 환경변수또는설정파일로자격증명을관리하며, 심각도수준에따라다른채널로라우팅하는규칙을설정할수있습니다 (예: 경고 = 이메일, 심각 =Slack+ 이메일).

8.2.3 로컬사운드알림 데스크톱환경에서의즉각적인청각적피드백을제공합니다. 사용자지정가능한알림소리와볼륨, 그리고 "방해금지모드" 에서의자동음소거기능을포함합니다.

9. 라이브러리화및배포

9.1 핵심모듈패키징

9.1.1 `har_core`: 포즈추정 + 추적 가장기본적인기능을담당하는패키지로, YOLOv8-Pose/MediaPipe 통합, DeepSORT/ByteTrack 추적, 그리고 HAL 의기본구현을포함합니다. 의존성: PyTorch, OpenCV, NumPy, Ultralytics. 최소한의기능만으로빠른설치와임포트를목표로합니다.

9.1.2 `har_action`: 행동분류엔진 LSTM, ST-GCN, 규칙기반분류기를통합한패키지로, 시퀀스처리파이프라인과훈련/추론유틸리티를포함합니다. 의존성: PyTorch, scikit-learn(평가메트릭). `har_core`의출력을입력으로받아동작하며, 독립적으로업데이트가능합니다.

9.1.3 `har_pipeline`: end-to-end 파이프라인 전체워크플로우를 orchestration 하는고수준패키지로, 멀티카메라관리, 설정로딩, 로깅, 알림등의기능을포함합니다. 의존성: `har_core`, `har_action`, Streamlit(선택적). 빠른시작을위한사전설정파이프라인템플릿을제공합니다.

9.2 플랫폼별빌드

9.2.1 PyPI 패키지 (pip install) 표준 Python 패키지관리자를통한배포로, `pip install har-core har-action har-pipeline` 명령으로설치가능합니다. 플랫폼별 wheel 파일을미리빌드하여, 사용자의로컬컴파일없이빠른설치를지원합니다. 의존성버전은엄격하게고정하여, 환경불일치문제를방지합니다.

9.2.2 Docker 컨테이너 (Jetson 호환) NVIDIA Container Toolkit 을활용한 GPU 가속컨테이너로, Jetson 용으로는 l4t-base 이미지를기반으로합니다. 개발환경과배포환경의완전한일치를보장하며, Kubernetes 와같은오케스트레이션플랫폼과의통합도용이합니다. 멀티스테이지빌드를활용하여최종이미지크기를최소화합니다.

9.2.3 Windows 실행파일 (PyInstaller) 단일파일실행가능한배포물로, Python 런타임과모든의존성을포함합니다. `--onefile` 옵션의단순함과 `--onedir` 옵션의빠른시작시간사이의트레이드오프를고려하여, 최종사용자경험에맞게선택합니다. 코드서명인증서를적용하여, Windows Defender 의오탐지를방지합니다.

9.3 API 문서화

9.3.1 자동생성 Sphinx 문서 코드내 docstring 에서자동으로 HTML 문서를생성합니다. Google Style 또는 NumPy Style docstring 을사용하며, 타입힌트도함께문서화합니다. Read the Docs 또는 GitHub Pages 를통한무료호스팅을활용합니다.

9.3.2 Jupyter 노트북튜토리얼 **단계별학습을위한실행가능한예제**를제공합니다. 기본사용법부터고급커스터마이징까지, 실제데이터와함께동작하는코드를포함합니다. Google Colab 호환성도검증하여, 로컬환경없이도체험가능하게합니다.

10. 개발로드맵및체크리스트

10.1 1 단계: 단일카메라프로토타입 (4 주)

주차	주요작업	산출물	성공기준
1 주	개발환경설정, HAL 초기구현	MPS 백엔드동작확인	YOLOv8n-pose 30fps 이상
2 주	Streamlit 대시보드기본구조	실시간영상표시	웹인터페이스응답 $<$ 100ms
3 주	규칙기반행동감지구현	쓰러짐/침입감지로직	정확도 $>$ 80% (검증데이터)
4 주	통합테스트및문서화	Phase 1 보고서	전체파이프라인안정화

10.2 2 단계: 멀티카메라확장 (4 주)

주차	주요작업	산출물	성공기준
1 주	멀티스레드입력파이프라인	2 카메라동시처리	카메라간지연 $<$ 50ms
2 주	DeepSORT 통합및튜닝	다중객체추적	ID 스위칭률 $<$ 5%
3 주	LSTM 행동분류통합	4 가지기본행동인식	분류정확도 $>$ 85%
4 주	A/B 테스트인터페이스완성	알고리즘비교도구	4 카메라 10fps 안정화

10.3 3 단계: Jetson 최적화 (4 주)

주차	주요작업	산출물	성공기준
1 주	ONNX 변환및 TensorRT 빌드	최적화된엔진파일	FP16 대비 2 배가속
2 주	GStreamer 파이프라인구축	하드웨어가속디코딩	CPU 사용률 $<$ 30%
3 주	Docker 컨테이너화	배포가능한이미지	cold start $<$ 60 초

주차	주요작업	산출물	성공기준
4 주	엣지-클라우드하이브리드설계	원격모니터링프로토콜	네트워크단절시자율동작

10.4 4 단계: Windows 네이티브전환 (4 주)

주차	주요작업	산출물	성공기준
1 주	PyQt/Flutter UI 프로토타입	네이티브앱기본구조	주요화면 1:1 구현
2 주	DirectShow 카메라통합	Windows 네이티브입력	USB/RTSP 모두지원
3 주	설치프로그램및자동업데이트	MSI/EXE 빌드	설치시간 $<$ 5 분
4 주	최종통합테스트및릴리스	v1.0 정식버전	72 시간연속안정동작

핵심체크리스트 (개발시작시)

- ☐ Python 3.8+ 및 PyTorch 2.1+ 설치
- ☐ Ultralytics (`pip install ultralytics`)
- ☐ OpenCV (`pip install opencv-python`)
- ☐ Streamlit (`pip install streamlit`)
- ☐ 테스트용 CCTV 샘플영상확보 (YouTube 또는자체촬영)
- ☐ M1 Pro MPS 가속확인 (`torch.backends.mps.is_available()`)
- ☐ YOLOv8n-pose 기본추론테스트

권장첫번째실험: YOLOv8-Pose 를이용해영상에서사람의관절을실시간으로추출하는코드를실행한후, 이를기반으로규칙기반쓰러짐감지로직을추가합니다. 이단계가완료되면 Phase 2 의멀티카메라확장으로진행합니다.