FPGA-Enhanced Basketball Analytics System (FE-BAS)

Team Number: AOHW25_775

Allen Chellasamy
Electrical and Computer Engineering
Santa Clara University
achellasamy@scu.edu

Ronit Kumar
Electrical and Computer Engineering
Santa Clara University
rkumar4@scu.edu

August 28, 2025

Abstract

We present FE-BAS (FPGA-Enhanced Basketball Analytics System) focused on inference-only: (i) training and exporting YOLOv11 detectors, (ii) graph-level operator rewrites to increase AMD/Xilinx DPUCZDX8G compatibility, (iii) static INT8 QDQ post-training quantization purpose-built for ZCU104, and (iv) a portable ONNX Runtime (ORT) + Vitis-AI Execution Provider (VAIP) inference client with a batchable demo runner. The end-to-end toolchain yields trained weights, original and patched ONNX graphs, INT8 QDQ models, per-model compatibility reports, ready-to-run VAI compile scripts, and deployment JSON (board/DPU metadata, quant settings, performance estimates). HDMI input, multi-object tracking (DeepSORT), and scoreboard OCR are intentionally deferred to Future Work.

Contents

1 Introduction

- **Problem Statement:** Real-time sports analytics benefits from low-latency, power-efficient, on-device inference. Zynq UltraScale+ boards (e.g., ZCU104) accelerate common CNN ops on the DPUCZDX8G, but require export/graph surgery and INT8 quantization to minimize ARM CPU fallbacks.
- **Scope of This Report:** This phase delivers a repeatable train → export → patch → quantize → compile → run pipeline and a VAIP ONNX inference path on ZCU104. Video/HDMI, tracking, and OCR are out-of-scope here.
- **Research Question:** How to co-design YOLOv11 training/export and ONNX transforms so that (a) DPU coverage is maximized pre-compile, (b) INT8 QDQ quantization preserves accuracy and tool robustness, and (c) VAIP executes the resulting model with predictable throughput on ZCU104 with minimal CPU partitioning?

2 Background

YOLOv11 offers strong speed–accuracy trade-offs and a stable ONNX export path, though heavy backbones or heads can introduce operators that are awkward for DPUs. On ZCU104, the DPUCZDX8G is an INT8 accelerator with a limited operator set (e.g., Conv, BatchNorm, ReLU, pooling, Concat, and Resize), so models must be in INT8 QDQ form with DPU-friendly shapes and broadcast semantics. We use ORT static QDQ quantization, which inserts QuantizeLinear/DequantizeLinear around DPU-mappable subgraphs, selecting per-tensor INT8 for stability with Vitis AI. Because ORT quantizers prefer models with IR ≤ 10 and Vitis AI compilers are most robust with default-domain opset ≤ 13, we respect those constraints during export and optimization. Finally, VAIP—the ONNX Runtime Vitis-AI Execution Provider—offloads compiled subgraphs to the DPU on ZCU104 and caches artifacts on disk to enable faster warm starts.

3 Methods

3.1 Dataset & Labels

- **Source:** Roboflow — workspace `basketballyolo`, project `basketball-hoop-ball-and-player-` version 1 (classes include *player*, *ball*, *hoop*, court zones/numbers).
- **Format:** Consumed via supplied `data.yaml` (Ultralytics-compatible).

3.2 Environment & Dependencies

- Programmatic install: `ultralytics`, `roboflow`, `onnx`, `onnxruntime`, `onnx-graphsurgeon`, `numpy`, `torch`, `torchvision`.
- **Repro logging:** `yolo_dpu_training.log` (stream + file).

3.3 Training Configuration

- **Models:** `yolo11n`, `yolo11s`, `yolo11m`.
- **Hyperparameters:** 50 epochs, batch size 2, image size 640.
- **Export:** ONNX opset 17, `simplify=False`, `dynamic=False`.
- **Outputs:** `outputs/<model>_dpu/...`

3.4 Backbone Patching (PyTorch level)

- Preserve `Detect` head.
- Replace composite blocks (C3/PSA/SPPF/DFL/Attention/SiLU) with Conv–BN–ReLU sequences.
- Convert `ModuleList` → `Sequential` where appropriate.
- Log unsupported types before/after and replacement counts.

3.5 ONNX Export & Light Rewrites

- Export original ONNX from trained weights.
- **Graph surgery:**
  - $\text{Div}(x, c) \rightarrow \text{Mul}(x, 1/c)$
  - $\text{Sub}(x, c) \rightarrow \text{Add}(x, -c)$
  - Keep `Split`/`Slice` intact but flag them.
- Save as `<model>_patched.onnx` and maintain a patch notes list.

3.6 CPU Validation (ORT)

- Create ORT CPU session; run inference with dummy input $(1, 3, 640, 640)$.
- Record true input name/shape and log output tensor shapes.

3.7 DPU Compatibility Analysis & Report

- Compare operator histogram to DPU-friendly set: Conv/Relu/LeakyRelu/Pool/Add/Mul/BN/Co
- Treat `Split`/`Slice` as structural (not "hard unsupported").
- Emit `compatibility_report.txt` with pre/post-patch unsupported layers, layer type counts, ONNX sizes, and a pre-quant verdict.

3.8 Reproducibility & Packaging

- Package all model outputs plus `yolo_dpu_training.log` into `yolo11_dpu_models.zip`.
- Provide console summary of present files and DPU-compatibility flags.

3.9 Quantization Overview (ZCU104 DPUCZDX8G)

- Target `DPUCZDX8G_ISA0_B4096_MAX_BG2` (ZCU104).
- Use ORT static QDQ, per-tensor INT8 for activations/weights, `reduce_range=False`.

3.10 Calibration Data Reader & Preprocessing

- Auto-discover `~/basketball_vision/calib_images`; fallback to `./calib_images`.
- Load up to 300 images across common extensions.
- Letterbox to 640×640 (padding value 114); BGR→RGB; normalize to $[0, 1]$; NCHW; optional brightness jitter ($\sim$1/3 of images).

3.11 Model Validation & DPU-Oriented Optimizer

- **Validation:** List DPU-eligible vs. non-eligible op types; capture input tensor metadata.
- **IR handling:** If model IR > 10, create a temporary IR $= 10$ copy for ORT.
- **Optimizer transforms:** set batch $= 1$; normalize `Split` (attribute vs. second-input forms); drop `Reshape.allowzero`; clamp default-domain opset to 13; export `<model>_dpuczdx8g_optimized.onnx`.
- Log estimated DPU utilization (percentage of nodes on DPU).

3.12 Static QDQ Quantization (ORT)

- Apply `quantize_static` with `QuantFormat.QDQ` and `QuantType.QInt8` (activations/weights), `per_channel=False`.
- If needed, quantize on temporary IR10; restore original IR post-quant.
- Output `<model>_dpuczdx8g_int8.onnx`; re-validate with ORT CPU dummy input.

3.13 Fallback (No ORT Available)

- Copy optimized ONNX to `<model>_dpuczdx8g_ready.onnx` for later Vitis-side quantization.

3.14 Packaging for Compilation & Deployment

- **Compile script:** `compile_<model>.sh`:

```
vai_c_xir -x <model>_dpuczdx8g_int8.onnx \
  -a /opt/vitis_ai/compiler/arch/DPUCZDX8G/ZCU104/arch.json \
  -n <model>
```

- Produces `<model>.xmodel`.
- **Deployment JSON:** `<model>_deployment.json` (board/DPU info, quant settings INT8/QDQ/300 samples, file paths, performance estimates for n/s/m).

3.15 ONNX + VAIP Inference Client (ZCU104)

- **Script:** `deploy_onnx_vaip.py` (runs on the board).
- **Providers:** `("VitisAIExecutionProvider", {"log_level":"info","cache_dir":...})` with CPU fallback.
- **Preprocess:** PIL → letterbox(640,640) → normalize $[0, 1]$ → CHW → NCHW batch.
- **Postprocess:** supports raw YOLO head $[B, A, 5{+}C]$ or $[B, 5{+}C, A]$ (transpose if needed); overall class confidence = obj × class_prob; $(c_x, c_y, w, h) \to (x_1, y_1, x_2, y_2)$; reverse letterbox offsets/scale; filter by confidence and IoU NMS; map class IDs to basketball labels.
- **Drawing:** bounding boxes and labels; optional per-class colors.
- **Runtime metrics:** warmup iterations, FPS, mean/median ms, detections/image.
- **CLI:**
  - `--model <onnx>` (required).
  - Exactly one input: `--frames <dir>` or `--video <file>` (video not yet implemented).
  - Common flags: `--size 640 640`, `--conf`, `--iou`, `--cache`, `--warmup`, `--max_frames`, `--show`.

3.16 One-Command Demo Runner (ZCU104)

- **Script:** `run_demo.sh <model_name> <onnx_model_path> <frames_or_video_path>`.
- Creates per-model VAIP cache at `/home/xilinx/.vaip_cache/<model_name>`.
- Invokes `deploy_onnx_vaip.py` for image directories (video path rejected with guidance).
- Provides optional (commented) VAIP/VART environment variables.
- Prints a banner (model, ONNX path, input, board) and persists cache across runs.

4 Results

We obtained minimal yet sufficient artifacts in this phase. A single trained YOLOv11 variant reached approximately 90% validation accuracy and was successfully quantized to INT8 in QDQ format. Detailed size breakdowns and DPU throughput are deferred to later iterations.

5 Discussion

Residual non–DPU operators typically arise around shape–manipulation paths (e.g., `Resize+Concat` and mixed broadcast forms) or within composite blocks that elude PyTorch–level substitutions. ONNX Runtime's quantizers are sensitive to the model IR/opset: they prefer IR = 10, so our pipeline temporarily down–levels the model for quantization, then restores the original IR while enforcing a default–domain opset ≤ 13 for Vitis robustness. The postprocessing head can emit raw tensors in either $[B, (5{+}C), A]$ or $[B, A, (5{+}C)]$ layout; the inference client detects the layout and transposes as needed, and practitioners must keep the class list length consistent with C when retraining on different label sets. After compilation, small CPU subgraphs ("islands," e.g., bookkeeping nodes) may persist; our reports and optimizer reduce these without aggressive surgery. Finally, VAIP's on–disk cache materially improves iteration speed, and the demo runner provides a standardized one–line launch with guardrails such as requiring an image directory.

6 Conclusion

We provide a complete, reproducible pipeline that: (1) trains YOLOv11 variants, (2) exports patches ONNX graphs, (3) quantizes to INT8 QDQ tailored to DPUCZDX8G, (4) produces compile scripts deployment metadata, and (5) runs with an ONNX VAIP client and a demo launcher on ZCU104. The artifacts enable immediate Vitis AI compilation and practical, cached VAIP inference while illuminating exact operator gaps for the next optimization round.

7 Future Work

Near-term (toolchain)

- Compile INT8 ONNX to `.xmodel` on ZCU104 via the generated scripts; analyze partition logs and iterate to eliminate CPU "islands."
- Add optional per-channel quantization if/when stable for our models; explore QAT if accuracy drops.
- Extend the VAIP client to video streams; integrate a fast on-device NMS path when applicable.

Deferred system integration

- HDMI input capture and preprocessing on ZCU104.
- DeepSORT multi-object tracking for persistent IDs.
- Scoreboard OCR and event alignment to auto-validate shots.

References

[1] Ultralytics. *YOLOv11: Documentation and Code*. 2024–2025.

[2] AMD/Xilinx. *Vitis AI: Quantization & Compiler User Guides*. 2024–2025.

[3] Roboflow. *"basketball-hoop-ball-and-player-exknu" Dataset*. 2025.

[4] ONNX Runtime. *Static QDQ Quantization & Vitis-AI Execution Provider Documentation*. 2024–2025.