JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 27, No. 3, pp. 813-820, Mar. 2026

Check for updates

# NPU 기반 엣지 AI 가속을 위한 파이프라인의 지배적 병목 분석: 전·후처리 단계의 정량적 영향 평가

서 영 진[1*] · 장 훈[2]
[1]세종사이버대학교 소프트웨어공학과 겸임교수
[2]숭실대학교 컴퓨터학과 교수

# Dominant Bottleneck Analysis in NPU-Accelerated Edge AI Pipelines: Quantitative Evaluation of Pre- and Post-Processing Stages

**Young-Jin Suh[1*] · Hoon Jang[2]**

[1]Adjunct Professor, Department of Software Engineering, Sejong Cyber University, Seoul 05000, Korea
[2]Professor, Department of Computer Science, Soongsil University, Seoul 06978, Korea

**[요 약]**

본 연구는 NPU(Neural Processing Unit) 가속 엣지 환경에서 전·후처리 단계가 시스템 성능과 발열에 미치는 영향을 정량적으로 분석하였다. 라즈베리 파이 5와 Hailo-8 환경에서 YOLOv8을 대상으로 ARM NEON과 OpenMP를 결합한 하이브리드 병렬화 전략을 적용한 결과, 기존 방식(Baseline) 대비 최대 2.26배의 처리 속도(24.2 FPS)와 1.77배의 에너지 효율 향상을 달성하였다. 특히 전처리 최적화만으로도 2.14배의 성능 개선을 보여 전처리 단계가 전체 성능 향상의 90%를 차지하는 핵심 병목임을 입증하였다. 또한, 단계별 비교 분석을 통해 파이프라인 최적화가 처리량 증가에도 불구하고 CPU 온도를 2.7℃ 감소시킴을 확인하였다. 본 연구는 NPU 가속 엣지 AI 시스템 설계 시 파이프라인 균형을 고려한 단계별 최적화 가이드라인을 제공한다.

**[Abstract]**

In this study, we quantitively analyze the impact of preprocessing and postprocessing stages on system performance and thermal characteristics in NPU-accelerated edge AI pipelines. A hybrid parallelization strategy was implemented by combining ARM NEON SIMD and OpenMP on Raspberry Pi 5 equipped with Hailo-8 accelerator running YOLOv8. The proposed methodology achieved 2.26× speedup (24.2 FPS) and 1.77× improvement in energy efficiency when compared with thebaseline system. Notably, preprocessing optimization alone yielded 2.14× performance gain, accounting for 90% of total improvement and identifying preprocessing as the dominant bottleneck. Through a comparative analysis of three optimization scenarios (baseline, preprocessing-only, full optimization), we quantified the contribution of each stage and demonstrated that pipeline balancing reduces CPU temperature by 2.7 °C despite increased throughput. This research provides stage-specific optimization guidelines for NPU-accelerated ARM-based edge AI systems.

**색인어 :** NPU 가속, 전·후처리 병목, 엣지 AI 가속, 하이브리드 병렬화, YOLOv8
**Keyword :** NPU Acceleration, Pre- and Post-Processing Bottleneck, Edge AI Acceleration, Hybrid Parallelization, YOLOv8

**http://dx.doi.org/10.9728/dcs.2026.27.3.813**



**Received** 29 January 2026; **Revised** 23 February 2026
**Accepted** 05 March 2026

***Corresponding Author; Young-Jin Suh**
**Tel:** ※ 개인정보 표시제한
**E-mail:** valentis@gmail.com

## Ⅰ. 서 론

### 1-1 연구 배경 및 엣지 AI(Edge AI)의 중요성

인공지능(AI) 기술의 발전으로 엣지 AI의 중요성이 증대되고 있다[1]. 엣지 AI 시장은 2025년 90억 달러에서 2030년 496억 달러로 성장할 것으로 예상되며[2], 저전력 디바이스에 최첨단 모델을 배포하려면 세심한 최적화가 필수적이다[3].

### 1-2 문제 정의 및 기존 연구의 한계

객체 인식과 같은 AI 작업은 (1) 전처리, (2) 추론, (3) 후처리 단계로 구성된다. 기존 연구는 주로 NPU 하드웨어나 추론 엔진 최적화에 집중해왔다[3],[4]. 그러나 Hailo-8 같은 NPU가 추론을 가속해도, 전·후처리 단계는 여전히 병목으로 남아있으며 전체 파이프라인 시간에서 불균형적으로 높은 비중을 차지한다. 이는 병렬화되지 않은 구간이 전체 성능을 제한하는 암달의 법칙(Amdahl's law)과 일치한다. 또한 높은 성능 추구는 열 스로틀링으로 이어져 지속적인 시스템 신뢰성을 저하시킨다[5].

### 1-3 연구 목표 및 주요 기여

본 연구는 이러한 한계를 극복하기 위해 라즈베리 파이 5와 Hailo-8 환경에서 YOLOv8 기반 시스템에 ARM NEON과 OpenMP를 결합한 하이브리드 병렬화를 적용하여, 전·후처리 단계의 병목 기여도를 정량화하고 성능-전력-열 특성 간 상호 의존성을 분석함으로써 응용별 최적화 우선순위 기준을 제시하고자 한다.

### 1-4 논문의 구성

본 논문은 다음과 같이 구성된다. 2장에서 관련 연구를 검토하고, 3장에서는 병목 분석 프레임워크 설계를, 4장은 이를 검증하기 위한 최적화 구현을 설명한다. 5장에서는 실험 결과를 분석하며, 6장에서 결론을 제시한다.

## Ⅱ. 관련 연구(Related Work)

### 2-1 객체 인식 및 엣지 AI 가속 기술

객체 인식 기술은 딥러닝의 도입으로 비약적으로 발전하였다. 특히 1-Stage Detector인 YOLO 시리즈는 속도와 정확도의 균형을 통해 실시간 처리에 최적화되어 있다[6],[7]. 본 연구에서 사용된 YOLOv8은 Anchor-free 구조를 채택하여 NMS 효율을 개선하고 다양한 비전 작업을 지원한다. 한편, 엣지 환경에서는 NPU(Neural Processing Unit)가 핵심 요소로 부상하고 있다. NPU는 행렬 연산에 특화되어 CPU 대비 월등한 전력 효율을 제공하지만[8], 전·후처리는 여전히 호스트 CPU가 담당해야 하므로 이기종 프로세서 간의 효율적인 워크로드 분배가 필수적이다[4],[5].

### 2-2 ARM NEON 및 OpenMP 병렬화

제한된 엣지 자원 활용을 위해 SIMD(Single Instruction, Multiple Data) 기술인 ARM NEON과 멀티스레딩 API인 OpenMP가 널리 사용된다. NEON은 128-bit 레지스터를 통해 픽셀 단위 병렬 연산을 수행하며[9], OpenMP는 멀티코어 환경에서 작업 수준 병렬화를 지원한다[10]. 기존 연구는 주로 모델 추론 병렬화에 집중했으나, 본 연구는 ARM NEON의 데이터 수준 병렬화와 OpenMP의 작업 수준 병렬화를 계층적으로 결합한 하이브리드 병렬화 방식을 채택하여, 전·후처리 단계의 상대적 병목 기여도를 수치적으로 비교할 수 있는 분석 프레임워크를 제시한다. 이 통합 접근 방식은 CPU의 코어 병렬성과 명령어 수준 병렬성을 동시에 활용함으로써, 라즈베리 파이 5와 같은 제한된 환경에서도 실시간 AI 처리 성능을 극대화하는 중요한 기술적 기반을 제공한다.

### 2-3 전·후처리 병목에 관한 기존 연구

엣지 AI 최적화에서 전·후처리 병목은 최근 주목받기 시작했다. NEON을 이용한 연구들은 주로 신경망 연산 가속[11]에 집중하고 있으며, 전처리 단계에 NEON을 적용한 사례[12]는 있으나 전·후처리를 체계적으로 분석한 연구는 부족하다. Jeong et al.[13]은 NVIDIA Jetson에서 TensorRT를 사용한 추론 최적화를 연구했으나, CPU 기반 전·후처리는 다루지 않았다.

NPU 환경에서의 연구는 더욱 제한적이다. K. Hsu, et al.[14]은 Google Coral TPU에서 전처리 병목을 지적했으나, CUDA/OpenCL 기반의 프레임워크를 제안했다. 반면, 본 연구는 ARM NEON과 OpenMP를 결합한 하이브리드 병렬화를 통해 전·후처리 각각의 기여도(90.4% vs 9.6%)를 정량적으로 분리하고, 최적화 우선순위를 명확히 제시한다는 점에서 차별화된다.

## Ⅲ. 병목 분석 프레임워크 설계

본 연구의 핵심 목표는 NPU 가속 환경에서 전처리와 후처리 중 어느 단계가 지배적 병목인지를 정량적으로 규명하는 것이다. 이를 위해 각 단계를 독립적으로 최적화하고 그 영향을 측정할 수 있는 분석 프레임워크를 설계하였다.

### 3-1 분석 대상 파이프라인 구조

본 연구는 YOLOv8 기반 객체 인식 파이프라인을 대상으로 하며, 다음과 같이 5단계로 구성된다:

(1) 캡처: V4L2 API를 통한 640×480 YUYV 영상 획득
(2) 전처리: YUV→RGB 변환, 리사이즈, 정규화, NCHW 재배치

(3) 추론: Hailo-8 NPU를 통한 YOLOv8 모델 실행
(4) 후처리: Bounding Box 디코딩, 신뢰도 필터링, NMS
(5) 출력: 결과 시각화 및 저장

그림 1은 전체 파이프라인의 데이터 흐름과 각 단계 간 의존성을 보여준다.

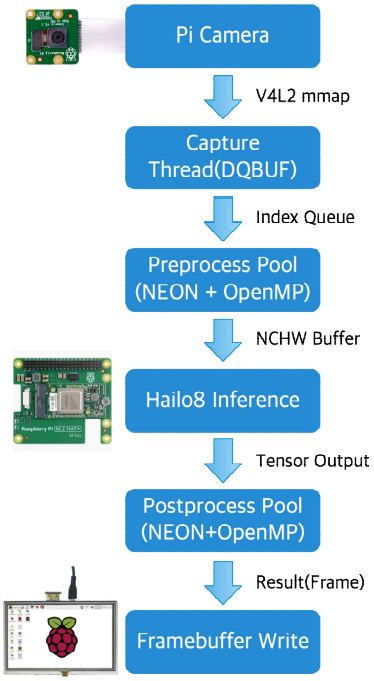


**그림 1.** 제안된 전체 시스템 아키텍처 및 데이터 흐름
**Fig. 1.** Overall system architecture and data flow

### 3-2 병목 측정 방법론

각 단계의 병목 기여도를 정량화하기 위해 다음 세 가지 시나리오를 설계하였다:

- Baseline: 전·후처리 모두 C++ 구현(최적화 없음)
- Pre-only: 전처리만 NEON+ OpenMP 최적화
- Full: 전·후처리 모두 NEON+ OpenMP 최적화

주요 설계 목표는 다음과 같다:

- Zero/Low-copy: V4L2 mmap 버퍼를 활용해 캡처 데이터가 전처리 단계로 전달될 때 메모리 복사를 최소화한다.
- 파이프라인 독립성: 각 단계는 독립적으로 구성하여 처리량(throughput)을 극대화한다.
- 재현성/계측성: CLOCK_MONOTONIC_RAW을 clock_gettime() 함수의 인자로 사용하여 나노초 단위로 타이밍을 측정하고, 각 단계의 기여도는 식 (1)과 같이 계산한다. 여기서 $FPS_{Baseline}$은 최적화를 적용하지 않은 기준 시나리오, $FPS_{StageOnly}$는 특정 단계(예: Pre-only)만 최적화한 시나리오, $FPS_{Full}$은 전체 최적화 시나리오의 초당 프레임 수를 각각 의미한다:

$$Contribution_{Stage} = \frac{FPS_{StageOnly} - FPS_{BaseLine}}{FPS_{Full} - FPS_{BaseLine}} \times 100(\%) \quad (1)$$

이 기여도 정의는 전처리·후처리 최적화 효과가 독립적으로 선형 합산된다고 가정한다. Pre-only와 Full 간 추가 향상폭이 전체 향상 대비 약 10% 미만이며(표 2, 표 3), CPU 온도·전력 차이도 제한적인 수준에 머무르는 점을 고려할 때, 본 실험 범위에서는 전·후처리 기여도를 선형 합으로 근사하는 것이 타당하다고 판단된다.

### 3-3 평가 지표 정의

병목 분석을 위해 다음의 평가 지표를 측정한다:

(1) 처리 속도: FPS (Frames Per Second)
  - 측정 범위: 캡처부터 후처리 완료까지의 E2E 지연 시간
(2) 에너지 효율: FPS/W (Frames Per Watt)
  - 측정 방법: FNB58 전력계를 통한 100sps 샘플링
(3) 발열 특성: CPU/NPU 온도 (℃)
  - CPU: /sys/class/thermal/thermal_zone*/temp 파일을 통해 1초 간격으로 샘플링
  - NPU: ST490+ 열화상 카메라를 통해 칩 표면 온도를 측정하였다. 측정의 일관성을 위해 실험은 실내 온도 24±1℃의 환경에서 수행하였으며, 각 시나리오는 10분간 워밍업 후 10분간 계측하여 온도 및 전력의 정상상태 값을 확보하였다. NPU 표면 온도 측정을 위해 Hailo-8 칩 전체가 열화상 계측 내에 포함되도록 하는 최소 거리인 6cm로 고정하고, 방사율을 0.95로 설정한 상태에서, Hailo-8 패키지 중앙 지점만을 대상으로 계측하였다. 표면 온도는 칩 내부 온도보다 10-15℃ 낮을 수 있으나[15], 시나리오 간 상대적 비교에는 충분히 유효하다.

본 프레임워크는 단순히 속도만이 아닌, 에너지와 열을 포함한 다차원적 병목 분석을 가능하게 한다.

## Ⅳ. 실험적 검증을 위한 최적화 구현

본 장에서는 Ⅲ장에서 설계한 병목 분석 프레임워크를 실증하기 위해 구현한 최적화 기법을 설명한다.

### 4-1 실험 환경

- 하드웨어: 라즈베리 파이 5 (Cortex-A76 쿼드코어) + Hailo-8 NPU(26 TOPS, PCIe Gen-3.0) + Pi Camera v2 + 액티브 쿨러

- 소프트웨어 : 소프트웨어 스택은 Table 1과 같다. 라즈베리 파이 OS(Linux) 상에서 C++로 구현하였으며, HailoRT SDK와 OpenMP, ARM NEON Intrinsics를 활용하였다. 특히 범용 라이브러리(OpenCV)의 오버헤드를 제거하고 메모리 제어권을 확보하기 위해 커스텀 파이프라인을 구축하였다(표 1 참조).

**표 1.** 제안된 전체 시스템의 소프트웨어 스택
**Table 1.** The proposed system's software stack

| Category | Specification / Version |
|---|---|
| Operating System | Linux (Raspberry Pi OS) |
| Development Environment | g++ (12.2.0/AARCH64) |
| Core APIs & Library | V4L2 (mmap), HailoRT SDK, OpenMP, ARM NEON Intrinsics |

- 컴파일 옵션: 전체 시나리오에 -O3 -lhailort 옵션 사용, 병렬화 적용 시 -fopenmp -march=armv8-a+ simd, Baseline은 명시적 SIMD/OpenMP 코드를 사용하지 않은 표준 C++ 구현이며, -fno-tree-vectorize로 컴파일러 자동 벡터화를 비활성화하여 제안 기법(NEON+OpenMP)과의 명확한 성능 차이를 측정하였다. 이는 컴파일러 최적화에 의존하지 않고 개발자가 능동적으로 SIMD를 활용할 때의 효과를 정량화하기 위함이다.

### 4-2 영상의 캡처(입력)

V4L2 API를 사용하여 Pi Camera v2로부터 640×480 해상도의 저지연 영상 스트림을 획득한다. 카메라 대역폭 및 호환성을 고려하여 영상 포맷으로 YUYV(YUV 4:2:2)를 사용하며, 버퍼링 방식으로는 오버헤드가 가장 낮은 MMAP 기반 방식을 채택하였다.

비디오 캡처는 select() API를 통해 버퍼 가용성을 확인한 후, VIDIOC_DQBUF를 호출하여 프레임을 획득한다. 이 시점에 타임스탬프를 기록하고, 해당 프레임 인덱스를 전처리 워커 큐로 전달하여 파이프라인 처리를 시작한다.

효율적인 ARM NEON 연산을 위해, 전처리 및 모델 입력 버퍼는 posix_memalign() 함수를 사용하여 64-bit 경계에 맞게 메모리 정렬(Memory Alignment)을 수행하였다.

### 4-3 전처리 최적화 구현

전처리는 캡처된 YUYV 원본 데이터를 Hailo-8의 입력 레이아웃(NCHW float32 또는 quantized int8)으로 변환하는 과정이다. 이 단계의 효율성은 전체 시스템의 초당 프레임 수(FPS)를 결정하는 주요 병목 요소이다.

본 연구에서는 이 전처리 경로를 NEON 및 OpenMP 기반으로 최적화하여, 전처리가 전체 파이프라인 지연 시간과 에너지 효율에 미치는 영향을 정량적으로 평가한다.

전처리는 YUYV→RGB 변환, 리사이즈, 정규화, NCHW 재배치로 구성된다. YUV-to-RGB 변환은 다음 수식을 따른다.

$$\begin{aligned} R &= 1.164(Y-16)+1.596(V-128) \\ G &= 1.164(Y-16)-0.813(V-128)-0.391(U-128) \\ B &= 1.164(Y-16)+2.018(U-128) \end{aligned} \quad (2)$$

부동 소수점 연산은 스칼라 처리 시 상당한 사이클을 소모하여 병목을 유발한다. 본 연구는 YUV-to-RGB 변환을 ARM NEON으로 벡터화하여 전처리 병목을 효과적으로 해소하였다.

전처리의 마지막 단계는 정규화(Normalization) 및 레이아웃 변환이다. YOLOv8 모델은 입력 이미지를 640×640의 고정된 크기로 표준화해야 한다. 이를 위해 이미지의 크기를 조정하고 남은 영역을 패딩 처리한다. 이후 RGB 픽셀 값을 다음의 식 (3)과 같이 정규화한다.

$$P_{\text{norm}} = \frac{(P_{\text{pixel}}/255.0 - \mu)}{\sigma} \quad (3)$$

마지막으로, 정규화된 픽셀 데이터를 모델이 요구하는 NCHW(Channel-First) 형식으로 메모리를 재배치한다. 이 모든 과정에서 ARM NEON Intrinsics를 활용하면, 메모리 정렬 및 버스트 로드/저장 작업을 극대화하여 이미지 픽셀 배열 처리를 효율적으로 병렬 수행함으로써 최종 처리량을 최대화할 수 있다.

- NEON 최적화: float32x4_t 레지스터로 4픽셀 병렬 처리
- OpenMP 최적화: #pragma omp parallel for로 행 단위 분산

### 4-4 후처리 최적화 구현

후처리는 Hailo-8로부터 획득한 모델 출력 텐서를 최종 Bounding Box 결과로 복원하고 시각화하는 단계이다. 후처리 단계에 대해서도 유사한 병렬화를 적용함으로써, 전처리와 후처리 간 최적화 수준 차이가 전체 성능에 미치는 상대적 기여도를 비교할 수 있도록 한다. 후처리는 Bounding Box 디코딩, 신뢰도 필터링, NMS로 구성된다.

- NEON 최적화: IoU 계산을 vmaxq_f32/vminq_f32로 벡터화
- OpenMP 최적화: 박스별 독립적인 작업을 코어에 분산

## V. 실험 결과 및 병목 분석

### 5-1 실험 시나리오

각 시나리오(Baseline, Pre-only, Full)를 각각 10회 독립 실행하였다(매 실행마다 10분 워밍업 후 10분 측정). YOLOv8s(11.2M 파라미터)와 COCO 데이터셋을 사용하였으며, 재현성 확보를 위해 Optical HIL 시뮬레이션을 구축하였다. 실제 엣지 AI 응용 환경을 위하여 분주하게 사람과 자

동차가 가장 많이 다니는 장소인 뉴욕 타임 스퀘어의 'EarthCam Live: Times Square 4K' 실시간 스트리밍의 15분간 구간을 녹화하여 고해상도 디스플레이로 동일한 부분을 재생하고 Pi Camera가 재수집하는 방식을 적용하였다.

본 실험은 디스플레이 화면 재촬영 방식을 채택하였으며, 이 과정에서 디스플레이 주사율과 카메라 센서(기본 Auto Exposure 모드) 간의 비동기화로 인한 미세한 노출 변화나 모아레(Moire) 현상이 발생할 수 있다. 그러나 카메라의 수평 및 초점을 물리적으로 고정하고, 이러한 스트리밍 입력 조건(왜곡 요인 포함)을 Baseline과 최적화(Pre-only, Full) 시나리오에 완벽히 동일하게 주입되도록 통제하였다. 따라서 캡처 단계의 화질적 변인이 각 파이프라인 단계의 연산량에 미치는 영향은 상수화(Constant)되며, 무엇보다 동일한 입력 경로가 Baseline 및 모든 최적화 시나리오에 공통으로 적용되므로, 시나리오 간 상대적 성능 차이 분석에 체계적 편향을 유발하지 않는다.

## 5-2 성능 측정 결과

### 1) 처리 속도

Full 최적화 시 24.2 FPS를 기록하여 Scalar Baseline (10.7 FPS) 대비 2.26배의 성능 향상을 달성했다. 특히 전처리 단독 최적화만으로도 22.9 FPS(2.14배 향상)를 기록했는데, 이는 YUV 변환 및 리사이징 단계에서 SIMD 병렬화가 주효했음을 시사한다.

각 시나리오별 성능 측정 결과는 표 2와 같다.

### 2) 전력 및 발열

병렬화로 인한 CPU 활용률 증가로 전력 소모는 9.0W에서 11.5W로 약 27.8% 증가하였으나, 전력 효율(FPS/W)은 1.19에서 2.10으로 1.77배 개선되었다.

전처리만 최적화한 경우(65.5℃)가 전·후처리 모두 최적화한 경우(62.8℃)보다 CPU 온도가 약 2.7℃ 더 높게 측정되었다. 이는 최적화 시 파이프라인 균형으로 CPU 유휴 구간이 늘어나 동적 전압·주파수 조절이 효과적으로 작동한 결과로 분석된다. 모든 최적화 시나리오에서 CPU 온도는 스로틀링 임계값(80℃) 대비 안정적인 범위를 유지하여 지속적인 고성능 추론이 가능함을 확인하였다.

## 5-3 병목 기여도 및 심층 분석

### 1) 전처리의 지배적 영향

식 (1)에 대입하여 산출한 기여도는 전처리 90.4%, 후처리 9.6%로 나타났다. 이는 본 실험 파이프라인에서 전처리가 지배적 병목임을 의미한다.

### 2) 암달의 법칙 관점

본 파이프라인에서 병렬화 개선 가능 부분의 상대적 크기는 전처리가 후처리에 비해 약 9.4배(90.4% / 9.6%) 더 크며, 이는 암달의 법칙에 따라 전처리 최적화의 효과가 훨씬 큼을 설명한다. NPU가 추론을 가속해도, 병렬화되지 않은 전처리가 전체 성능을 지배한다. 따라서 엣지 AI 최적화 시 전처리에 우선적 자원을 배분해야 한다.

그림 2는 세 시나리오의 FPS와 온도 분포를 시각화한다.

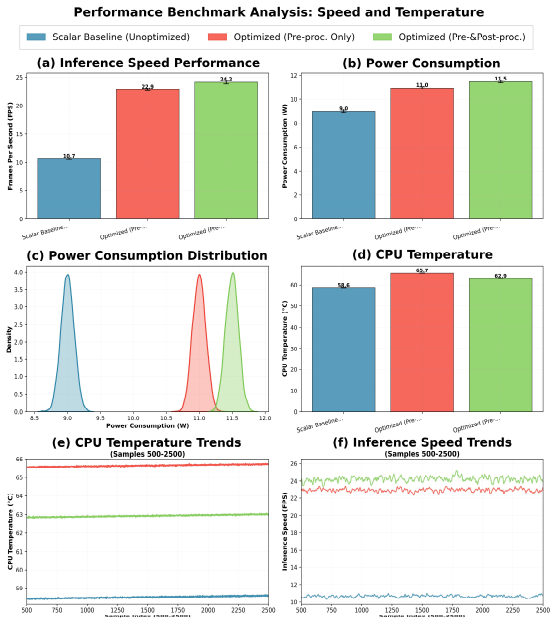


**그림 2.** 측정 값의 분포도 그래프(속도와 온도)
**Fig. 2.** Measured value distribution: Speed and temperature

**표 2.** Raspberry Pi 5 기반 YOLOv8s 객체 탐지 성능 비교 결과
**Table 2.** YOLOv8s object detection performance comparison results

| Evaluation Metric | Baseline (Unoptimized) | Optimized (Pre-only) | Optimized (Full) | Gain |
|---|---|---|---|---|
| Inference Speed (FPS) (Mean±SD) | 10.7 (±0.315) | 22.9 (±0.403) | 24.2 (±0.621) | ~2.26× speedup via SIMD/OpenMP |
| Power Consumption (W) | 9.0 | 11.0 | 11.5 | 22.2% (Pre-only) / 27.8% (Full) increase due to SIMD/OpenMP usage |
| CPU Temperature (℃) (Mean±SD) | 58.4 (±0.976) | 65.5 (±1.106) | 62.8 (±0.805) | Attributed mainly to increased multi-core utilization |
| NPU Temperature (℃) | 35.7 | 36.3 | 36.9 | Correlated with increased throughput |

*Statistical significance was assessed using paired t-tests (n=10 independent runs). Both optimized configurations showed significant improvements over baseline ($p < 0.01$).

실험 결과, 전처리 단계가 전체 지연 시간의 가장 큰 비중을 차지하는 지배적 병목임이 확인되었다. 특히 전처리만 최적화한 경우에도 Baseline 대비 2.14배의 속도 향상을 보인 반면, 후처리까지 포함한 전체 최적화는 추가적인 5% 내외의 성능 향상에 그쳤다. 이는 본 실험 설정에서 전처리(YUV 변환, 리사이즈, 정규화)가 전체 파이프라인 지연 시간에 훨씬 더 지배적인 영향을 미치며, 후처리 최적화는 보조적인 역할을 수행함을 시사한다.

전처리만 최적화했을 경우 NPU 처리량은 증가하지만 후처리 병목으로 인해 CPU 온도가 오히려 상승하는 현상이 관찰되었다. 이는 엣지 AI 최적화 시 단순 속도 향상 기술(SIMD 등)의 적용뿐만 아니라, 파이프라인 전체의 균형(Balancing)을 고려한 단계별 최적화 전략이 필수적임을 시사한다.

이는 암달의 법칙 관점에서, 본 실험 설정의 파이프라인에서 병렬화로 개선 가능한 부분(전처리)의 비중이 후처리에 비해 압도적으로 큼을 의미한다.

#### 3) 파이프라인 단계별 시간 분해

표 3은 각 시나리오별 파이프라인 단계의 평균 실행 시간을 보여준다. 표 3의 스테이지별 시간은 단일 대표 세션의 평균값이며, 표 2의 E2E FPS는 10회 실행의 전체 평균이다. 측정 방식 차이로 절대값은 다르지만, 전처리/후처리의 상대적 비중과 최적화 효과는 일관된 경향을 보인다.

**표 3.** 파이프라인 단계별 평균 실행 시간(ms/frame)
**Table 3.** Average execution time per pipeline stage (ms/frame)

| Stage | Baseline (Unoptimized) | Optimized (Pre-only) | Optimized (Full) |
|---|---|---|---|
| Capture | 2.3 | 2.3 | 2.3 |
| Preprocessing | 78.5 | 36.2 | 35.8 |
| Inference | 8.7 | 8.7 | 8.7 |
| Postprocessing | 3.8 | 3.8 | 2.5 |
| Display | 0.2 | 0.2 | 0.2 |
| Total Latency (E2E) | 93.5 | 51.2 | 49.5 |

*Note: Capture, Inference, and Display times are denoted as identical values due to negligible differences (< 0.1 ms) across scenarios. This table shows a single representative session's values to identify trends; please refer to Table 2 for statistical analysis.

표 3은 전체적인 경향성 분석을 위해 전체 실험 중 중앙값에 해당하는 대표 세션을 선정하여 상세 시간을 도출한 것이다. 표 2의 FPS가 10회 반복 측정의 전체 평균을 나타낸다면, 표 3은 파이프라인 내부의 세부 병목 지점을 시각화하기 위해 각 스테이지별 지연 시간을 분해하여 제시한 결과이다. 두 지표 모두 최적화 전후의 성능 향상 비율에서 일관된 경향을 보임을 확인하였다. Pre-only 대비 Full에서 전처리 시간이 0.4ms 감소한 것은 측정 편차 범위 내의 변동으로, 통계적으로 유의미한 차이가 아니다.

Baseline에서 전처리가 전체 시간의 84%(78.5/93.5)를 차지하며, NPU 추론(9%)과 후처리(4%)를 압도한다. Pre-only 최적화 시 전처리 시간이 54% 감소(78.5→36.2ms)하여 전체 성능을 2.14배 향상시켰다. Full 최적화 시 후처리 시간이 추가로 34% 감소(3.8→2.5ms)하였으나, 전체 시간에서 차지하는 비중이 작아 전체 성능 향상은 5% 수준에 그쳤다. 이는 전처리가 지배적 병목이라는 본 연구의 핵심 발견을 정량적으로 뒷받침한다.

#### 4) 응용별 최적화 권장사항

표 4는 응용별 권장 구성을 요약한다.

**표 4.** 응용 시나리오별 최적화 권장 구성
**Table 4.** Recommended optimization configurations by application

| Application | Config | Priority | Trade-off |
|---|---|---|---|
| Real-time CCTV | Full | Speed | +27.8% Power |
| Battery Device | Pre-only | Efficiency | 93% of max speed |
| Thermally Constrained Device | Full | Thermal | Balanced |

본 연구 결과를 바탕으로 엣지 AI 응용별 최적화 전략을 다음과 같이 제안한다:

(1) 실시간 감시 시스템: 처리 속도가 최우선이므로 전·후처리 전체 최적화를 권장한다. 전력 소모 27.8% 증가는 AC 전원 환경에서 수용 가능하며, 2.26배 속도 향상으로 24.2 FPS를 달성하여 실시간 요구사항(15-20 FPS)을 만족한다.

(2) 배터리 기반 디바이스: 전력 효율이 중요하므로 전처리 단독 최적화를 권장한다. 전력 소모 22.2% 증가로 2.14배 속도 향상을 달성하며, 에너지 효율(FPS/W)도 Baseline 대비 약 1.75배 개선된다.

(3) 열 제약 환경: 방열 설계가 제한적인 경우 전체 최적화를 권장한다. 파이프라인 균형으로 CPU 온도가 Pre-only 대비 2.7℃ 낮아져 장기 안정성이 향상된다.

### 5-4 논의 및 해석

#### 1) 전처리 지배성의 원인

본 실험에서 전처리가 지배적 병목으로 나타난 주요 원인은 다음과 같다:

첫째, 해상도 변환 부담이다. 640×480(307K 픽셀)을 640×640(410K 픽셀)으로 변환하는 과정에서 바이큐빅 보간법(bicubic interpolation)을 사용하며, 각 출력 픽셀마다 16개의 입력 픽셀을 참조하는 계산 집약적 작업이다. 실시간성이 더 중요한 응용에서는 쌍선형 보간으로 대체하여 연산 부하를 경감할 수 있으며, 이는 향후 연구 과제로 남긴다.

둘째, 색 공간 변환이다. YUYV→RGB 변환은 픽셀당 3개의 부동소수점 곱셈과 2개의 덧셈을 요구하며, 640×480 해

상도에서 약 $4.3 \times 10^6$회 이상의 연산이 필요하다.

셋째, 메모리 재배치다. HWC(Height-Width-Channel)에서 NCHW(Channel-Height-Width)로의 변환은 비연속적 메모리 접근을 유발하여 캐시 미스율을 증가시킨다. 반면 후처리는 평균 10-15개의 객체만 처리하므로 계산량이 상대적으로 작다.

#### 2) 방법론의 확장성

본 연구의 하이브리드 병렬화 방법론은 엣지 LLM 토큰화, 음성 인식 MFCC 추출 등 다양한 영역으로 확장 가능하다.

#### 3) 연구의 적용 범위와 일반화 가능성

객체 수가 100개를 넘는 혼잡한 장면에서는 NMS 복잡도가 증가하여 후처리 비중이 커질 수 있다. 본 실험의 Times Square 장면(평균 10-15개 객체)과 같은 일반적인 감시 응용에서는 전처리가 우선순위이나, 밀집 환경에서는 후처리 최적화도 중요해질 수 있다.

## Ⅵ. 결론 및 향후 연구

### 6-1 주요 기여

본 연구는 다음과 같은 주요 성과를 달성하였다:

(1) NPU 가속 환경에서 전·후처리 병목을 정량적으로 분석하는 프레임워크를 개발하고, 라즈베리 파이 5 + Hailo-8 기반 YOLOv8 시스템에 적용하였다.

(2) ARM NEON과 OpenMP 결합으로 2.26배 속도 및 1.77배 에너지 효율 향상을 달성하였으며, 전체 성능 향상의 대부분(약 90%)이 전처리 최적화에서 기인함을 보였다. 이는 엣지 AI 시스템 설계 시 전처리에 우선적 자원 배분이 필요함을 시사한다.

(3) 성능-전력-열 특성 분석을 통해 응용별 최적화 가이드라인을 제시하였다.

### 6-2 확장 방향과 향후 연구

본 연구는 다음과 같은 한계를 가진다:

(1) 입력 조건 의존성: 640×480→640×640 리사이즈 설정에서 약 1.33배 업스케일링과 패딩 과정의 보간(Interpolation) 연산 부하에 대한 전처리 비중이 컸다. 입력 해상도가 모델 입력과 유사한 경우(예: 640×640 입력) 리사이즈 부담이 감소하여 전처리 비중이 줄어들 수 있다. 또한 검출 객체 수가 많은 경우(예: 100개 이상) NMS 부하로 후처리 비중이 증가할 가능성이 있다.

(2) 단일 모델 검증: YOLOv8s에 국한되어 다른 객체 인식 모델(YOLOv5, v7, v10, v11 등)에 대한 일반화 가능성은 추가 검증이 필요하다. 특히 Transformer 기반 모델(DETR 등)은 후처리 복잡도가 다를 수 있다.

(3) 특정 하드웨어 종속: Hailo-8 NPU 환경에서만 검증되었으며, Google Coral(8/16 TOPS), Intel Movidius(2.5 TOPS) 등 성능이 다른 NPU에서의 효과는 미검증이다.

(4) 제한된 작업 영역: 객체 탐지에 국한되며, 인스턴스 세그멘테이션(Mask R-CNN, YOLACT), 포즈 추정(OpenPose), 의미론적 세그멘테이션(DeepLab) 등 후처리 복잡도가 높은 작업에 대한 적용은 향후 연구가 필요하다.

(5) Zero-copy 미구현: DMABUF 기반 완전 zero-copy는 본 연구에서 구현되지 않아, 메모리 복사 오버헤드 제거로 추가적인 5-10% 성능 향상 여지가 남아있다.

향후 연구에서는 다양한 입력 해상도와 모델(YOLOv5/v7/v10/v11, DETR), NPU(Coral, Movidius, Rockchip NPU), 비전 작업(세그멘테이션, 포즈 추정)에 본 방법론을 적용하여 일반화 가능성을 검증할 필요가 있다. 그리고 실제 현장 카메라 입력에 대한 검증과 보간법별 성능-품질 트레이드오프 비교는 향후 연구 과제로 남긴다.

## 참고문헌

[1] R. Singh and S. S. Gill, “Edge AI: A survey,” *Internet of Things and Cyber-Physical Systems,* Vol. 3, pp. 71-92, 2023. https://doi.org/10.1016/j.iotcps.2023.02.004

[2] MarketsandMarkets. Edge Computing Market Size, Share Industry Analysis [Internet]. Available: https://www.marketsandmarkets.com/Market-Reports/edge-computing-market-133384090.html.

[3] A. Fanariotis, T. Orphanoudakis, K. Kotrotsios, V. Fotopoulos, G. Keramidas, and P. Karkazis, “Power-Efficient Machine Learning Models Deployment on Edge IoT Devices,” *Sensors*, Vol. 23, No. 3, 1595, 2023. https://doi.org/10.3390/s23031595

[4] N. Mayumu, “A Survey of Deep Learning at the Edge Computing,” TechRxiv, 2022. https://doi.org/10.36227/techrxiv.19640265.v1

[5] A. Ahmadi, H. A. Abdelhafez, K. Pattabiraman, and M. Ripeanu, “EdgeEngine: A Thermal-Aware Optimization Framework for Edge Inference,” in *Proceedings of the Eighth ACM/IEEE Symposium on Edge Computing,* Wilmington: DE, pp. 67-79, 2023. https://doi.org/10.1145/3583740.3626616

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas: NV, pp. 779-788, 2016. https://doi.org/10.48550/arXiv.1506.02640

[7] Ultralytics. YOLOv8 Documentation, Discover Object Detection Evolution: YOLO to YOLO11 [Internet].

Available: https://www.ultralytics.com/ko/blog/the-evolution-of-object-detection-and-ultralytics-yolo-models.

[8] Hailo Technologies. Hailo-8 AI Accelerator Datasheet [Internet]. Available: https://hailo.ai/products/ai-accelerators/Hailo-8-ai-accelerator.

[9] A. Maciá-Lillo, S. Barrachina, G. Fabregat, and M. F. Dolz, "Optimizing Convolutions for Deep Learning Inference on ARM Cortex-M Processors," *IEEE Internet of Things Journal*, Vol. 11, No. 15, pp. 26203-26219, 2024. https://doi.org/10.1109/JIOT.2024.3395335

[10] OpenMP ARB. OpenMP API Specification for Parallel Programming [Internet]. Available: https://www.openmp.org/specifications.

[11] M. A. Al Jbaar and S. A. Dawwd, "SIMD Implementation of Deep CNNs for Myopia Detection on a Single-Board Computer System," *Eastern-European Journal of Enterprise Technologies*, Vol. 5, No. 9, pp. 98-108, 2023. https://doi.org/10.15587/1729-4061.2023.289007

[12] J. Song, et al., "Face Recognition: The World's Fastest and Most Accurate Walkthrough," *Kakao Enterprise AI Report 2022*, Vol. 1, 2022. https://kakaoenterprise.com/wp-content/uploads/2023/01/Kakao-Enterprise-AI-Report-2022.pdf

[13] E. Jeong, J. Kim, and S. Ha, "TensorRT-Based Framework and Optimization Methodology for Deep Learning Inference on Jetson Boards," *ACM Transactions on Embedded Computing Systems (TECS),* Vol. 21, No. 5, 51, pp. 1-26, 2022. https://doi.org/10.1145/3508391

[14] K.-C. Hsu and H.-W. Tseng, "Accelerating Applications Using Edge Tensor Processing Units," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis,* St. Louis: MO, *pp. 1-14, 2021.* https://doi.org/10.1145/3458817.3476177

[15] J. E. Galloway, S. Bhopte, and C. Nelson, "Characterizing Junction-to-Case Thermal Resistance and Its Impact on End-Use Applications," in *Proceedings of the 13th InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems,* San Diego: CA, pp. 1342-1347, 2012. https://doi.org/10.1109/ITHERM.2012.6231576

**서영진(Young-Jin Suh)**

2005년 : 숭실대학교 대학원 (공학석사-컴퓨터구조)
2009년 : 숭실대학교 대학원 (공학박사수료-컴퓨터구조)
2020년 9월～2002년 2월: 새롬기술 새롬기술연구소 연구원
2005년 8월～2006년 7월: 키네마스터(주) 책임연구원
2020년～현 재: 세종사이버대학교 소프트웨어공학과 겸임교수
※관심분야 : 사물인터넷(IoT), 딥러닝/인공지능(AI). 유비쿼터스 컴퓨팅(AR) 등

**장훈(Hoon Jang)**

1987년 : 서울대학교 (공학학사 - 전자공학과)
1987년 : 서울대학교 대학원 (공학석사 - 전자공학과)
1993년 : University of Texas at Austin (공학박사)
1991년: IBM Inc. Senior Member of Technical Staff
1993년: Motorola Inc. Senior Member of Technical Staff
1994년～현 재: 숭실대학교 컴퓨터학부 교수
※관심분야 : 통신, 컴퓨터, 신호처리, 반도체 등