fire

MDPI

Article

YOLO-FSD: A Deployment-Validation-Oriented Lightweight Fire Smoke Detection Network for Resource-Constrained ZYNQ7020 FPGA Edge Platforms

Chaoyun Mai, Panrong Chen, Haipeng He, Hao Xie *, Chongyi Huang, Tianlei Wang, Zhiyuan Su and Hongye Li

College of Electronics and Information Engineering, Wuyi University, Jiangmen 529020, China; j002365@wyu.edu.cn (C.M.); 2112424012@wyu.edu.cn (P.C.); 2112424028@wyu.edu.cn (H.H.); 2112524007@wyu.edu.cn (C.H.); j001803@wyu.edu.cn (T.W.); 2112424004@wyu.edu.cn (Z.S.); 2112424086@wyu.edu.cn (H.L.)

* Correspondence: j002906@wyu.edu.cn

Abstract

Early-stage flame and smoke in forest fire scenes are often small, weakly textured, and easily confused with complex backgrounds, while many accurate YOLO-based detectors are difficult to deploy on resource-constrained FPGA devices. This study proposes YOLO-FSD, a lightweight detector derived from YOLOv4-tiny for FPGA-oriented deployment. It integrates inverted residual and depthwise separable structures, a lightweight semantic enhancement (LSE) block at the deep feat2 feature, a lightweight P_4 detection head, and a shallow detail compensation branch. On the combined test set of the D-Fire and New Fire and Smoke datasets, YOLO-FSD achieves a mean average precision at an intersection-over-union threshold of 0.5 (mAP50) of 69.36%, with 3.951 M parameters and 1.520 G multiply-accumulate operations (MACs). Compared with YOLOv4-tiny, mAP50 increases by 2.76 percentage points, while parameters and MACs decrease by 32.76% and 55.53%, respectively. For deployment validation, batch normalization (BN) fusion and 16-bit integer (INT16) parameter conversion are followed by fixed-point forward inference on a Xilinx Zynq-7020 FPGA, with a software-side parameter-quantization sensitivity analysis used as an intermediate check. FPGA raw output evaluation achieves 69.14% mAP50, only 0.22 percentage points below the PyTorch 32-bit floating-point (FP32) model, demonstrating the feasibility of FPGA-side forward inference.

Keywords: fire detection; smoke detection; lightweight YOLO; INT16 quantization; ZYNQ7020 FPGA

Check for updates

Academic Editor: Grant Williamson

Received: 13 August 2026
Revised: 31 August 2026
Accepted: 2 September 2026
Published: 13 September 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Forest fire detection is an important research topic for safeguarding forest resources and ecological environments. Image-based flame and smoke recognition extracts edge, shape, color, texture, and spatial-location cues from fire-related targets, thereby formulating early fire warning as a computer-vision object detection problem. Research on forest fire detection has improved automatic recognition, localization, and early-warning capability for flame and smoke targets, supporting applications such as forest fire prevention [1], unmanned aerial vehicle inspection [2], and smart agriculture [3]. These advances are important for early warning, loss mitigation, and public safety.

Forest fire images are usually acquired in open environments and are inevitably affected by target scale, illumination variation, cloud occlusion, vegetation backgrounds,

and other factors. In the early stage of a fire, flame regions are often small, while smoke targets typically have diffuse boundaries and weak textures. Such targets can be visually confused with clouds, fog, shadows, and complex backgrounds, which may cause the detector to learn insufficient or noisy discriminative features. As a result, missed detections and false alarms are likely to occur for small flames and thin smoke.

Traditional forest fire monitoring mainly relies on manual patrols and sensor-based detection [4], but these approaches are limited by coverage, response latency, and environmental interference in open forest regions. With the development of deep learning, convolutional neural networks and object detection models have increasingly been applied to flame and smoke detection tasks [5].

Among these methods, the YOLO family has become an important framework for flame and smoke detection because its one-stage end-to-end design provides a favorable balance between detection accuracy and inference efficiency [6]. To address scale variation, strong background interference, and small early-stage fire targets in forest fire images, existing studies often enhance feature fusion [7,8] or introduce attention mechanisms [9,10], achieving promising results in fire detection tasks [11–15]. However, these methods usually increase structural complexity, parameter count, and computational cost, which is unfavorable for deployment on resource-constrained FPGA platforms such as the ZYNQ7020. YOLOv4 provides a representative one-stage detection framework with a favorable speed-accuracy trade-off [16], while its scaled-down variant, YOLOv4-tiny, adopts a relatively simple two-scale detection topology with lower model complexity [17]. Therefore, YOLOv4-tiny provides a structurally simple baseline for investigating deployment-oriented lightweight redesign. MobileNetV2 reduces convolutional complexity through inverted residual blocks and depthwise separable convolutions [18], providing a useful basis for lightweight detector design on resource-constrained platforms.

To reduce model complexity, lightweight object detection research commonly focuses on lightweight backbones [19,20], efficient convolutional structures [21], and model compression [22,23]. Architectures such as MobileNet, ShuffleNet, and GhostNet reduce parameters and computation through depthwise separable convolution, group convolution, channel shuffle, or feature redundancy exploitation [24,25], providing foundations for edge-side detector design.

However, in forest fire scenes with small targets, weak textures, and complex backgrounds, straightforward lightweighting may weaken semantic representation or discard fine details. Therefore, some studies combine lightweight backbones or efficient convolutional structures with upsampling reorganization, attention mechanisms, or multi-scale feature enhancement to compensate for the loss of representation capacity [26,27]. Similar lightweight detection strategies have also been adopted in other natural-scene object detection tasks [28]. These studies indicate that lightweight fire detection models should not only reduce computation, but also compensate for flame morphology, weak smoke texture, and shallow boundary information.

Lightweight model design is a key prerequisite for FPGA deployment, but whether a model can run on the device is also determined by on-chip computing resources, memory resources, data bandwidth, operator types, and software-hardware co-scheduling. FPGA-oriented neural network deployment requires comprehensive optimization in model compression [29], low-bit fixed-point representation [30], the use of regular convolutional operators such as depthwise separable convolution [31], and software-hardware co-scheduling [32].

In practical deployment, quantization, fixed-point conversion, regular operator mapping, and software-hardware co-scheduling are key steps in FPGA neural network implementation. Quantization and fixed-point conversion map FP32 parameters from training

to low-bit fixed-point data, thereby reducing parameter storage, data movement, and multiply-accumulate resource consumption. Previous studies have reduced hardware complexity and improved FPGA-side inference efficiency for object detection by using quantization-aware neural network layers or low-bit quantization [33,34]. Regular operators, such as standard convolution, 1×1 pointwise convolution, 3×3 depthwise convolution, pooling, upsampling, and element-wise addition, have stable computation patterns, regular data access, and high parallelism, making them suitable for high-level synthesis (HLS) or custom intellectual-property (IP) core implementation [35]. In contrast, complex attention mechanisms, dynamic branches, and irregular memory access may improve accuracy but increase computational cost, control logic, and data scheduling overhead, making efficient deployment more difficult on resource-constrained FPGA platforms [36]. In addition, deploying object detection models on ZYNQ platforms requires consideration of processing system (PS)/programmable logic (PL) collaboration, memory access, data transfer, and hardware resource constraints so that accuracy, inference speed, and power consumption can be evaluated together [37].

The above studies show that existing fire detection models often improve accuracy through complex backbones, multi-scale feature fusion, or attention mechanisms. Such designs can increase structural and dataflow complexity and may complicate direct mapping to low-resource FPGA platforms such as the ZYNQ7020, depending on the accelerator architecture and deployment workflow. Conversely, using a lightweight backbone alone can reduce model complexity but may weaken deep semantic representation and shallow-detail retention, leaving small flames, thin smoke, and weak-boundary targets vulnerable to missed detections, false alarms, and inaccurate localization. Therefore, this study follows the main line of lightweight structural design, key feature compensation, fixed-point quantization, and FPGA deployment validation, and develops YOLO-FSD, where FSD denotes Fire Smoke Deployable design, as a lightweight flame and smoke detection method for resource-constrained FPGA platforms.

Within this deployment-oriented context, YOLOv4-tiny was selected as a relatively simple two-scale architectural starting point rather than as a state-of-the-art accuracy baseline. This selection does not imply that every operator in the original YOLOv4-tiny model is inherently FPGA-friendly. Instead, its relatively simple and static two-scale topology provides a suitable basis for structural reorganization. In YOLO-FSD, the original backbone is substantially redesigned using inverted residual, depthwise, and pointwise convolutions, resulting in a more regular fixed-shape dataflow for INT16 custom-HLS implementation.

Parameter count and MACs alone do not fully characterize the implementation cost of a detector on a resource-constrained FPGA. Intermediate-feature lifetime, high-resolution feature-map buffering, branch synchronization, tensor concatenation, nonlinear-function implementation, and external-memory traffic are also important hardware considerations. Although recent nano-scale models have competitive parameter counts and MACs, their operator and dataflow characteristics differ from those of the two-scale structure adopted in this work. YOLOv8n uses C2f blocks that split the input and retain multiple intermediate bottleneck outputs for subsequent concatenation, together with spatial pyramid pooling-fast (SPPF) and sigmoid linear unit (SiLU) activation [38]. More recent generations include YOLOv9 [39], which introduces programmable gradient information and GELU-based architectural design, and YOLOv10 [40], which develops an end-to-end detection framework without non-maximum suppression (NMS). YOLOv11n replaces C2f with C3k2 blocks and introduces C2PSA after SPPF, adding CSP-style feature branching and attention-related transformations [41]. YOLOv12n further employs attention-oriented A2C2f blocks together with C3k2, increasing the requirements for intermediate-feature buffering, tensor rearrangement, and fixed-point implementation of area-attention operations [42]. In addition, the

standard detection configurations of these models use $P_3/P_4/P_5$ three-scale prediction, whereas YOLO-FSD retains a P_4/P_5 two-scale prediction structure. The additional high-resolution P_3 pathway increases feature-map storage and data movement even when the parameter count is relatively small. These recent models are not inherently unsuitable for FPGA deployment; however, their retained intermediate features, split/concat branching, nonlinear activations, attention-related transformations, and three-scale prediction dataflow introduce additional buffering, memory traffic, and scheduling requirements on a resource-constrained platform such as the ZYNQ7020.

This consideration is also supported by recent FPGA-oriented studies. For example, Li et al. implemented a YOLOv8-derived detector on the Zynq-7020 and reported that the standard C2f multi-branch split-and-merge topology requires intermediate feature buffering for branch synchronization. Their hardware-oriented redesign removes parallel branches, channel-splitting operations, and feature-concatenation paths to reduce the associated control and data-path overhead [43]. This evidence indicates that branch structure, intermediate-feature lifetime, and data movement can materially affect FPGA implementation cost beyond parameter count and MACs. Nevertheless, the present study does not provide a same-board hardware implementation of YOLOv8n, YOLOv11n, or YOLOv12n; therefore, these considerations are used only as a deployment-oriented architectural rationale and should not be interpreted as experimentally demonstrated hardware superiority of YOLO-FSD over these models.

Experimental results show that YOLO-FSD improves the YOLOv4-tiny baseline while reducing model size and computation. The FPGA-side raw-output evaluation shows a 0.22 percentage-point mAP50 decrease compared with the PyTorch FP32 model.

The main contributions of this study are as follows:

1. A lightweightforest fire object detection model, YOLO-FSD, is proposed for resource-constrained FPGA deployment validation. Based on YOLOv4-tiny, the model replaces part of the original structure with inverted residual and depthwise separable operators to reduce parameters and computation while retaining an operator pattern that is suitable for fixed-point conversion and custom HLS/IP mapping.
2. A deployment-oriented LSE block based on an inverted residual-type operator pattern is placed once at the deep feat2 feature and integrated with the P_5 and P_4 detection pathways. In this work, the LSE block is characterized by its fixed lightweight configuration, specific placement at feat2, and integration with the two-scale detection architecture. In addition, a lightweight P_4 detection head and a shallow detail-compensation branch are used to reduce high-resolution computation and mitigate the potential loss of local-detail sensitivity introduced by P_4-head lightweighting.
3. BN fusion, software-side INT16 parameter-quantization sensitivity analysis, and ZYNQ7020 FPGA-side forward-inference validation are conducted. Comparisons among the original PyTorch2.7.1FP32 model, the BN-fused FP32 model, software INT16 parameter-quantization evaluation, and FPGA raw-output evaluation are used to evaluate quantization sensitivity, software-hardware output comparison, resource utilization, forward-only latency, power consumption, and the current deployment limitations of the model.

2. Materials and Methods

2.1. Dataset Construction and Partitioning

The dataset used in this study was constructed from two publicly available sources: the D-Fire dataset [44] and the New Fire and Smoke dataset [45]. D-Fire is distributed under the CC0 1.0 Universal license, while the New Fire and Smoke data used in this study correspond to the v2 public release on Roboflow Universe (20 August 2024) under the

CC BY 4.0 license. Both datasets are publicly accessible from the D-Fire GitHubrepository (https://github.com/gaia-solutions-on-demand/DFireDataset, accessed on 13 July 2026) and the FireSmokeNEWdataset project on Roboflow Universe (https://universe.roboflow.com/catargiuconstantin2/firesmokenewdataset, accessed on 13 July 2026), respectively (accessed on 13 August 2026). The final training, validation, and test sets contain 17,855, 2795, and 4810 images, respectively, and all subsets were fixed before model training. The merged corpus was partitioned at the image level using fixed training, validation, and test manifests rather than by location, sequence/event, or time grouping. A cross-split similarity audit found no test image that was identical to a training or validation image at either the file-byte level or the decoded-pixel level. However, 1467 of the 4810 test images (30.50%) were associated with near-duplicate or highly similar scenes present in the training and/or validation subsets. Cross-split similarity was audited using complementary perceptual and content-based measures. Exact duplicates were first checked using SHA-256 hashes of both the original image files and the decoded RGB pixel arrays. For perceptual matching, each image was represented using a DCT-based perceptual hash (pHash), a difference hash (dHash), a normalized two-dimensional HSV histogram, and the structural similarity index (SSIM). For each test image, candidate matches were searched separately against the training and validation subsets. The five closest candidates were selected using a weighted hash distance score in which the pHash distance was assigned twice the weight of the dHash distance. Candidates with a pHash distance no greater than 12, a dHash distance no greater than 18, and an HSV-histogram similarity of at least 0.78 were retained for subsequent SSIM-based classification. Using predefined thresholds, a pair was classified as a near duplicate when the pHash distance was no greater than 4, the dHash distance was no greater than 6, the histogram similarity was at least 0.93, and the SSIM value was at least 0.88, or when the SSIM value alone was at least 0.97. A pair was classified as a highly similar scene when the pHash distance was no greater than 8, the dHash distance was no greater than 12, the histogram similarity was at least 0.88, and the SSIM value was at least 0.70. Representative high-similarity pairs were additionally assembled into contact sheets and visually inspected as a qualitative sanity check. This inspection was not used to alter the predefined thresholds, automated labels, or reported counts. Therefore, cross-split scene similarity remains a limitation of the current image-level partitioning protocol and may introduce some optimism into the absolute performance estimates. A fully sequence-, event-, camera-, or scene-grouped repartition would provide a stronger assessment of scene-independent generalization; however, complete and consistent grouping identifiers are not available across the entire merged corpus. Constructing such a partition retrospectively would therefore require additional heuristic grouping assumptions that cannot be verified uniformly for all images. Accordingly, the predefined image-level split is retained, and the quantitative cross-split audit and overlap-filtered sensitivity analysis are reported to make this limitation explicit.

To examine the sensitivity of the model comparison to this issue, a conservative overlap-filtered test subset was additionally constructed using a deliberately broader exclusion rule than that used to report the 1467 strict near-duplicate or highly similar cases. In addition to these strict cases, the exclusion set included lower-threshold similar-scene candidates, defined by a pHash distance no greater than 10, a dHash distance no greater than 14, a histogram similarity of at least 0.84, and an SSIM value of at least 0.60. New Fire and Smoke test frames whose original source-video identifiers were also represented in the training and/or validation subsets were additionally excluded. The final 1919 excluded images therefore represent the union of these broader exclusion criteria rather than 1919 images all classified as near duplicates or highly similar scenes, leaving

2891 images for the overlap-filtered analysis. This subset was used only for supplementary sensitivity analysis and did not replace the predefined 4810-image main test set.

The merged dataset contains Fire and Smoke targets from heterogeneous outdoor scenes, including forests, mountainous areas, grasslands, and aerial views, under daytime, dusk, nighttime, and low-illumination conditions. Target scales range from distant small flames and thin smoke plumes to large-area fire and smoke. The corresponding image and ground-truth statistics are summarized in Table 1.

Table 1. Dataset split and object-count statistics.

Split	Images	Fire GT	Smoke GT	Total GT
Train	17,855	18,514	18,976	37,490
Validation	2795	3153	2907	6060
Test	4810	3159	2718	5877
Total	25,460	24,826	24,601	49,427

Note: GT denotes the number of ground-truth bounding boxes, Fire GT and Smoke GT denote the number of ground-truth boxes for flame and smoke targets, respectively, and Total GT is the sum of the two classes.

The two source datasets have different image-resolution and preprocessing characteristics. D-Fire images do not have a uniform source resolution, whereas the New Fire and Smoke v2 release used here was exported after automatic orientation correction and resizing to 640 × 640 pixels. All images were subsequently converted to RGB and resized to 416 × 416 pixels for model input, as described in Section 2.2.

Complete metadata on camera-to-target distance, camera geometry, geographical location, and dominant vegetation type are not consistently provided by the source datasets. Therefore, these quantities are not inferred retrospectively. Although the merged corpus contains several conditions relevant to early visual fire smoke detection, it should not be regarded as a geographically balanced or fully metadata-controlled benchmark representative of all early-warning surveillance scenarios.

2.2. Experimental Setup and Evaluation Protocol

The software experiments were conducted on Windows 10 using Python 3.10.19, PyTorch 2.7.1, CUDA 11.8, and an NVIDIA GeForce RTX 3080 Ti GPU. Images were converted to RGB, resized to 416 × 416, and normalized to [0, 1]. Applicable backbone parameters were initialized from a pretrained MobileNetV2 checkpoint [46], while unmatched parameters were initialized randomly.

All models were trained for 250 epochs with a batch size of 16 using stochastic gradient descent with Nesterov momentum. The momentum and weight-decay coefficients were 0.937 and 0.0005, respectively. The base learning rate was 0.01 with a final factor of 0.01; after adaptive batch-size scaling, the effective cosine-annealing schedule ranged from 0.0025 to 0.000025. Mosaic [16] and MixUp [47] were each applied with a probability of 0.5 during the first 70% of training and were disabled during validation and testing. The same training settings were used for all model configurations.

The training subset was used for parameter optimization, while the validation subset was used for checkpoint and operating-threshold selection. For each configuration, the checkpoint with the minimum validation loss was selected. A global operating confidence threshold of 0.35 was determined by maximizing the macro-averaged $F1$ score of the IRDS-YOLOv4-tiny baseline on the validation subset and was then fixed for all models. The anchor boxes were (8, 11), (17, 22), (33, 43), (69, 76), (150, 162), and (329, 263), with the masks [[3, 4, 5], [0, 1, 2]] assigned to the P_5 and P_4 heads, respectively. This configuration was inherited from YOLOv4-tiny. The network architecture, ablation configurations, and

inference parameters were also fixed before testing, and the test subset was used only for final software- and FPGA-side evaluation.

For AP50 and mAP50 calculation, predictions were retained at a low confidence threshold of 0.001 and processed by class-wise NMS with an intersection-over-union (IoU) threshold of 0.5. Precision, Recall, and $F1$ scores were reported at the fixed operating threshold of 0.35.

FPGA-side forward-inference evaluation covered the IRDS-YOLOv4-tiny baseline, all predefined ablation configurations, and YOLO-FSD. For each configuration, all 4810 test images were executed on the ZYNQ7020 using the same inputs, anchors, thresholds, class order, and matching protocol as the software-side evaluation. The FPGA generated the P_4 and P_5 raw outputs, while decoding, confidence filtering, NMS, and metric calculation were performed on the PC.

To characterize training variability, the four principal IRDS-based ablation configurations were independently trained using three predefined random seeds: 11, 12, and 1024. For each run, the corresponding seed was applied to the Python, NumPy, PyTorch CPU, and CUDA random-number generators. Training-set shuffling, DataLoader-worker initialization, and stochastic data augmentation were also governed by the run-specific seed. Model construction was performed after seed initialization, so the random initialization of newly introduced or otherwise unmatched parameters was controlled by the corresponding PyTorch random state, while compatible backbone parameters were loaded from the same pretrained MobileNetV2 checkpoint. The dataset partition, input resolution, training hyperparameters, augmentation settings, checkpoint-selection criterion, confidence threshold, anchor configuration, and evaluation protocol were kept unchanged across runs.

For each run, the checkpoint with the minimum validation loss was selected for test-set evaluation. Unless otherwise specified, repeated-training results are reported as mean $\pm$ standard deviation over the three independent runs. Because FPGA deployment requires a concrete trained checkpoint rather than an averaged performance value, a representative checkpoint with principal detection performance close to the mean of the repeated runs was retained for the subsequent deployment-validation workflow. The same representative checkpoint was used consistently throughout the software-to-hardware deployment-validation experiments.

2.3. Evaluation Metrics

This study evaluates detection performance using Precision, Recall, $F1$-score, IoU, AP50, and mAP50. TP, FP, and FN denote true positives, false positives, and false negatives, respectively. B_{pred} and B_{gt} denote the predicted and ground-truth bounding boxes. The main detection metrics are defined as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

$$\text{IoU} = \frac{B_{pred} \cap B_{gt}}{B_{pred} \cup B_{gt}} \tag{4}$$

$$\text{AP}_{50} = \int_0^1 P(R)dR, \quad \text{IoU} = 0.5 \tag{5}$$

$$\text{mAP}_{50} = \frac{1}{C}\sum_{c=1}^{C} \text{AP}_{50}^{(c)} \tag{6}$$

where $P(R)$ denotes Precision as a function of Recall, C is the number of target classes, and $AP_{50}^{(c)}$ denotes the AP50 of class c. In this study, $C = 2$ corresponds to Fire and Smoke.

For overall two-class reporting, Precision, Recall, and $F1$ are macro-averaged independently over the Fire and Smoke classes. Therefore, the reported macro-averaged $F1$ is not necessarily equal to the harmonic mean of the macro-averaged Precision and Recall.

For FPGA deployment evaluation, latency, FPGA forward-inference FPS, energy per frame, FPS/W, and GOPS are used. Let T_{lat} denote the single-frame inference latency in ms, P_{power} denote power consumption in W, Ops denote the number of operations required for one-frame inference, and M_{MAC} denote the total number of multiply-accumulate operations per frame in GMACs.

$$\text{FPS} = \frac{1000}{T_{lat}} \tag{7}$$

$$E_{frame} = P_{power} \times \frac{T_{lat}}{1000} \tag{8}$$

$$\text{FPS/W} = \frac{\text{FPS}}{P_{power}} \tag{9}$$

$$\text{GOPS} = \frac{\text{Ops}}{T_{lat} \times 10^6} \tag{10}$$

where one multiply-accumulate operation is counted as two operations when calcu-lating the achieved GOPS. This set of metrics jointly reflects detection accuracy, fixed-point deployment consistency, and on-board inference efficiency.

2.4. Overall Framework of YOLO-FSD

To improve the deployment adaptability of flame and smoke detection models on resource-constrained FPGA platforms, this study first constructs an inverted residual and depthwise separable (IRDS) YOLOv4-tiny baseline by replacing the original YOLOv4-tiny backbone with IRDS structures. On this basis, YOLO-FSD is developed as a lightweight fire smoke detection network for low-resource deployment validation. The model preserves the two-scale detection framework of YOLOv4-tiny and integrates an IRDS-based lightweight backbone, LSE-based deep semantic enhancement, a lightweight P_4 detection head, and shallow detail compensation to balance detection accuracy, model complexity, and hardware implementation cost. The overall architecture of YOLO-FSD is shown in Figure 1.

As shown in Figure 1, the input image is first processed by the IRDS backbone to extract shallow high-resolution features and deep semantic features. The deep feature is strengthened by the LSE module and used for the low-resolution detection branch, while its upsampled semantic information is fused with shallow detail cues for the P_4 branch. In the P_4 detection head, standard convolution is replaced with a P_4-DSConv head based on depthwise and pointwise convolutions, and the detail compensation branch supplies complementary edge, texture, and local contour information. After training, BN fusion, INT16 fixed-point parameter export, and on-board inference on a ZYNQ7020 FPGA are performed to evaluate fixed-point deployment feasibility and output comparison between software and hardware.

2.5. IRDS-YOLOv4-Tiny Baseline Model

This study adopts YOLOv4-tiny as the base framework because of its simple two-scale detection pipeline, regular operators, and relatively low mapping difficulty on resource-constrained FPGA platforms. The original backbone is replaced with an IRDS backbone, forming the IRDS-YOLOv4-tiny baseline. The IRDS backbone reduces parameters and multiply-accumulate operations through pointwise expansion, depthwise spatial convolu-

tion, and pointwise projection while maintaining a regular operator pattern suitable for fixed-point conversion and FPGA mapping. Two feature layers are selected for detection: the shallow feature feat1 retains higher-resolution edge, texture, and local color information, whereas the deep feature feat2 provides stronger semantic representation for flame and smoke discrimination.

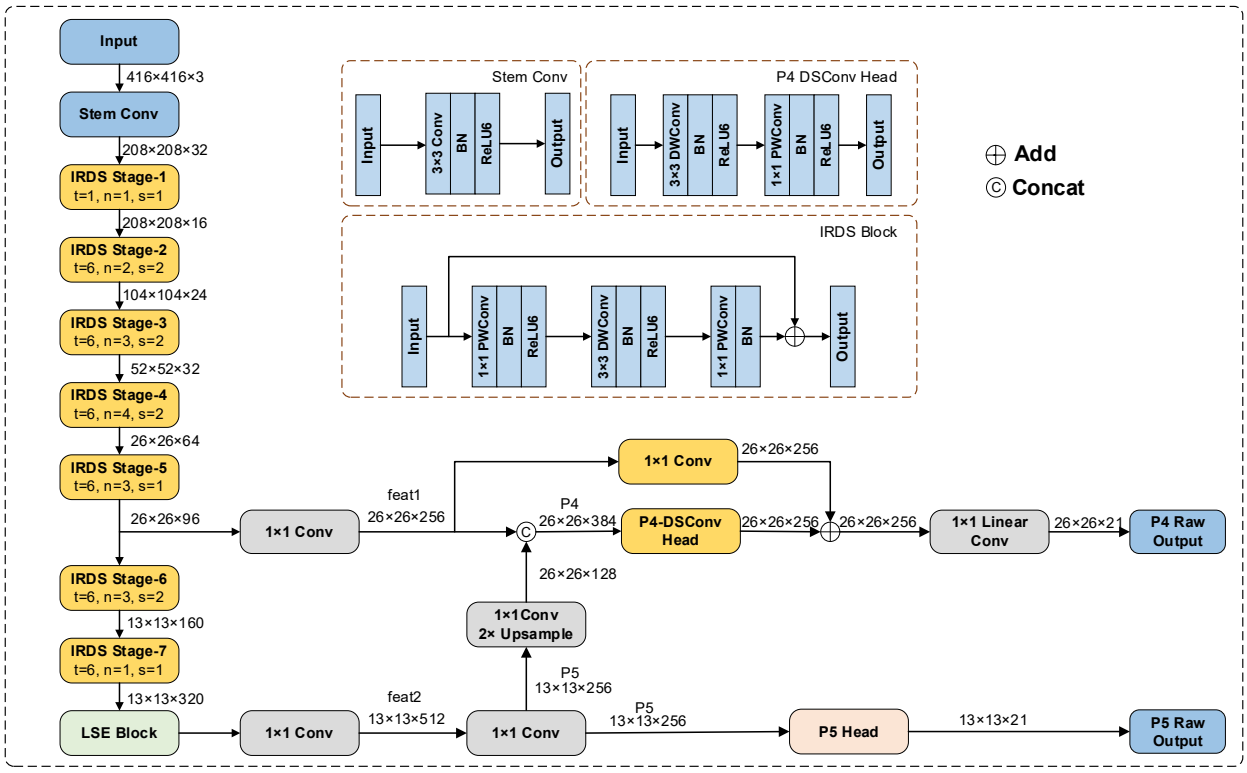


Figure 1. Overallarchitecture of YOLO-FSD.

2.6. Lightweight Semantic Enhancement (LSE) Module

The IRDS backbone reduces the computational burden of the detector, but lightweight feature extraction may also reduce the representation capacity of the deep feature. To address this issue, an LSE block is inserted once at the deep feat2 feature. The LSE block builds on the MobileNetV2-type inverted residual design used in the lightweight backbone. Accordingly, the contribution claimed for LSE in this work concerns its deployment-oriented configuration, placement, and integration rather than the invention of the underlying inverted residual operator. It is configured as a stride-1 feature-transformation block with a fixed expansion ratio of 2, an identity residual connection, and a post-addition ReLU6. Its technical distinction lies in this specific configuration, its functional role and placement at feat2, and its integration with the P_5 and P_4 detection pathways.

The ordinary IRDS blocks are repeatedly used for backbone feature extraction and downsampling. Depending on the stage, they may use a stride of 1 or 2, change the spatial resolution and channel dimension, and apply a residual connection only when the stride is 1 and the input and output dimensions are identical. Their residual output is not followed by an additional activation. In contrast, LSE is applied only at the $13 \times 13 \times 320$ feat2 feature, performs a 320–640–320 channel transformation with a stride of 1, preserves the input and output dimensions, always uses an identity residual connection, and applies ReLU6 after residual addition. These configuration choices retain a regular operator sequence suitable for INT16 conversion and FPGA implementation.

Let X denote the input deep feature of the LSE module and $\hat{X}$ denote the enhanced output feature. The activation function used in the module is ReLU6:

$$\sigma_6(z) = \min(\max(z, 0), 6) \tag{11}$$

where z denotes an arbitrary scalar element of the input feature tensor. The ReLU6 function is applied element-wise to the feature tensor and limits the activation value to the interval [0, 6]

$$U = \sigma_6\left(\mathrm{BN}_e\left(W_e^{1\times1} * X\right)\right) \tag{12}$$

Here, U denotes the expanded semantic feature, $W_e^{1\times1}$ is the expansion pointwise convolution kernel, $*$ denotes the convolution operation, and $\mathrm{BN}_e(\cdot)$ denotes batch normalization after expansion. The subscript e refers to the expansion stage. Then, a 3×3 depthwise convolution is used to capture local semantic variation. For the r-th expanded semantic component at spatial position $q = (i, j)$, the operation is expressed as:

$$V_r(q) = \sigma_6\left(\mathrm{BN}_d\left(\sum_{a\in\mathcal{S}} K_{r,a}^d U_r(q+a)\right)\right), \quad \mathcal{S} = \{(-1,-1),\ldots,(1,1)\} \tag{13}$$

In Equation (13), $V_r(q)$ denotes the depthwise-transformed feature at spatial position q in the r-th expanded channel, and $U_r(q + a)$ denotes the corresponding input value of the expanded feature. Here, r indexes the expanded feature channels; $q = (i, j)$ denotes a two-dimensional spatial position, where i and j are the row and column indices, respectively; and $a = (m, n)$ denotes a local spatial offset. The set $\mathcal{S} = \{(-1, -1), \ldots, (1, 1)$ represents the offsets within the 3×3 convolution neighborhood. $K_{r,a}^d$ denotes the depthwise convolution kernel weight associated with channel r and offset a and $\mathrm{BN}_d(\cdot)$ denotes batch normalization after depthwise convolution. The subscript d refers to the depthwise-convolution stage.

Finally, the locally transformed feature is projected back to the original channel dimension by a 1×1 pointwise convolution and fused with the identity branch:

$$R(X) = \mathrm{BN}_p\left(W_p^{1\times1} * V\right), \quad \hat{X} = \sigma_6(X + R(X)) \tag{14}$$

In Equation (14), V denotes the depthwise-transformed feature tensor, $W_p^{1\times1}$ is the projection pointwise convolution kernel, and $\mathrm{BN}_p(\cdot)$ denotes batch normalization after projection. The term $R(X)$ denotes the residual correction produced by the projection branch, and $\hat{X}$ is the final output of the LSE block. Together, the pointwise expansion, channel-wise local spatial filtering, linear projection, and residual connection constitute a lightweight local feature-transformation block. The 3×3 depthwise convolution processes each expanded feature channel within a local spatial neighborhood, while the regular operator sequence facilitates fixed-point conversion and FPGA implementation. The structure of the LSE module is shown in Figure 2.

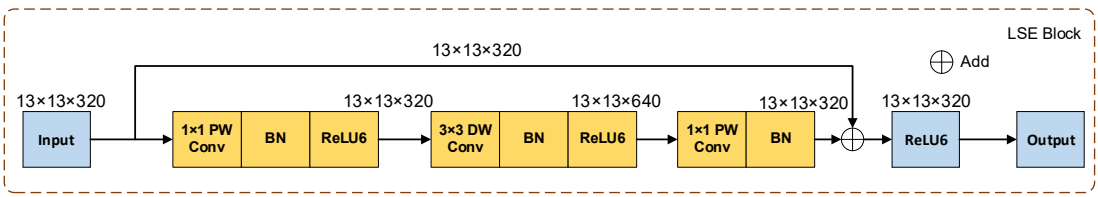


Figure 2. Structureof the LSE module.

2.7. Lightweight P_4 *Detection Head and Detail Compensation Branch*

In the two-scale detection structure, the P_4 branch preserves fine spatial cues that are important for small flames, thin smoke, and weak fire smoke boundaries. However, directly using standard convolution in this branch introduces considerable computation and memory-access overhead. To reduce the cost of high-resolution feature processing, this study replaces the standard convolution in the P_4 detection head with a depthwise separable convolution structure, referred to as the P_4-DSConv head. Meanwhile, a shallow detail-compensation branch is introduced to supplement local edges, textures, and contour cues that may be weakened by lightweight convolution.

Let F_1 denote the shallow feature after channel adaptation, P_5 denote the deep semantic feature from the upper detection branch, and $\mathcal{A}(\cdot)$ denote the upsampling path including feature projection and spatial upsampling. The fused P_4 feature is formulated as:

$$P_4 = \mathrm{Concat}_c(\mathcal{A}(P_5), F_1) \tag{15}$$

where the concatenation operator denotes channel-wise concatenation. The fused feature contains both deep semantic information from the upper branch and shallow spatial information from the shallow feature.

For the lightweight P_4 detection head, the spatial filtering stage is first performed by depthwise convolution:

$$G_s(p) = \sigma_6\left(\mathrm{BN}_{dw}\left(\sum_{\Delta\in\mathcal{N}} \theta^{dw}_{s,\Delta} P_{4,s}(p+\Delta)\right)\right), \quad \mathcal{N} = \{(-1,-1),\ldots,(1,1)\} \tag{16}$$

In Equation (16), $p = (i, j)$ denotes a spatial position, s denotes the s-th feature component, $\Delta = (m, n)$ denotes the local spatial offset in the 3×3 neighborhood, $\theta^{dw}_{s,\Delta}$ is the depthwise convolution kernel weight, G_s is the spatially filtered feature, $\mathrm{BN}_{dw}(\cdot)$ is batch normalization after depthwise convolution, and $\sigma_6(\cdot)$ denotes ReLU6.

After spatial filtering, a 1×1 pointwise convolution is used for cross-component information mixing:

$$Z_{4,t}(p) = \sigma_6\left(\mathrm{BN}_{pw}\left(\sum_s \theta^{pw}_{t,s} G_s(p)\right)\right) \tag{17}$$

Here, $Z_{4,t}$ denotes the t-th output component of the lightweight P_4 head, $\theta^{pw}_{t,s}$ is the pointwise convolution weight connecting the s-th input component to the t-th output component, and $\mathrm{BN}_{pw}(\cdot)$ denotes batch normalization after pointwise convolution.

Although the depthwise-pointwise structure reduces computation, it decouples spatial filtering and component mixing, which may weaken local detail representation. Therefore, a detail compensation branch is constructed from the shallow feature F_1:

$$D_4 = \psi(F_1) = \sigma_6\left(\mathrm{BN}_{dc}\left(\theta^{1\times1}_{dc} * F_1\right)\right) \tag{18}$$

The compensated P_4 feature is obtained by element-wise addition:

$$\widetilde{Z}_4 = Z_4 + D_4 \tag{19}$$

Here, Z_4 is the output of the lightweight P_4 head before compensation, D_4 provides shallow boundary and texture cues, and $\widetilde{Z}_4$ is the detail-enhanced P_4 feature used for final prediction.

The final P_4 prediction is produced by a 1×1 output convolution:

$$Y_4 = \theta_{out}^{1\times1} * \widetilde{Z}_4 + b_{out} \tag{20}$$

where Y_4 denotes the P_4 detection output, and $\theta_{out}^{1\times1}$ and b_{out} are the weight and bias of the output prediction convolution.

The computational advantage of the lightweight head can be expressed by comparing standard convolution and depthwise-pointwise convolution. For kernel size k, input component number N_{in}, output component number N_{out}, and spatial position set Ω, the costs are:

$$\text{MAC}_{std} = \left|\Omega\right| k^2 N_{in} N_{out}, \quad \text{MAC}_{dw-pw} = \left|\Omega\right| \left(k^2 N_{in} + N_{in} N_{out}\right). \tag{21}$$

Here, Ω denotes the number of spatial positions. Equation (21) shows that the depthwise-pointwise head separates spatial filtering from component mixing, thereby re-ducing computation in the P_4 branch. Since the detail compensation branch only uses 1×1 convolution and element-wise addition, it supplements shallow cues without in-troducing global attention, dynamic routing, or irregular data access. Therefore, the lightweight P_4 head is designed to reduce the computational cost of high-resolution feature processing, while the detail-compensation branch is designed to mitigate po-tential loss of local boundary and texture information. Both components retain an FPGA-friendly regular operator pattern.

The structural comparison between the IRDS-YOLOv4-tiny baseline and YOLO-FSD is shown in Figure 3.

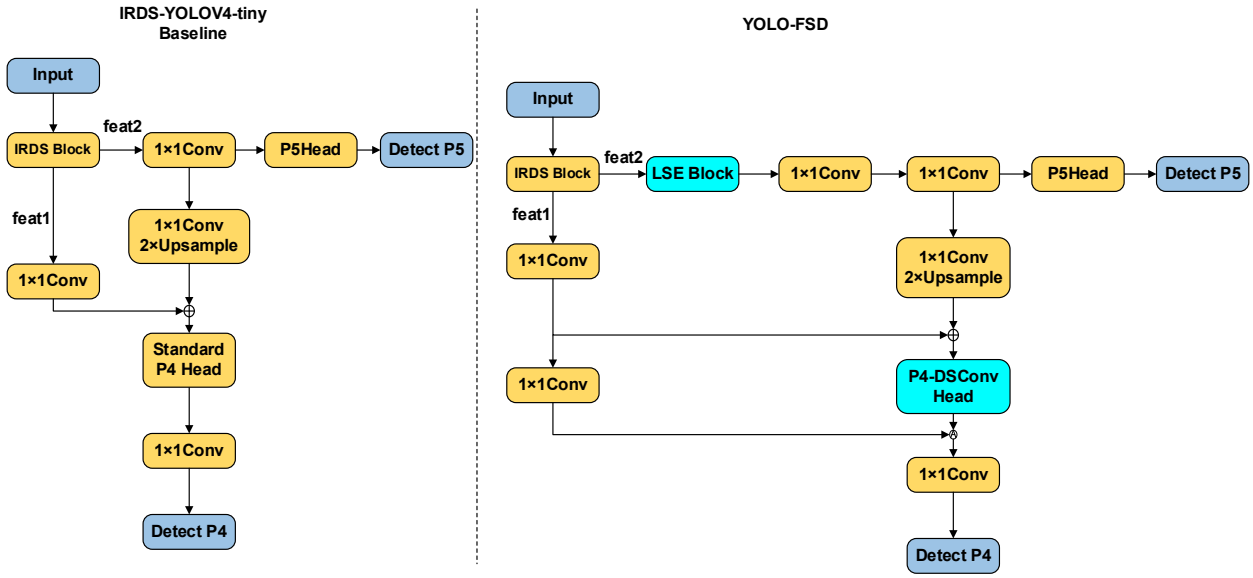


Figure 3. Structuralcomparison between the IRDS-YOLOv4-tiny baseline and YOLO-FSD.

2.8. FPGA-Oriented Deployment Workflow

The on-board validation was conducted using an ALINX AX7020 development board [48], which is built around the Xilinx Zynq-7000 XC7Z020-2CLG400I SoC [49]. The Zynq-7020 integrates a dual-core ARM Cortex-A9 processing system (PS) and programmable logic (PL), namely reconfigurable hardware that can be tailored to repeated neural-network operations. In this study, the PS coordinates input and weight loading, memory-buffer management, execution control, and layer scheduling, whereas the PL executes the main convolution kernels. This FPGA-based organization provides configurable parallelism, a customized fixed-point data path, and predictable hardware execution for

edge deployment. Its main trade-offs are limited digital signal processing (DSP) slices and block random-access memory (BRAM) resources, external-memory traffic, and a more complex hardware/software co-design flow. In comparison, GPU/NPU platforms such as Jetson modules generally provide higher out-of-the-box throughput and a simpler software deployment environment, whereas the FPGA allows more direct customization of operators, numerical precision, and dataflow. A photograph of the ZYNQ7020 deployment platform and its hardware connections is provided in Figure 4. The deployed system performs FPGA-side network forward inference and generates the P_4 and P_5 raw prediction tensors.

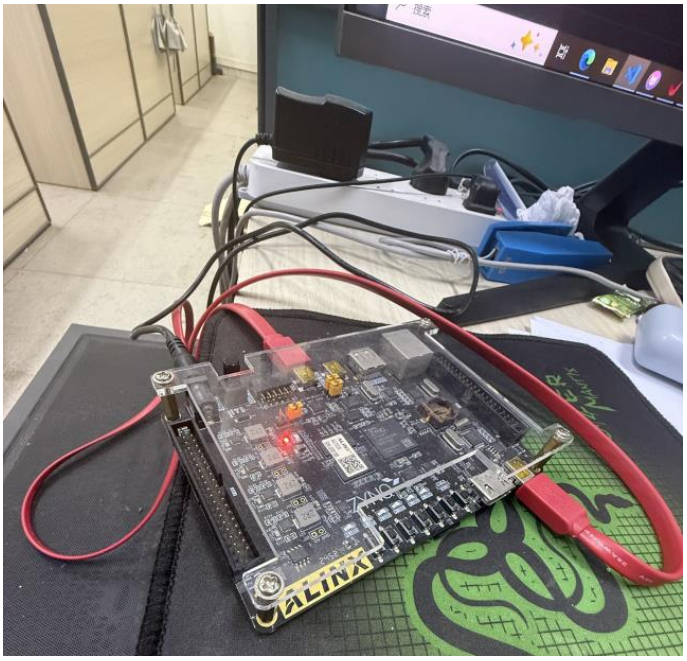


Figure 4. Physical ZYNQ7020 deployment platform and hardware connection.

To verify the implementability of YOLO-FSD on resource-constrained FPGA platforms, this study constructs a deployment-validation workflow from PyTorch model training, BN fusion, and INT16 fixed-point parameter export to on-board inference on the ZYNQ7020 and PC-side result evaluation. Considering the limited logic resources, DSP resources, and on-chip memory on the FPGA, the model design prioritizes regular operators such as 1×1 pointwise convolution, 3×3 depthwise convolution, ReLU6, upsampling, feature concatenation, and element-wise addition, while avoiding complex attention modules and dynamic branches. This improves fixed-point conversion and hardware mapping friendliness.

Unlike toolchain-based deployment using a general Deep-learning Processor Unit (DPU) [50], this study completes network inference through custom HLS convolution IP combined with layer-wise scheduling on the PS side. The overall deployment process includes two stages. The first stage is software-side parameter conversion and fixed-point export: BN fusion is performed on the trained PyTorch weights, and the fused weights and biases are exported as INT16 binary files according to the on-board fixed-point format, thereby reducing independent BN computation and parameter-reading overhead on the board. The second stage is on-board execution: basic convolution operators are implemented in Vivado HLS, IP integration and bitstream generation are completed in Vivado, and layer-wise whole-network inference is performed in Vitis through PS/PL cooperation. This workflow provides the implementation basis for subsequent FPGA-side output validation and deployment-performance analysis.

2.8.1. BN Fusion and Parameter Export

After model training, batch-normalization (BN) folding/fusion [51] is performed on the best PyTorch weights by absorbing the batch-normalization parameters into the preceding convolutional weights and bias. This converts each Conv-BN pair into an equivalent single convolution during inference, thereby reducing independent BN computation and intermediate memory access on the FPGA side.

The fused floating-point weights and biases are then exported as INT16 fixed-point binary files according to the data format used by the HLS implementation. A fixed scaling factor consistent with the on-board fixed-point representation is applied during export, and the quantized values are clipped to the valid INT16 range. Through this process, the trained PyTorch parameters are converted into FPGA-loadable fixed-point parameters, providing the parameter basis for on-board inference and output-consistency validation.

After Conv-BN fusion, the fused convolutional weights and biases were quantized using a signed INT16 fixed-point representation with a global scaling factor of $S = 512 = 2^9$, corresponding to a fixed fractional precision of 9 bits. Each floating-point value x was converted as:

$$q = \text{clip}(\text{round}(xS), -32768, 32767) \tag{22}$$

and de-quantized in simulation as:

$$\hat{x} = \frac{q}{S} \tag{23}$$

Thus, the representable real-valued range was approximately $[-64.0, 63.998]$, with a quantization step of $1/512$. The same scaling factor was applied globally to all fused weights and fused biases, rather than using layer-wise or channel-wise scaling. Biases were quantized after BN fusion using the same INT16 format. No calibration set was used to estimate dynamic ranges; the scale factor was fixed a priori to match the FPGA fixed-point configuration.

2.8.2. On-Board Execution Workflow

During hardware implementation, the low-level computation logic of basic operators, including standard convolution, pointwise convolution, and depthwise convolution, is first described in C/C++ with Vivado HLS. Data bit width, input/output interfaces, address transfer, and on-chip buffering are then designed. After C simulation, synthesis, and interface verification, the operator module is exported as an IP core callable by Vivado. Subsequently, the ZYNQ PS, DDR memory, AXI bus, and custom convolution IP are connected and configured in Vivado, and the bitstream file required for on-board execution is generated. The architecture of the configurable HLS convolution accelerator IP is shown in Figure 5.

The accelerator is implemented as a single Vivado HLS conv IP on a Zynq-7020 device. It supports standard 3×3 convolution, pointwise convolution, depthwise convolution, max-pooling, and nearest-neighbor upsampling through a runtime kernel selector. Feature maps and weights use a signed Q7.9 fixed-point representation ap_fixed<16,7>, corresponding to 7 integer bits including the sign bit and 9 fractional bits. Partial sums use 30-bit accumulators for standard and depthwise convolution and 32-bit accumulators for pointwise convolution. Convolution products and biases are converted to the corresponding accumulator format before accumulation and bias addition. During accumulator-to-output conversion, the result is requantized to Q7.9 using the Vivado HLS AP_RND rounding mode and AP_SAT saturation mode. Because the residual branches use the same Q7.9 scale, no additional rescaling is required before residual addition. The addition is performed in the wider arithmetic domain and converted back to the Q7.9 output format with saturation. For the LSE block, the residual addition is followed by ReLU6, corresponding

to the quantized output range [0, 3072]. Layer-wise overflow and clipping counters are not instrumented in the current FPGA implementation; therefore, per-layer overflow and clipping statistics are not reported.

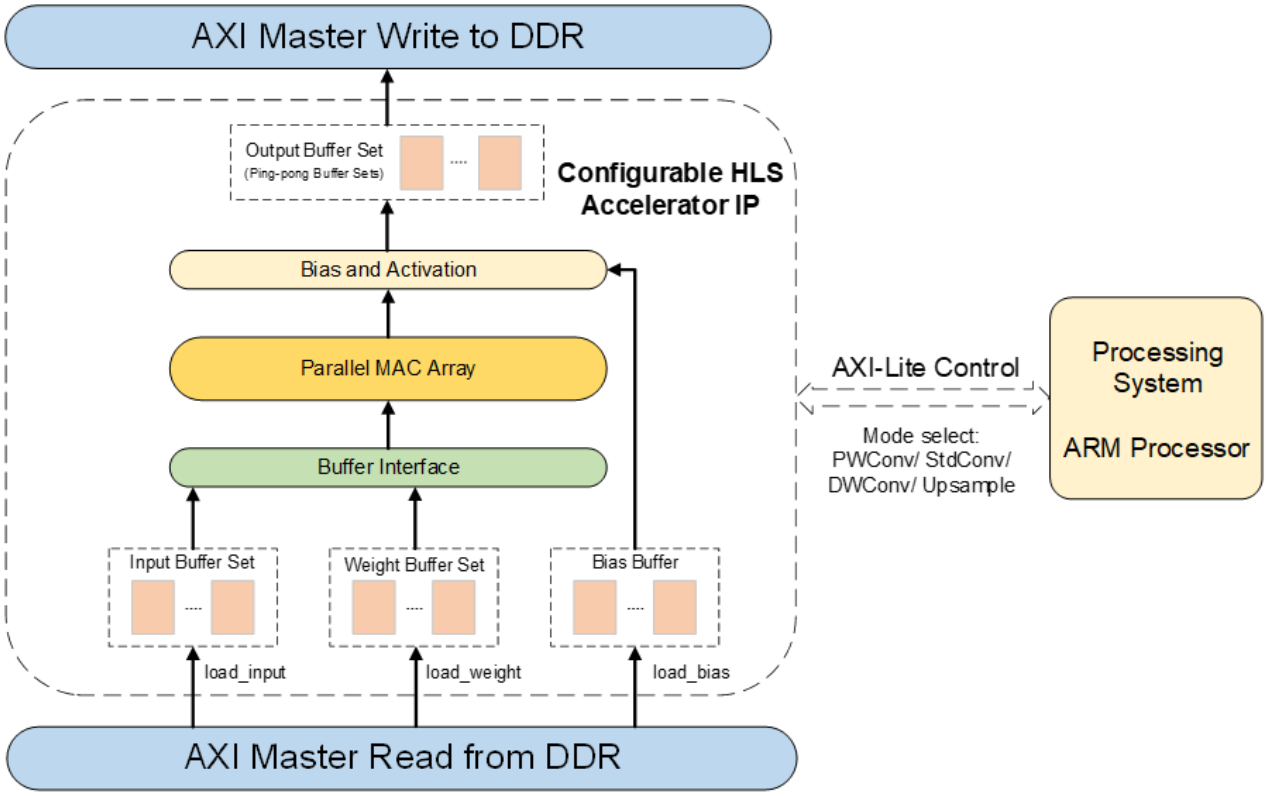


Figure 5. Architectureof the configurable HLS convolution accelerator IP.

The main tiling factors are Tr = Tc = 13, Tn = 4, and Tm = 4; standard and depthwise convolution use a 4 × 27 × 27 input tile and a 4 × 13 × 13 output tile, while pointwise convolution uses a 169-pixel tile. The design uses ping-pong on-chip buffers and pipelined load/compute/store loops. The AXI interface consists of four feature-map master ports and four weight/bias master ports with 32-bit AXI data width and 16-bit user payloads. No explicit DDR bandwidth model is assumed in the HLS code; bandwidth effects are captured in board-level timing. Upsampling is executed in PL through the same HLS IP, whereas residual/detail additions and tensor copy/concat operations are executed on the PS.

On the Vitis side, an application project is created based on the generated hardware platform, and the PS-side control program is developed for whole-network scheduling. During on-board inference, the PS is responsible for input data reading, weight-file loading, feature-map buffer management, convolution parameter configuration, branch synchronization, and output saving. The PL mainly executes compute-intensive convolution operations, including standard convolution, 1 × 1 pointwise convolution, and 3 × 3 depthwise convolution. For element-wise operations such as the residual connection in the LSE module, P_4 feature fusion, and shallow detail compensation, the PS organizes intermediate feature access and computation according to the network topology. The PS/PL cooperative deployment architecture on the ZYNQ7020 platform is shown in Figure 6.

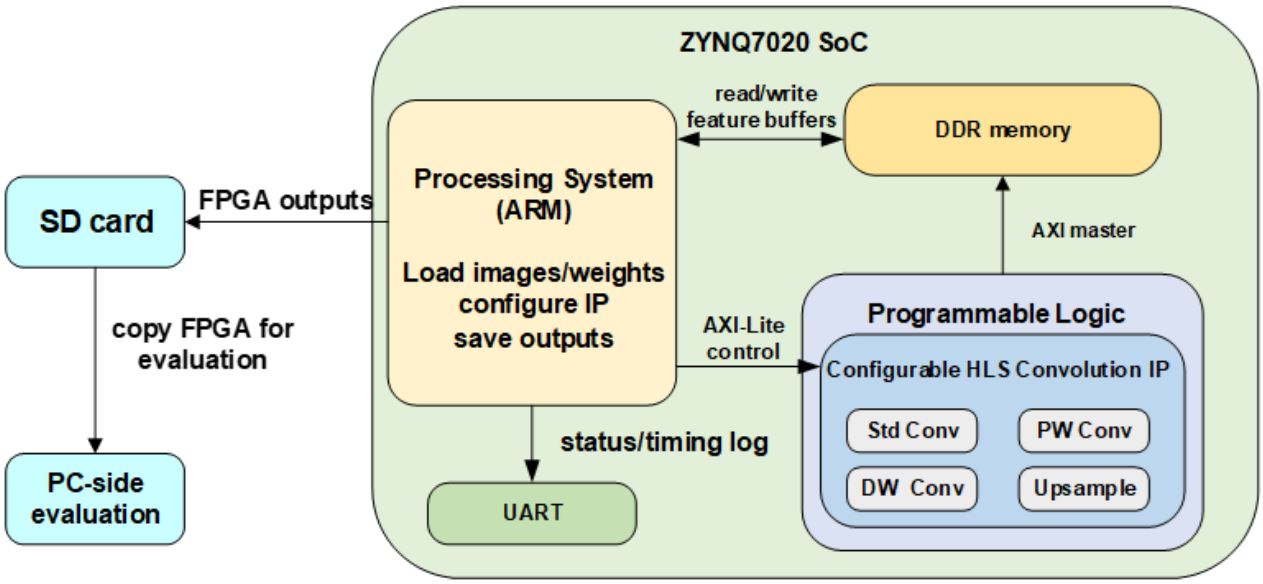


Figure 6. PS/PL cooperative deployment architecture on the ZYNQ7020 SoC.

During layer-wise inference, the PS configures the IP core with parameters such as input and output feature-map addresses, feature-map size, channel number, convolution type, stride, and weight address. After each layer is computed by the PL, the output feature map is written back to the assigned buffer and used for subsequent layers. In this way, the board completes IRDS backbone feature extraction, LSE semantic enhancement, and P_5/P_4 two-scale detection-head forward inference following the same topology as the PyTorch model. The generated raw outputs are then transferred to the PC side, where bounding-box decoding, NMS, and metric calculation are performed using the same anchors, class order, confidence threshold, NMS threshold, and evaluation protocol as the PyTorch side. The layer-wise scheduling and data interaction flow are shown in Figure 7.

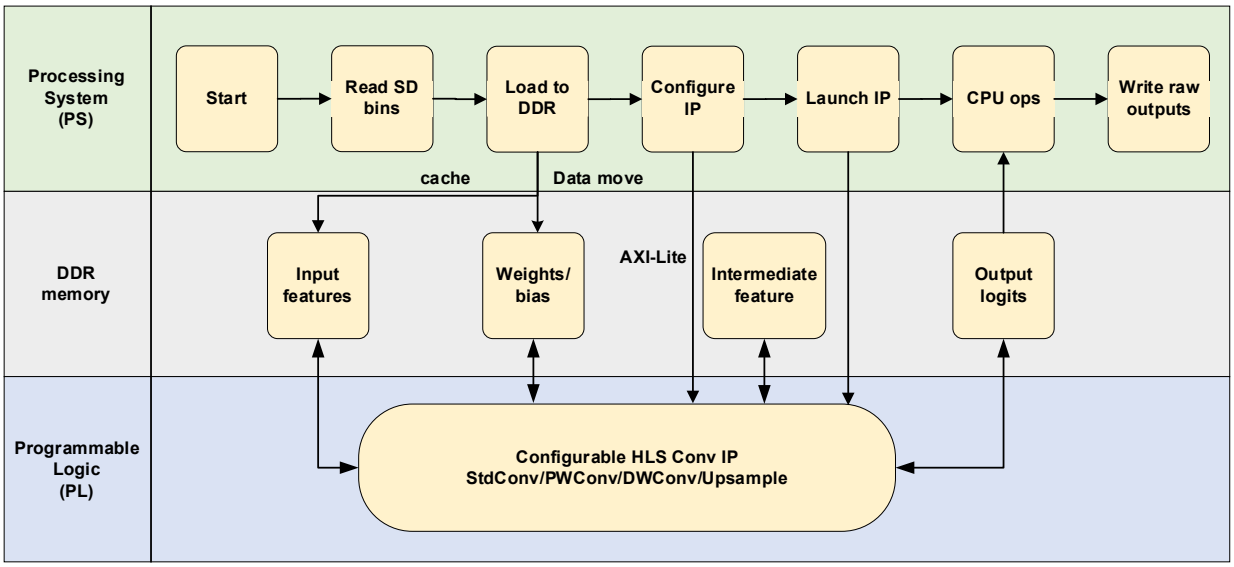


Figure 7. Layer-wise scheduling and data interaction flow during FPGA on-board inference.

3. Results and Discussion

3.1. Cross-Split Similarity Sensitivity Analysis

To examine whether the relative performance improvement depended strongly on cross-split scene similarity, the IRDS-YOLOv4-tiny baseline and the final YOLO-FSD were additionally compared on the conservative overlap-filtered subset of 2891 test images. The IRDS-YOLOv4-tiny baseline achieved an mAP50 of 60.80%, whereas YOLO-FSD achieved 62.24%, corresponding to an improvement of 1.44 percentage points. Thus, although image-level partitioning remains a limitation for interpreting the absolute performance values obtained on the main test set, the relative improvement of YOLO-FSD over the IRDS baseline was retained after substantially reducing potentially overlapping or highly similar scenes. This analysis is intended as a sensitivity check on the relative model comparison and should not be interpreted as establishing fully scene-independent generalization.

3.2. Detection Performance and Analysis

3.2.1. Comparison with Typical YOLO Detection Models

To ensure a fair comparison, all models reported in Table 2, including the original YOLOv4-tiny, YOLOv5s [52], YOLOv8n [38], YOLOv11n [41], YOLOv12n [42], and YOLO-FSD, were trained and evaluated using the same fixed training, validation, and test split, the same input resolution of 416 × 416, the same training settings, the same validation-based checkpoint-selection rule, and the same evaluation protocol. The results are shown in Table 2.

Table 2. Quantitative comparison with typical YOLO detection models.

Model	Input	mAP50/%	Params/M	MACs/G
YOLOv4-tiny	416 × 416	66.60	5.876	3.418
YOLO-FSD	416 × 416	69.36	3.951	1.520
YOLOv5s	416 × 416	70.59	7.016	3.330
YOLOv8n	416 × 416	70.86	3.006	1.708
YOLOv11n	416 × 416	70.91	2.624	1.397
YOLOv12n	416 × 416	71.03	2.603	1.405

As shown in Table 2, compared with the original YOLOv4-tiny, YOLO-FSD improves mAP50 from 66.60% to 69.36%, a gain of 2.76 percentage points (pt). The parameter count decreases from 5.876 M to 3.951 M, and MACs decrease from 3.418 G to 1.520 G, corresponding to reductions of 32.76% and 55.53%, respectively. Overall, the proposed structural reorganization improves the detection performance of the selected YOLOv4-tiny baseline while substantially reducing its complexity and retaining a regular operator and dataflow pattern suitable for the reported ZYNQ7020 forward-inference validation.

YOLOv5s, YOLOv8n, YOLOv11n, and YOLOv12n achieve mAP50 values of 70.59%, 70.86%, 70.91%, and 71.03%, respectively, which are 1.23, 1.50, 1.55, and 1.67 pt higher than that of YOLO-FSD. However, the CSP-style branching and concatenation dataflow, SPPF/SiLU activations, and attention-related transformations were used, and the $P_3/P_4/P_5$ three-scale prediction adopted by these models introduces additional intermediate-feature buffering, memory access, nonlinear-function implementation, and scheduling requirements in a custom fixed-point HLS/IP workflow for resource-constrained FPGAs. Consequently, although these models provide higher software-side accuracy and, in some cases, more favorable raw complexity metrics, parameter count and MACs alone should not be interpreted as evidence of lower deployment cost or easier mapping for the current FPGA implementation. Because the recent nano-scale models were not implemented under the same ZYNQ7020 custom HLS/IP workflow, the present software-side comparison does not

establish a hardware-efficiency advantage of YOLO-FSD over these models. The structural discussion above therefore serves as a deployment-oriented architectural rationale rather than a controlled hardware-performance conclusion.

3.2.2. Ablation Experiment and Complexity Comparison

To address the relatively large parameter count and computational cost of the original YOLOv4-tiny, an IRDS backbone was first used to construct the IRDS-YOLOv4-tiny baseline. The LSE module, P_4-DSConv detection head, and shallow detail-compensation branch were then introduced progressively. Accordingly, this experiment evaluates the predefined cumulative design sequence used to construct YOLO-FSD rather than a complete factorial combination of all components. The observed differences between adjacent configurations should therefore be interpreted as incremental changes within this design sequence, rather than as fully isolated causal effects of individual modules. The ablation analysis considers both detection performance and model complexity. All configurations reported in Table 3, including the original YOLOv4-tiny, the IRDS-YOLOv4-tiny baseline, the cumulative ablation variants, and the final YOLO-FSD, use the same fixed training, validation, and test split, input size, training settings, checkpoint-selection criterion, and evaluation protocol. The results are reported in Table 3.

Table 3. PyTorch-side ablation results.

Model	mAP50/%	Recall/%	*F*1/%	Params/M	MACs/G	ΔmAP50/pt
YOLOv4-tiny	66.60	59.45	62.42	5.876	3.418	−1.15
IRDS- YOLOv4-tiny	67.75	67.57	66.42	4.248	1.932	0.00
+LSE	68.68	69.85	67.07	4.667	2.004	+0.93
+LSE+P_4-DSConv	68.84	68.69	66.98	3.885	1.475	+1.09
YOLO-FSD	69.36	70.15	67.19	3.951	1.520	+1.61

Note: ΔmAP50 denotes the improvement over the IRDS-YOLOv4-tiny baseline, in percentage points.

As shown in Table 3, replacing the original YOLOv4-tiny backbone with the IRDS design reduces the parameter count from 5.876 M to 4.248 M and MACs from 3.418 G to 1.932 G, corresponding to reductions of 27.71% and 43.48%, respectively. At the same time, mAP50 increases from 66.60% to 67.75%, Recall increases from 59.45% to 67.57%, and *F*1 increases from 62.42% to 66.42%. Thus, the IRDS-based backbone reduces model complexity while improving all three detection metrics under the same evaluation protocol, providing a lightweight reference for the subsequent ablations.

After the LSE module is introduced into the IRDS-YOLOv4-tiny baseline, mAP50, Recall, and *F*1 increase to 68.68%, 69.85%, and 67.07%, respectively. These values correspond to gains of 0.93, 2.28, and 0.65 pt over the IRDS baseline, while the parameter count and MACs increase slightly to 4.667 M and 2.004 G. Within the cumulative design sequence, introducing LSE at the deep feat2 feature is associated with improved detection performance. The present experiment is intended to evaluate the contribution of LSE within the proposed YOLO-FSD design sequence rather than to establish superiority over all alternative inverted residual configurations. When P_4-DSConv is further introduced to reduce the computational cost of the high-resolution detection head, the parameter count and MACs decrease to 3.885 M and 1.475 G, which are reductions of 16.76% and 26.40%, respectively, relative to the LSE-only configuration. The mAP50 value remains essentially unchanged, whereas Recall and *F*1 decrease by 1.16 and 0.09 pt, respectively. P_4-DSConv substantially reduces the computational cost of the high-resolution P_4 head while preserving essentially the same mAP50. The small Recall and *F*1 decreases at the fixed operating threshold indicate a limited trade-off in local-detail sensitivity.

With the shallow detail-compensation branch added, the final YOLO-FSD model achieves an mAP50 of 69.36%, a Recall of 70.15%, and an *F*1 of 67.19%. Compared with the +LSE+P_4-DSConv configuration, the final model improves mAP50, Recall, and *F*1 by 0.52, 1.46, and 0.21 pt, respectively, while adding only 0.066 M parameters and 0.045 G MACs. After the shallow detail-compensation branch is added to the cumulative +LSE+P_4-DSConv configuration, the final model shows recovery in Recall and *F*1 together with a modest increase in mAP50. This observation supports the role of detail compensation within the final cumulative design, but does not constitute an independently isolated estimate of the branch contribution. Relative to the IRDS-YOLOv4-tiny baseline, YOLO-FSD improves mAP50, Recall, and *F*1 by 1.61, 2.58, and 0.77 pt, respectively, while retaining 6.99% fewer parameters and 21.33% fewer MACs.

Compared with the original YOLOv4-tiny, YOLO-FSD improves mAP50 by 2.76 pt while reducing the parameter count and MACs by 32.76% and 55.53%, respectively. Overall, the results demonstrate that the coordinated design of the IRDS backbone, LSE, P_4 detection-head lightweighting, and shallow detail compensation produces a favorable accuracy-complexity trade-off.

The repeated-training results are summarized in Table 4. Across the three independent runs, the mAP50 standard deviation ranges from 0.21 to 0.25 pt. The three-run mean mAP50 increases from 67.72% for the IRDS-YOLOv4-tiny baseline to 69.33% for YOLO-FSD, corresponding to an absolute difference of 1.61 pt. The smaller differences between adjacent cumulative configurations should be interpreted cautiously. In particular, the 0.52-pt mean difference between the +LSE+P_4-DSConv configuration and YOLO-FSD is treated as a modest cumulative improvement rather than evidence of an independently statistically significant contribution from the detail-compensation branch.

Table 4. Repeated-training results of the principal ablation configurations over three independent runs.

Configuration	mAP50/%	Recall/%	*F*1/%
IRDS- YOLOv4-tiny	67.72 ± 0.25	67.12 ± 0.43	66.22 ± 0.22
+LSE	68.65 ± 0.24	69.40 ± 0.45	66.87 ± 0.21
+LSE+P_4-DSConv	68.81 ± 0.24	68.24 ± 0.48	66.78 ± 0.22
YOLO-FSD	69.33 ± 0.21	69.70 ± 0.41	66.99 ± 0.18

3.2.3. Class- and Scale-Wise Changes After Introducing LSE

The preceding ablation results show that introducing the LSE module into the IRDS-YOLOv4-tiny baseline increases mAP50 from 67.75% to 68.68%, Recall from 67.57% to 69.85%, and *F*1 from 66.42% to 67.07%, corresponding to gains of 0.93, 2.28, and 0.65 pt, respectively. To further examine whether this improvement is associated with particular classes or target scales, the class-wise and scale-wise detection results before and after introducing LSE are compared under the same dataset split, input size, and evaluation protocol. The results are shown in Table 5.

Table 5. Class-wise and scale-wise detection results before and after introducing the LSE module.

Evaluation Object	Fire AP50/%	Fire Recall/%	Smoke AP50/%	Smoke Recall/%	Small AP50/%	Medium AP50/%	Large AP50/%
IRDS YOLOv4-tiny	63.47	59.20	72.03	75.94	23.96	46.89	66.02
+LSE	64.42	62.55	72.93	77.15	24.76	47.10	66.15
Difference	+0.95	+3.35	+0.90	+1.21	+0.80	+0.21	+0.13

Note: +LSE indicates that the LSE module is introduced at the feat2 deep feature layer of the IRDS-YOLOv4-tiny baseline.

As shown in Table 5, LSE produces positive changes for both Fire and Smoke. For Fire, AP50 increases from 63.47% to 64.42%, and Recall increases from 59.20% to 62.55%, corresponding to gains of 0.95 and 3.35 pt, respectively. For Smoke, AP50 increases from 72.03% to 72.93%, while Recall increases from 75.94% to 77.15%, corresponding to gains of 0.90 and 1.21 pt. The larger Recall gain for Fire indicates that LSE is particularly helpful for reducing missed fire detections at the fixed operating threshold, whereas the positive AP50 and Recall changes for both classes show that the module does not introduce an evident class-specific degradation under the present evaluation protocol.

For target-scale analysis, AP50 increases from 23.96% to 24.76% for Small targets, from 46.89% to 47.10% for Medium targets, and from 66.02% to 66.15% for Large targets, corresponding to gains of 0.80, 0.21, and 0.13 pt, respectively. Thus, the scale-wise effect is positive but modest, with the largest improvement observed for Small targets. This pattern is consistent with deep semantic information at feat2 being propagated to the high-resolution P_4 branch through upsampling and feature fusion, which can provide additional context for small flames, thin smoke, and weak-boundary targets.

Overall, the performance changes observed after introducing LSE are not confined to a single class or target scale, although their magnitude varies across the reported metrics. The class-wise results show a clearer Recall improvement, especially for Fire, while the scale-wise AP50 results show the most noticeable gain for Small targets.

3.2.4. Class-Wise and Scale-Wise Detection Performance of YOLO-FSD

To further analyze the detection performance of YOLO-FSD for different classes and target scales, the final model is compared with the IRDS-YOLOv4-tiny baseline. The class-wise and scale-wise results are reported in Tables 6 and 7, respectively.

Table 6. Class-wise object detection comparison for YOLO-FSD.

Class	Model	AP50/%	Recall/%	*F*1/%
Smoke	Baseline	72.03	75.94	69.49
Smoke	YOLO-FSD	73.56	77.12	69.62
Smoke	Difference	+1.53	+1.18	+0.13
Fire	Baseline	63.47	59.20	63.35
Fire	YOLO-FSD	65.16	63.18	64.76
Fire	Difference	+1.69	+3.98	+1.41

Table 7. Scale-wise detection comparison for YOLO-FSD.

Size	GT Num	Baseline mAP50/%	YOLO-FSD mAP50/%	ΔmAP50 /pt	Baseline Recall/%	YOLO-FSD Recall/%	ΔRecall/pt
Small	1956	23.96	26.59	+2.63	49.82	53.68	+3.86
Medium	1782	46.89	47.11	+0.22	66.98	68.49	+1.51
Large	2139	66.02	67.26	+1.24	81.64	83.24	+1.60

Note: Δ denotes the improvement of YOLO-FSD over Baseline, in percentage points.

As shown in Table 6, YOLO-FSD produces positive changes for both Smoke and Fire, although the magnitude of the improvement differs between the two classes. For Smoke, AP50 increases from 72.03% to 73.56%, Recall increases from 75.94% to 77.12%, and *F*1 increases from 69.49% to 69.62%, corresponding to gains of 1.53, 1.18, and 0.13 pt, respectively. These results indicate a modest improvement in smoke detection, particularly in AP50 and Recall, while the fixed-threshold *F*1 remains nearly unchanged.

For Fire, AP50 increases from 63.47% to 65.16%, Recall increases from 59.20% to 63.18%, and *F*1 increases from 63.35% to 64.76%, corresponding to gains of 1.69, 3.98, and 1.41 pt,

respectively. The more pronounced Recall gain for Fire suggests that YOLO-FSD reduces missed detections of flame targets at the fixed operating threshold. Overall, the class-wise results support an improvement for both Smoke and Fire, with a stronger effect on the recall and $F1$ of Fire.

As shown in Table 7, YOLO-FSD obtains higher mAP50 and Recall than the baseline in all three size bins. The largest scale-wise improvement occurs for Small targets: mAP50 increases from 23.96% to 26.59%, and Recall increases from 49.82% to 53.68%. This result indicates the clearest benefit for small flames, thin smoke, and weak-boundary targets. For Medium targets, mAP50 changes only from 46.89% to 47.11%, whereas Recall increases from 66.98% to 68.49%. For Large targets, mAP50 and Recall increase from 66.02% and 81.64% to 67.26% and 83.24%, corresponding to gains of 1.24 and 1.60 pt, respectively.

The class-wise and scale-wise results therefore indicate that the benefit of YOLO-FSD is not confined to a single class or target scale, but its magnitude is uneven across scales. The strongest scale-wise gain is observed for Small targets, while the mAP50 change for Medium targets is marginal and the improvements for Large targets are moderate. Taken together with the ablation results in Table 3 and the module-level analysis in Table 5, these results indicate that the full YOLO-FSD design improves detection performance for both classes and across target scales while maintaining low model complexity.

3.2.5. BN Fusion, INT16 Parameter-Quantization Sensitivity and FPGA Output Analysis

To compare detection performance across the deployment workflow, four stages are compared: the original PyTorch FP32 model, the BN-fused FP32 model, software INT16 parameter-quantization evaluation, and FPGA raw-output evaluation. The comparison examines the effects of BN fusion, software-side INT16 quantization sensitivity, and FPGA-side forward inference on detection performance. The results are reported in Tables 8 and 9.

Table 8. Detection performance across deployment stages.

Stage	mAP50/%	Precision/%	Recall/%	*F*1/%
Original PyTorch FP32	69.36	64.94	70.15	67.19
BN-fused FP32	69.33	64.86	70.08	67.11
Software INT16 parameter quantization	69.32	64.73	70.07	67.07
FPGA	69.14	64.48	69.58	66.93

Table 9. Class-wise detection comparison between PyTorch and FPGA.

Class	Platform	AP50/%	Precision/%	Recall/%	*F*1/%
Fire	PyTorch	65.16	66.42	63.18	64.76
Fire	FPGA	65.12	65.61	62.89	64.52
Fire	Difference	−0.04	−0.81	−0.29	−0.24
Smoke	PyTorch	73.56	63.46	77.12	69.62
Smoke	FPGA	73.16	63.35	76.27	69.34
Smoke	Difference	−0.40	−0.11	−0.85	−0.28

All stages in Table 8 were evaluated using the same final checkpoint, the same 4810-image test set, and identical image preprocessing and PC-side DecodeBox/NMS settings. The FPGA results were obtained by decoding the board-generated P_4 and P_5 raw output tensors using the same PC-side post-processing procedure. The software INT16 parameter-quantization evaluation quantizes the BN-fused weights and biases to INT16 and then performs inference in PyTorch after dequantization. It is used as a software quantization-sensitivity evaluation and is not claimed to be a bit-accurate reproduction of all FPGA fixed-

point operations, including intermediate activation quantization, accumulation, rounding, and saturation.

As shown in Table 8, the original PyTorch FP32 model obtains an mAP50 of 69.36% and an $F1$ score of 67.19%. After BN fusion, mAP50 and $F1$ are 69.33% and 67.11%, respectively, indicating only minor changes. Software INT16 parameter-quantization evaluation yields an mAP50 of 69.32% and an $F1$ score of 67.07%, corresponding to decreases of 0.04 and 0.12 percentage points (pt), respectively, relative to the original PyTorch FP32 model. The FPGA raw-output evaluation achieves an mAP50 of 69.14% and an $F1$ score of 66.93%, corresponding to decreases of 0.22 and 0.26 pt, respectively. These results demonstrate the feasibility of the FPGA-side forward-inference and raw-output-generation workflow. They do not represent a fully on-board end-to-end detection system because bounding-box decoding and NMS are performed on the PC.

Table 9 further presents the class-wise comparison between the PyTorch and FPGA evaluations. For Fire, AP50 changes from 65.16% to 65.12%, a decrease of 0.04 pt, while Precision, Recall, and $F1$ change by -0.81, -0.29, and -0.24 pt, respectively. For Smoke, AP50 changes from 73.56% to 73.16%, a decrease of 0.40 pt, while Precision, Recall, and $F1$ change by -0.11, -0.85, and -0.28 pt, respectively. The class-wise changes remain limited at the reported operating threshold, although their magnitudes differ between Fire and Smoke.

3.2.6. Resource Utilization, Latency, and Power Consumption

After comparing the PyTorch-side and FPGA-side detection results, this section further evaluates the hardware behavior of YOLO-FSD on the ZYNQ7020 platform from the perspectives of same-board ablation performance, implementation resource utilization, forward-inference timing characteristics, power and energy efficiency, and cross-study deployment context. Table 10 first reports the FPGA forward-inference results of the predefined ablation configurations under the same INT16 custom HLS/IP workflow. Table 11 then summarizes the implementation resource utilization. Table 12 presents the measured forward-inference timing profile of the final YOLO-FSD model, while Table 13 summarizes its timing, power, and energy-efficiency metrics. Finally, Table 14 provides implementation context through comparison with previously reported FPGA-based YOLO deployments.

Table 10. Same-board FPGA forward-inference ablation results under a common INT16 custom HLS/IP workflow.

Model	FPGA Raw-Output mAP50	MACs/G	Forward-Only Latency (ms)	Achieved GOPS	GOPS/DSP
IRDS- YOLOv4-tiny	67.56	1.932	1861.24	2.076	0.01160
+LSE	68.46	2.004	1920.49	2.087	0.01166
+LSE+P_4-DSConv	68.26	1.475	1663.51	1.773	0.00991
YOLO-FSD	69.14	1.520	1712.61	1.775	0.00992

Note: All configurations use the same ZYNQ7020 platform, INT16 format, and forward-only timing boundary. GOPS/DSP is calculated using 179 DSP slices.

Table 11. Resource utilization on the ZYNQ7020 platform.

Resource	Used	Available	Utilization/%
LUT	29,719	53,200	55.86%
FF	35,894	106,400	33.73%
BRAM	76	140	54.29%
DSP	179	220	81.36%

Table 12. Measured forward-inference timing profile of YOLO-FSD on ZYNQ7020.

Component	Executor/Timing Category	Mean Latency (ms)	Share (%)
Forward total	PS + PL	1712.61	100.00
PL-kernel wait interval	PL + associated memory path	1711.25	99.92
PS non-wait interval	PS	1.36	0.08
Backbone stem to P_4	PL-associated forward stage	1175.45	68.63
Backbone P_5 body	PL-associated forward stage	200.10	11.68
P_5 head stem	PL-associated forward stage	106.74	6.23
P_4 head stem	PL-associated forward stage	61.90	3.61
Deep semantic enhancement	PL-associated forward stage	59.26	3.46
Detail convolution	PL-associated forward stage	31.89	1.86
Other stages	PL-associated forward stages	77.28	4.51

Note: The PL-kernel wait interval includes accelerator execution and associated memory/synchronization overhead rather than pure PL arithmetic time. Timing category and network stage rows are alternative decompositions of the same forward-inference interval and should not be summed.Percentages were calculated using the unrounded mean latency values. Minor discrepancies may occur due to rounding of the displayed values.

Table 13. Latency, power consumption, energy efficiency, and timing results on the ZYNQ7020.

Metric	Results
Platform	ZYNQ7020
Frequency/MHz	120
WNS/ns	0.172
Forward-only latency/ms	1712.61
FPGA forward-inference FPS	0.584
Estimated FPS/W	0.224
Vivado estimated on-chip Power/W	2.603
Estimated energy/frame/J	4.458

Note: Forward-only latency covers network forward inference and raw-output generation. WNS denotes worst negative slack; WNS > 0 indicates that the implemented design meets the target timing constraint.

Table 14. Reported deployment characteristics of selected FPGA-based YOLO studies.

Work	Model/FPGA Platform	Latency/ms	GOPS	DSP Used	GOPS/DSP
Xu et al. [53]	YOLOv4-tiny/ZYNQ-7020	376	9.24	216	0.0428
Yan et al. [34]	YOLOv5s/ZCU102	12.633	808.6	1647	0.49
Montgomerie-Corcoran et al. [54] (SATAY, 2023)	YOLOv5s/VCU110	46.4	392.0	1794	0.22
YOLO-FSD (ours)	YOLO-FSD/ZYNQ-7020	1712.61	1.775	179	0.00992

Note: Xu et al. [53] use 416 × 416 input and 16-bit fixed-point arithmetic on the ZYNQ-7020, but their exact timing boundary and MAC-counting convention are not explicitly specified. YOLO-FSD reports forward-only latency and calculates GOPS using two operations per MAC. Therefore, GOPS and GOPS/DSP are provided for implementation context rather than strictly normalized comparison.

Table 10 presents the same-board FPGA forward-inference results of the four predefined ablation configurations. All configurations were implemented on the same ZYNQ7020 platform using the same INT16 representation, 120-MHz clock frequency, custom HLS/IP accelerator, and forward-only timing boundary, thereby allowing the effects of the cumulative structural modifications to be compared under a common hardware workflow. The IRDS-YOLOv4-tiny baseline achieves an FPGA raw-output mAP50 of 67.56% with a forward-only latency of 1861.24 ms. After introducing LSE, the FPGA raw-output mAP50 increases to 68.46%, corresponding to a gain of 0.90 percentage points, while the latency increases by 59.25 ms to 1920.49 ms because of the additional semantic-enhancement operations. When P_4-DSConv is subsequently introduced, MACs decrease from 2.004 G to

1.475 G, and the forward-only latency is reduced by 256.98 ms to 1663.51 ms, while the FPGA raw-output mAP50 changes only slightly from 68.46% to 68.26%. Finally, adding the shallow detail-compensation branch increases the FPGA raw-output mAP50 by 0.88 percentage points to 69.14%, while introducing a limited latency increase of 49.10 ms, resulting in a final forward-only latency of 1712.61 ms. Overall, the same-board results show that LSE improves detection performance at a moderate latency cost, P_4-DSConv substantially reduces the computational and latency burden of the high-resolution detection branch, and the detail-compensation branch recovers and further improves detection performance with a relatively small additional latency overhead. These results characterize the accuracy-latency trade-off of the proposed cumulative design under a consistent FPGA deployment workflow.

The same-board ablation results in Table 10 demonstrate the accuracy-latency trade-offs introduced by the cumulative structural modifications under a consistent FPGA deployment workflow. To further examine whether the final YOLO-FSD implementation can be accommodated within the resource constraints of the ZYNQ7020 device, the post-implementation utilization of the principal FPGA resources is summarized in Table 11.

As shown in Table 11, YOLO-FSD uses 29,719 look-up tables (LUTs), 35,894 flip-flops (FFs), 76 BRAMs, and 179 DSP slices, corresponding to utilization rates of 55.86%, 33.73%, 54.29%, and 81.36%, respectively. These results indicate that the current INT16 implementation fits within the available resources of the ZYNQ7020 platform. Among the reported resources, DSP slices exhibit the highest utilization rate and therefore represent the most constrained hardware resource in the current implementation, although the available DSP capacity is not fully exhausted.

After confirming that the final YOLO-FSD implementation fits within the available ZYNQ7020 resources, the forward-inference timing behavior was further analyzed to examine how the execution time is distributed across the deployed network. FPGA latency was measured using the Zynq global timer operating at 333,333,343 counts/s. Five warm-up runs were performed before formal measurement, after which each of the 4810 test images was measured three times. The three measurements for each image were first averaged to obtain an image-level latency, and the reported overall latency was then calculated as the mean of the 4810 image-level values; timing variability was characterized using the sample standard deviation across these image-level mean latencies. The primary timing metric used in this study is the forward-only latency. Its timing boundary includes PS-side network orchestration, PL accelerator execution, DDR-related feature and parameter transfers required during network execution, cache-management operations associated with the forward path, and PS/PL synchronization, while excluding SD card I/O, host communication, and PC-side DecodeBox/NMS processing. Accordingly, the reported forward-only latency characterizes FPGA-side network forward inference and raw-output generation rather than complete end-to-end detection latency.

As shown in Table 12, the mean forward-only latency of YOLO-FSD is 1712.61 ms per frame. Across the 4810 per-image measurements, the corresponding sample standard deviation was 0.09 ms. From the timing-category perspective, the PL-kernel wait interval accounts for 1711.25 ms, or 99.92% of the measured forward interval, whereas the PS non-wait interval accounts for only 1.36 ms, or 0.08%. However, the PL-kernel wait interval includes accelerator execution together with the associated external-memory data path and synchronization observed while the PS waits for kernel completion and therefore should not be interpreted as pure PL arithmetic execution time.

From the network-stage perspective, the backbone stem-to-P_4 stage contributes the largest portion of the forward latency, requiring 1175.45 ms and accounting for 68.63% of the total forward interval. The backbone P_5 body requires 200.10 ms (11.68%), followed

by the P_5 head stem at 106.74 ms (6.23%) and the P_4 head stem at 61.90 ms (3.61%). The deep semantic-enhancement stage requires 59.26 ms (3.46%), while the detail-convolution stage contributes 31.89 ms (1.86%). The remaining stages account for 77.28 ms (4.51%). These results indicate that the measured forward latency is dominated primarily by the backbone-associated computation and its accompanying data movement, whereas the additional LSE and detail-compensation operations account for relatively small portions of the total forward interval.

In addition to the primary forward-only measurement, a second raw-SD-pipeline timing was recorded to characterize the additional I/O overhead. This measurement additionally includes SD card input reading and raw-output writing together with their associated cache-management operations. The average RAW SD pipeline latency was 2155.81 ms per frame, compared with the forward-only latency of 1712.61 ms, corresponding to an additional 443.20 ms. Because this interval jointly contains SD card access, raw-output writing, and their associated cache-management overhead, the present measurement does not separately attribute the difference to individual I/O components. PC-side DecodeBox/NMS processing remains outside the synchronized ZYNQ timing procedure; therefore, a reproducible end-to-end detection latency is not reported.

The overall timing, power, and energy-efficiency characteristics of the final YOLO-FSD implementation are further summarized in Table 13. The implemented system operates at 120 MHz with a worst negative slack (WNS) of 0.172 ns, indicating that the design satisfies the target timing constraint. Under the forward-only timing boundary defined above, YOLO-FSD achieves a mean latency of 1712.61 ms per frame, corresponding to an FPGA forward-inference rate of 0.584 FPS.

The Vivado routed on-chip power estimate is 2.603 W. This value is a tool-derived on-chip power estimate rather than a board-level power measurement obtained using an external power meter or an on-board power sensor. Accordingly, the energy-per-frame and FPS/W values reported in Table 13 are also estimate-derived metrics. Based on the measured forward-only latency and the Vivado power estimate, the estimated energy consumption is 4.458 J per frame, while the corresponding energy efficiency is 0.224 FPS/W. These results characterize the timing feasibility and estimated power-efficiency level of the current ZYNQ7020 implementation; however, the reported power-related metrics should be interpreted as implementation estimates rather than direct measurements of total board-level power consumption.

Table 14 places YOLO-FSD in the context of previously reported FPGA-based YOLO deployments. The YOLO-FSD implementation reports a forward-only latency of 1712.61 ms, an achieved throughput of 1.775 GOPS, and a DSP usage of 179 slices on the ZYNQ-7020 platform. Table 14 is intended to provide implementation context rather than a strictly normalized cross-study ranking. For the compared implementations on different FPGA platforms, the reported latency, GOPS, and GOPS/DSP values are affected by differences in input configuration, numerical precision, accelerator architecture, memory system, timing boundary, and operation-counting convention. However, the comparison with Xu et al. [53] requires separate consideration because both studies use the ZYNQ-7020 platform and a 416 $\times$ 416 input. Therefore, the specific causes of the performance gap with Xu et al., including numerical precision, timing definition, operation-counting convention, channel-level parallelism, DSP utilization, and PS/PL-memory dataflow, are analyzed separately below.

Particular attention should be paid to the comparison with Xu et al. [53], because both implementations use the ZYNQ-7020 and a 416 $\times$ 416 input. Xu et al. employ 16-bit fixed-point arithmetic and report a latency of 0.376 s per image, a throughput of 9.24 GOPS, and 216 of 220 DSP slices (98.18%). Their convolution accelerator applies input- and output-

channel parallelism, pipelining, and ping-pong buffering, with maximum parallel factors of $T_n = 4$ and $T_m = 32$. In contrast, the present YOLO-FSD implementation uses $T_m = T_n = 4$, 179 DSP slices (81.36%), a single configurable HLS convolution IP, and layer-wise PS/PL scheduling. Residual/detail additions and tensor copy/concatenation are executed on the PS, while intermediate feature maps are exchanged through the DDR/AXI data path. These implementation differences indicate that the substantially lower throughput of YOLO-FSD cannot be attributed solely to different model complexity or measurement conventions. The lower channel-level parallelism, lower DSP utilization, and current layer-wise data movement and scheduling strategy constitute genuine efficiency limitations of the present accelerator implementation.

The timing and GOPS definitions are also not fully identical between the two studies. Xu et al. report an average calculation time of 0.376 s per image and describe an ARM-FPGA system in which the ARM performs image preprocessing, upsampling, and YOLO post-processing; however, the published paper does not explicitly specify the exact inclusion/exclusion boundary of these operations in the reported 0.376 s timing. Similarly, although 9.24 GOPS is reported, the paper does not explicitly state whether one multiply-accumulate operation is counted as one or two operations. In the present study, the 1712.61 ms value is explicitly defined as forward-only latency including PS orchestration, PL accelerator execution, DDR/cache maintenance, and PS/PL synchronization, while excluding SD card I/O, host communication, and PC-side DecodeBox/NMS. The YOLO-FSD throughput of 1.775 GOPS is calculated using two operations per MAC. Therefore, the absolute GOPS values should not be interpreted as a strictly normalized head-to-head comparison.

Overall, the same-board measurements demonstrate that YOLO-FSD can complete network forward inference and raw-output generation on the resource-constrained ZYNQ7020 FPGA under the reported INT16 custom HLS/IP workflow. However, the current forward-inference FPS of 0.584 remains insufficient for real-time video-stream processing.

3.2.7. Robustness, Failure Cases and Deployment Boundaries

A limited 1080p feasibility test was additionally conducted to examine the effect of whole-frame downscaling and native-resolution tiling. Eight annotated 1920 × 1080 images were evaluated using the same deployed model and FPGA accelerator. Whole-frame resizing to 416 × 416 achieved an mAP50 of 33.97%, whereas serial processing of 15 native-resolution 416 × 416 tiles achieved 6.90%. The corresponding FPGA forward-only latency increased from 1.713 s to 25.693 s per source frame. These results indicate that exhaustive native-resolution tiling is not an effective or practical solution under the current deployment configuration because it changes the target-scale/context distribution and introduces substantial latency. This small-scale experiment, based on only eight annotated 1920 × 1080 images, is intended solely as a preliminary feasibility observation of the current resizing and tiling strategies and should not be interpreted as a full high-resolution benchmark or as quantitative evidence of high-resolution generalization.

To evaluate the sensitivity of YOLO-FSD to controlled visual perturbations, brightness decrease, Gaussian blur, Gaussian noise, haze/fog, and local occlusion were applied to the test set. Specifically, brightness decrease was implemented using a brightness factor of 0.60. Gaussian blur was applied using PIL GaussianBlur (radius = 1.5). Gaussian noise was generated as zero-mean additive noise with a standard deviation of 10 intensity levels in the 0–255-pixel domain (variance = 100). The haze/fog condition followed $I\prime = 0.70I + 0.30A$, where the atmospheric-light vector was fixed to $A = [220, 220, 220]$. For local occlusion, one dark rectangle covering 10–20% of the image area was generated with an aspect ratio sampled from 0.5 to 2.0 and a random spatial location; the occluded pixels were set to RGB [40, 40, 40]. The robustness evaluation used the same representative checkpoint

adopted throughout the deployment-validation workflow. This checkpoint was selected because its principal detection performance was closest to the mean of the repeated training runs. For stochastic perturbations, a fixed base random seed was used, and deterministic image- and perturbation-specific random states were derived to ensure reproducibility. All experiments used the same input size, confidence threshold, NMS threshold, and evaluation protocol as the previous experiments. Each perturbation type was evaluated using one predefined setting only; therefore, the reported results should be interpreted as a controlled single-setting sensitivity/stress test rather than a complete severity-dependent robustness benchmark. The results are presented in Table 15.

Table 15. Robustness test results of YOLO-FSD under visual perturbations.

Condition	mAP50/%	Precision/%	Recall/%	*F*1/%	ΔmAP50 Drop/pt
Clean	69.36	64.94	70.15	67.19	0.00
Brightness_down	69.25	67.23	69.02	67.76	0.11
Gaussian_blur	58.38	61.68	58.47	59.48	10.98
Gaussian_noise	53.24	61.15	51.96	56.14	16.12
Haze_fog	69.02	64.58	69.64	66.85	0.34
Occlusion	39.53	48.12	48.26	47.82	29.83

Note: Each perturbation type was evaluated using one predefined setting. The stochastic Gaussian-noise and local-occlusion conditions were generated using a fixed base random seed of 11 together with deterministic image- and perturbation-specific random-state generation. Therefore, the reported results constitute a controlled single-setting sensitivity/stress test rather than a multi-severity robustness benchmark.

Under the clean condition, YOLO-FSD achieved an mAP50 of 69.36%. Brightness decrease and haze/fog caused only limited mAP50 reductions of 0.11 pt and 0.34 pt, respectively, indicating that the model remained relatively stable under the tested low-illumination and mild haze/fog conditions.

Gaussian blur and Gaussian noise resulted in more substantial degradation. The mAP50 decreased to 58.38% under Gaussian blur and to 53.24% under Gaussian noise, corresponding to reductions of 10.98 pt and 16.12 pt, respectively. These results indicate that blur and noise impair the visual information required for detecting flame boundaries, smoke textures, and small targets.

Local occlusion had the largest effect. Under the occlusion condition, mAP50 decreased from 69.36% to 39.53%, with a reduction of 29.83 pt. Precision, Recall, and *F*1 also decreased to 48.12%, 48.26%, and 47.82%, respectively. This result indicates that partial obstruction of fire or smoke regions is a major limitation of the current model.

The above results characterize robustness under the defined controlled perturbation conditions on the test set. They do not constitute cross-dataset or independent field-video validation.

To further analyze typical failure modes, representative cases were selected from the detection results under the clean condition, as shown in Figure 8. The selected cases include a missed small flame, a missed thin-smoke region, a smoke-like false alarm, and a fire-like false alarm. Small flames and thin smoke may be difficult to detect because of their limited visible area, weak contrast, and indistinct boundaries. In addition, smoke-like visual patterns and bright nighttime backgrounds may produce false alarms because they share partial visual characteristics with the target classes. These examples illustrate the main failure modes observed under the evaluated clean test condition.

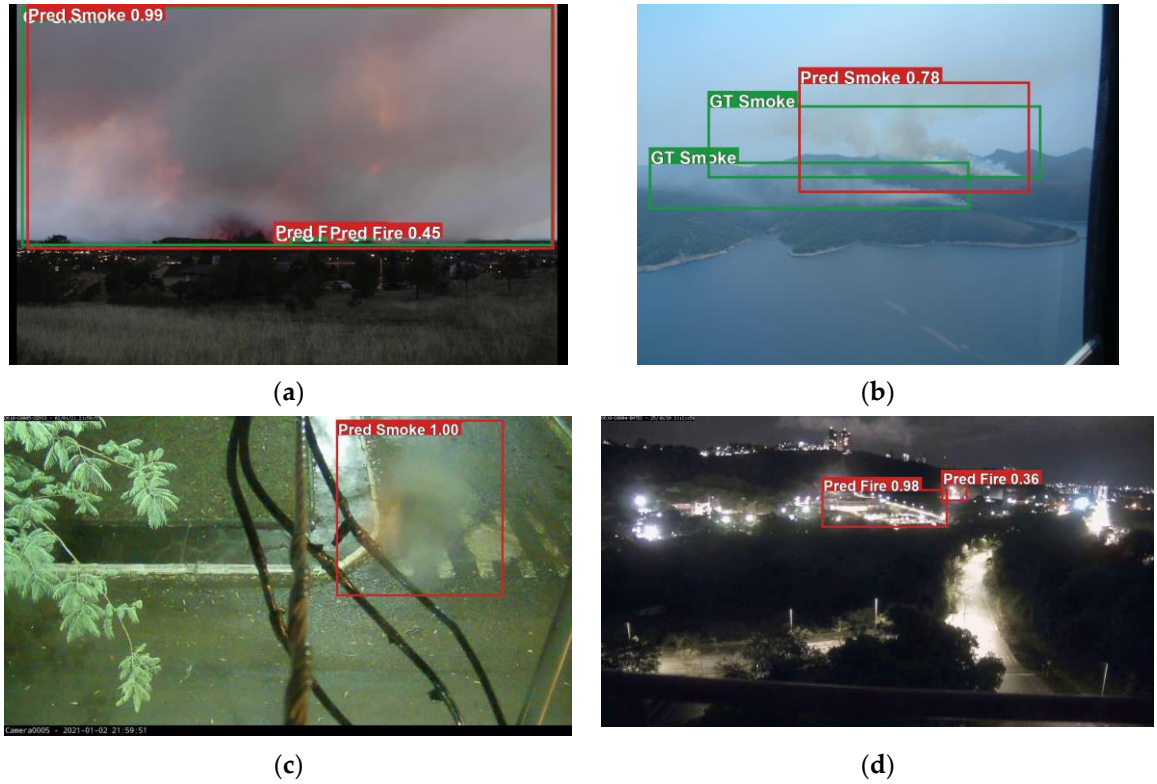


Figure 8. Representativefailure cases of YOLO-FSD on the D-Fire test set. (**a**) Small fire missed. (**b**) Thin smoke missed. (**c**) Smoke-like false alarm. (**d**) Fire-like false alarm. Green boxes indicate ground-truth annotations, and red boxes indicate model predictions.

3.3. Discussion and Limitations

The results of this study should be interpreted primarily from the perspective of deployment-oriented model redesign rather than software-side state-of-the-art accuracy or real-time FPGA throughput. YOLO-FSD improves the selected YOLOv4-tiny starting point while substantially reducing parameters and MACs, and the same-board ablation results show how the proposed structural changes affect both detection performance and FPGA forward-inference latency under a common custom HLS/IP workflow. At the same time, several recent nano-scale YOLO models achieve higher software-side mAP50 and, in some cases, lower parameter counts or MACs than YOLO-FSD. Because these models were not implemented under the same ZYNQ7020 custom accelerator, the architectural discussion concerning buffering, branch synchronization, intermediate-feature lifetime, and three-scale data movement should be interpreted as a deployment-oriented rationale rather than as experimentally demonstrated hardware superiority. Therefore, the main contribution of the present work is the joint consideration of lightweight structural redesign, fixed-point conversion, and same-board forward-inference validation under the constraints of a resource-limited FPGA platform.

Several limitations should be considered when interpreting the reported results. First, the merged dataset uses an image-level partition, and cross-split scene similarity may introduce optimism into absolute performance estimates, although the overlap-filtered sensitivity analysis retains the relative improvement of YOLO-FSD over the IRDS baseline. Second, the cumulative ablation protocol quantifies the incremental effect of the proposed design sequence but does not constitute a complete factorial analysis capable of isolating every component independently. The current cumulative ablation focuses on the progressive construction of YOLO-FSD and does not include a separate matched comparison among alternative inverted residual configurations. The LSE results are therefore inter-

preted within the context of its specific configuration, placement, and integration in the proposed architecture. Third, the software INT16 evaluation is a quantization-sensitivity analysis rather than a bit-accurate reproduction of the FPGA arithmetic, and the current implementation does not instrument layer-wise overflow or clipping statistics. Fourth, the reported FPGA latency corresponds to network forward inference and raw-output generation; bounding-box decoding and NMS are performed off-board on the PC, and therefore the reported 0.584 FPS should not be interpreted as fully on-board end-to-end throughput. Fifth, the reported power value is derived from the Vivado routed-design estimate rather than direct board-level power measurement. Sixth, recent nano-scale YOLO models were not implemented under the same ZYNQ7020 custom HLS/IP workflow, so no controlled same-board hardware-efficiency superiority over these models is claimed. Finally, the robustness evaluation uses controlled perturbations, and the 1080p experiment contains only eight annotated images; cross-dataset evaluation, independent field-video testing, broader perturbation severities, and more representative high-resolution deployment scenarios remain necessary for assessing generalization and practical robustness.

In addition, the current custom accelerator exhibits substantially lower achieved throughput than the implementation of Xu et al. [53] on the same ZYNQ-7020 device. Although the two studies do not provide fully identical timing and operation-counting definitions, the present implementation also uses lower channel-level parallelism and lower DSP utilization and incurs layer-wise PS/PL scheduling and external-memory data movement. Therefore, part of the observed performance gap represents a genuine efficiency limitation of the current hardware implementation rather than a measurement-convention difference alone.

4. Conclusions

Addressing the deployment-validation challenge of fire and smoke detection on resource-constrained edge devices, this study proposed YOLO-FSD, a lightweight detector evaluated on a Xilinx Zynq-7020 FPGA platform. On the test subset of the merged D-Fire and New Fire and Smoke corpus, YOLO-FSD achieves an mAP50 of 69.36% with 3.951 M parameters and 1.520 G MACs. Compared with YOLOv4-tiny, it improves mAP50 by 2.76 percentage points (pt) while reducing the parameter count and MACs by 32.76% and 55.53%, respectively. The results support the combined YOLO-FSD design as an accuracy-complexity trade-off: P_4-head lightweighting substantially reduces high-resolution computational cost, and shallow detail compensation helps recover detection performance in the final configuration. The final model improves Fire and Small-target AP50 relative to the IRDS-YOLOv4-tiny baseline, but the present ablation results do not imply that every individual component independently improves every class-wise or scale-wise metric.

Although the implementation completes network forward inference and raw-output generation on the resource-constrained FPGA platform, decoding and NMS remain off-board. Therefore, the reported 0.584 FPS represents FPGA forward-inference throughput rather than fully on-board end-to-end detection throughput. Future work should prioritize convolutional parallelism, on-chip buffer reuse, PL/DDR dataflow and data-movement scheduling, and on-board post-processing. Cross-dataset evaluation and independent field-video validation under different cameras, environmental conditions, and deployment scenes should also be conducted to further assess generalization. In addition, future work should include controlled same-board comparisons with recent nano-scale detectors, more comprehensive fixed-point error characterization, and direct board-level power measurement.

Author Contributions: Conceptualization, C.M. and H.X.; methodology, C.M.; software, C.M., P.C. and H.L.; validation, C.M. and H.H.; formal analysis, H.H.; investigation, P.C.; resources, P.C. and T.W.; writing—original draft preparation, P.C. and C.H.; writing—review and editing, P.C.; visualization, P.C.; supervision, T.W. and Z.S.; project administration, H.X. and C.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China, grant number 62273109; the Guangdong Provincial Science and Technology Achievements Transfer Program, grant number 2026A0202020030; and the Youth Program of the Guangdong Regional Joint Fund, grant number 2025A1515110259.

Data Availability Statement: The source datasets used in this study are publicly available. The D-Fire dataset is available at https://github.com/gaia-solutions-on-demand/DFireDataset, accessed on 13 July 2026, and the FireSmokeNEW dataset is available at https://universe.roboflow.com/catargiuconstantin2/firesmokenewdataset, accessed on 13 July 2026. The processed data generated during this study are available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Carta, F.; Zidda, C.; Putzu, M.; Loru, D.; Anedda, M.; Giusto, D. Advancements in forest fire prevention: A comprehensive survey. *Sensors* **2023**, *23*, 6635. [CrossRef] [PubMed]
2. Jemmali, M.; Melhim, L.K.B.; Boulila, W.; Amdouni, H.; Alharbi, M.T. Optimizing forest fire prevention: Intelligent scheduling algorithms for drone-based surveillance system. *Procedia Comput. Sci.* **2023**, *225*, 1562–1571. [CrossRef]
3. Morchid, A.; Oughannou, Z.; El Alami, R.; Qjidaa, H.; Jamil, M.O.; Khalid, H.M. Integrated Internet of Things (IoT) solutions for early fire detection in smart agriculture. *Results Eng.* **2024**, *24*, 103392. [CrossRef]
4. Alkhatib, A.A.A. A review on forest fire detection techniques. *Int. J. Distrib. Sens. Netw.* **2014**, *10*, 597368. [CrossRef]
5. Saponara, S.; Elhanashi, A.; Gagliardi, A. Real-time video fire/smoke detection based on CNN in antifire surveillance systems. *J. Real.-Time Image Process.* **2021**, *18*, 889–900. [CrossRef]
6. Jiang, P.; Ergu, D.; Liu, F.; Cai, Y.; Ma, B. A review of YOLO algorithm developments. *Procedia Comput. Sci.* **2022**, *199*, 1066–1073. [CrossRef]
7. Cao, X.; Wu, J.; Chen, J.; Li, Z. Complex scenes fire object detection based on feature fusion and channel attention. *Arab. J. Sci. Eng.* **2025**, *50*, 7587–7601. [CrossRef]
8. Shao, D.; Liu, Y.; Liu, G.; Wang, N.; Chen, P.; Yu, J.; Liang, G. YOLOv7scb: A small-target object detection method for fire smoke inspection. *Fire* **2025**, *8*, 62. [CrossRef]
9. Xu, Y.; Li, J.; Zhang, L.; Liu, H.; Zhang, F. CNTCB-YOLOv7: An effective forest fire detection model based on ConvNeXtV2 and CBAM. *Fire* **2024**, *7*, 54. [CrossRef]
10. Liu, W.; Shen, Z.; Xu, S. CF-YOLO: A capable forest fire identification algorithm founded on YOLOv7 improvement. *Signal Image Video Process.* **2024**, *18*, 6007–6017. [CrossRef]
11. Kim, S.-Y.; Muminov, A. Forest fire smoke detection based on deep learning approaches and unmanned aerial vehicle images. *Sensors* **2023**, *23*, 5702. [CrossRef] [PubMed]
12. Saydirasulovich, S.N.; Mukhiddinov, M.; Djuraev, O.; Abdusalomov, A.; Cho, Y.-I. An improved wildfire smoke detection based on YOLOv8 and UAV images. *Sensors* **2023**, *23*, 8374. [CrossRef] [PubMed]
13. Zhu, W.; Niu, S.; Yue, J.; Zhou, Y. Multiscale wildfire and smoke detection in complex drone forest environments based on YOLOv8. *Sci. Rep.* **2025**, *15*, 2399. [CrossRef] [PubMed]
14. Luan, T.; Zhou, S.; Liu, L.; Pan, W. Tiny-object detection based on optimized YOLO-CSQ for accurate drone detection in wildfire scenarios. *Drones* **2024**, *8*, 454. [CrossRef]
15. Li, C.; Du, Y.; Zhang, X.; Wu, P. YOLOGX: An improved forest fire detection algorithm based on YOLOv8. *Front. Environ. Sci.* **2025**, *12*, 1486212. [CrossRef]
16. Bochkovskiy, A.; Wang, C.-Y.; Liao, H.-Y.M. YOLOv4: Optimal speed and accuracy of object detection. *arXiv* **2020**, arXiv:2004.10934. [CrossRef]
17. Wang, C.-Y.; Bochkovskiy, A.; Liao, H.-Y.M. Scaled-YOLOv4: Scaling cross stage partial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 20–25 June 2021*; Institute of Electrical and Electronics Engineers: Piscataway, NJ, USA, 2021; pp. 13029–13038.

18. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.-C. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018*; IEEE Computer Society: Los Alamitos, CA, USA, 2018; pp. 4510–4520. [CrossRef]
19. Sheng, S.; Liang, Z.; Xu, W.; Wang, Y.; Su, J. FireYOLO-Lite: Lightweight forest fire detection network with wide-field multi-scale attention mechanism. *Forests* **2024**, *15*, 1244. [CrossRef]
20. Wu, C.; Ye, M.; Zhang, J.; Ma, Y. YOLO-LWNet: A lightweight road damage object detection network for mobile terminal devices. *Sensors* **2023**, *23*, 3268. [CrossRef] [PubMed]
21. Wang, J.; Qin, C.; Hou, B.; Yuan, Y.; Zhang, Y.; Feng, W. LCGSC-YOLO: A lightweight apple leaf diseases detection method based on LCNet and GSConv module under YOLO framework. *Front. Plant Sci.* **2024**, *15*, 1398277. [CrossRef] [PubMed]
22. Yu, H.; Luo, Q.; Peng, W.; Zheng, L.; Ju, J.; Zhuo, H. PKD-YOLOv8: A collaborative pruning and knowledge distillation framework for lightweight rapeseed pest detection. *Sensors* **2025**, *25*, 5004. [CrossRef] [PubMed]
23. Cao, X.; Hu, Y.; Zhang, H. LKD-YOLOv8: A lightweight knowledge distillation-based method for infrared object detection. *Sensors* **2025**, *25*, 4054. [CrossRef] [PubMed]
24. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018*; IEEE Computer Society: Los Alamitos, CA, USA, 2018; pp. 6848–6856. [CrossRef]
25. Han, K.; Wang, Y.; Tian, Q.; Guo, J.; Xu, C.; Xu, C. GhostNet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020*; IEEE Computer Society: Los Alamitos, CA, USA, 2020; pp. 1580–1589. [CrossRef]
26. Ma, S.; Li, W.; Wan, L.; Zhang, G. A lightweight fire detection algorithm based on the improved YOLOv8 model. *Appl. Sci.* **2024**, *14*, 6878. [CrossRef]
27. Lei, L.; Duan, R.; Yang, F.; Xu, L. Low complexity forest fire detection based on improved YOLOv8 network. *Forests* **2024**, *15*, 1652. [CrossRef]
28. Li, Z.; Kang, L.; Rao, H.; Nie, G.; Tan, Y.; Liu, M. Camellia oleifera fruit detection algorithm in natural environment based on lightweight convolutional neural network. *Appl. Sci.* **2023**, *13*, 10394. [CrossRef]
29. Kim, S.; Jang, B.; Lee, J.; Bae, H.; Jang, H.; Park, I.-C. A CNN inference accelerator on FPGA with compression and layer-chaining techniques for style transfer applications. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 1591–1604. [CrossRef]
30. Ding, C.; Wang, S.; Liu, N.; Xu, K.; Wang, Y.; Liang, Y. REQ-YOLO: A resource-aware, efficient quantization framework for object detection on FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, 24–26 February 2019*; Association for Computing Machinery: New York, NY, USA, 2019; pp. 33–42. [CrossRef]
31. Liu, B.; Zou, D.; Feng, L.; Feng, S.; Fu, P.; Li, J. An FPGA-based CNN accelerator integrating depthwise separable convolution. *Electronics* **2019**, *8*, 281. [CrossRef]
32. Guo, K.; Sui, L.; Qiu, J.; Yu, J.; Wang, J.; Yao, S.; Han, S.; Wang, Y.; Yang, H. Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 35–47. [CrossRef]
33. Pistellato, M.; Bergamasco, F.; Bigaglia, G.; Gasparetto, A.; Albarelli, A.; Boschetti, M.; Passerone, R. Quantization-aware NN layers with high-throughput FPGA implementation for edge AI. *Sensors* **2023**, *23*, 4667. [CrossRef] [PubMed]
34. Yan, Z.; Zhang, B.; Wang, D. An FPGA-based YOLOv5 accelerator for real-time industrial vision applications. *Micromachines* **2024**, *15*, 1164. [CrossRef] [PubMed]
35. Yang, X.; Zhuang, C.; Feng, W.; Yang, Z.; Wang, Q. FPGA implementation of a deep learning acceleration core architecture for image target detection. *Appl. Sci.* **2023**, *13*, 4144. [CrossRef]
36. Yang, G.; Lei, J.; Xie, W.; Fang, Z.; Li, Y.; Wang, J.; Zhang, X. Algorithm/hardware codesign for real-time on-satellite CNN-based ship detection in SAR imagery. *IEEE Trans. Geosci. Remote Sens.* **2022**, *60*, 5226018. [CrossRef]
37. Al Amin, R.; Hasan, M.; Wiese, V.; Obermaisser, R. FPGA-based real-time object detection and classification system using YOLO for edge computing. *IEEE Access* **2024**, *12*, 73268–73278. [CrossRef]
38. Jocher, G.; Chaurasia, A.; Qiu, J. Ultralytics YOLOv8, Version 8.0.0. 2023. Available online: https://github.com/ultralytics/ultralytics (accessed on 13 August 2026).
39. Wang, C.-Y.; Yeh, I.-H.; Liao, H.-Y.M. YOLOv9: Learning what you want to learn using programmable gradient information. In *Computer Vision—ECCV 2024*; Leonardis, A., Ricci, E., Roth, S., Russakovsky, O., Sattler, T., Varol, G., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2025; Volume 15089, pp. 1–21. [CrossRef]
40. Wang, A.; Chen, H.; Liu, L.; Chen, K.; Lin, Z.; Han, J.; Ding, G. YOLOv10: Real-time end-to-end object detection. *Adv. Neural Inf. Process. Syst.* **2024**, *37*, 107984–108011. [CrossRef]
41. Jocher, G.; Qiu, J. Ultralytics YOLO11, Version 11.0.0. 2024. Available online: https://github.com/ultralytics/ultralytics (accessed on 13 August 2026).
42. Tian, Y.; Ye, Q.; Doermann, D. YOLOv12: Attention-centric real-time object detectors. *arXiv* **2025**, arXiv:2502.12524. [CrossRef]

43. Li, C.; Jiang, H.; Li, Y.; Xie, H.; Wang, Y.; Fan, Y. Optimization and Deployment of Real-Time On-Orbit Intelligent Interpretation Algorithms for Spaceborne Remote Sensing. *Sensors* **2026**, *26*, 4377. [CrossRef] [PubMed]
44. de Venâncio, P.V.A.B.; Lisboa, A.C.; Barbosa, A.V. An automatic fire detection system based on deep convolutional neural networks for low-power, resource-constrained devices. *Neural Comput. Appl.* **2022**, *34*, 15349–15368. [CrossRef]
45. Catargiu, C.; Cleju, N.; Ciocoiu, I.B. A comparative performance evaluation of YOLO-type detectors on a new open fire and smoke dataset. *Sensors* **2024**, *24*, 5597. [CrossRef] [PubMed]
46. PyTorch. MobileNet V2—Torchvision Documentation. Available online: https://docs.pytorch.org/vision/main/models/generated/torchvision.models.mobilenet_v2.html (accessed on 13 August 2026).
47. Zhang, H.; Cisse, M.; Dauphin, Y.N.; Lopez-Paz, D. mixup: Beyond empirical risk minimization. In Proceedings of the International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, 30 April–3 May 2018.
48. ALINX. *ZYNQ FPGA Development Board AX7020 User Manual*; Rev. 1.0; ALINX Electronic Technology: Shanghai, China, 2019; Available online: https://www.alinx.com/public/upload/file/AX7Z020_UG.pdf (accessed on 27 August 2026).
49. Xilinx Inc. *Zynq-7000 SoC Data Sheet: Overview (DS190)*; Version 1.11.1; Xilinx, Inc.: San Jose, CA, USA, 2018.
50. AMD. Vitis AI User Guide (UG1414), Version 3.0. 2023. Available online: https://docs.amd.com/r/3.0-English/ug1414-vitis-ai (accessed on 13 August 2026).
51. Nagel, M.; Fournarakis, M.; Amjad, R.A.; Bondarenko, Y.; van Baalen, M.; Blankevoort, T. A white paper on neural network quantization. *arXiv* **2021**, arXiv:2106.08295. [CrossRef]
52. Jocher, G.; Chaurasia, A.; Stoken, A.; Borovec, J.; NanoCode012; Kwon, Y.; Michael, K.; TaoXie; Fang, J.; imyhxy; et al. *ultralytics/yolov5: v7.0—YOLOv5 SOTA Realtime Instance Segmentation*; Zenodo: Geneva, Switzerland, 2022. [CrossRef]
53. Xu, S.; Zhou, Y.; Huang, Y.; Han, T. YOLOv4-Tiny-Based Coal Gangue Image Recognition and FPGA Implementation. *Micromachines* **2022**, *13*, 1983. [CrossRef] [PubMed]
54. Montgomerie-Corcoran, A.; Toupas, P.; Yu, Z.; Bouganis, C.-S. SATAY: A streaming architecture toolflow for accelerating YOLO models on FPGA devices. *arXiv* **2023**, arXiv:2309.01587. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.