Academic Journal of Science and Technology
ISSN: 2771-3032 | Vol. 22, No. 1, 2026

A Third-Party Construction Detection Model Incorporating Residual Scaling and Channel-Adaptive Gating

Jing Li[1, a], Weixiang Tao[1, b], Xuanpeng Liu[1, c], Yanlei Ye[1, d], Ran Wang[2, e], Haiquan Jin[2, *]

[1]Zhengyuan Geographic Information Group Co., LTD, Beijing 101300, China
[2]Beijing Zhengyuan Digital Technology Information Co., LTD, Beijing 101300, China

Contributing authors: [a]1300206659@qq.com, [b]524957227@qq.com, [c]78484447@qq.com, [d]Reyanlei@163.com, [e]2653019117@qq.com

*Corresponding author(s). E-mail(s): tj13821918683@163.com

Abstract: Third-party construction is a leading cause of urban gas pipeline damage. Unregulated digging and drilling frequently rupture pipes, disrupt service, and create safety hazards. To address this, we propose TPC-DETR—an end-to-end detector built on the DETR framework—that identifies risky construction activities near pipelines and helps prevent human-caused damage. Three enhancements improve the baseline. In the backbone, we add a Residual Scaled Block (RSB) that stabilizes deep-network training and strengthens feature representation, with zero extra parameters or FLOPs. Between the backbone and neck, we insert a Channel-Adaptive Cascaded Gating Layer (CACG-Layer) that lowers computational cost while boosting speed and accuracy through fine-grained multi-branch feature modulation. In the feature extraction path, we adopt re-parameterized RefConv, whose kernels adjust dynamically at inference based on learned weights—improving adaptability and generalization at no additional cost. On our self-built dataset, TPC-DETR achieves 97% mAP@0.5 and 81.6% mAP@0.5:0.95. These results show that the model reliably detects a wide range of third-party construction activities in complex pipeline environments, and is practical for real-world deployment.

Keywords: Third-party construction detection; End-to-end object detection; Detection Transformer (DETR); Real-time monitoring.

1. Introduction

Natural gas has become a mainstream energy resource due to its clean combustion, low carbon emissions, and high energy utilization efficiency. It occupies a critical position in the global energy structure and facilitates the transition toward low-carbon energy systems. Pipeline transportation is still the most dependable method for large-scale and long-distance natural gas delivery. As core urban infrastructure, gas pipeline networks sustain stable energy supply and maintain normal socioeconomic operations.

Despite their importance, gas pipelines are susceptible to external damage. According to official accident data, unauthorized digging, drilling and excavation conducted by external contractors are the primary triggers of urban pipeline damage. One hit can cause a leak, a fire, or even an explosion. In the worst case, the whole pipeline system goes down. People are in danger immediately. And the costs keep adding up—money lost, services cut, and communities disrupted [3]. In many cities, pipeline records are incomplete or missing entirely. Surface markers fade or get buried over time. Regulations require a five-meter safety buffer around pipelines, with construction prohibited in that area. But workers lack accurate positioning tools and real-time alerts. They cannot tell where the buffer zone is. So they dig—and hit a pipeline. One strike can trigger a cascade of failures [4].

Warning piles and signs are widely used to protect urban gas pipelines. But these physical markers have clear downsides. They're spaced far apart, so workers often can't see them. Many get buried or damaged during excavation. That means they can't provide ongoing monitoring or real-time alerts around pipeline safety zones. These problems get worse in city renewal and road construction, where work is dense and site conditions change quickly. Relying on static markers alone just doesn't cut it anymore for modern gas pipeline safety.

To get past these practical limits, we move beyond passive markers and adopt active monitoring. We develop an intelligent detection algorithm that spots high-risk construction work near gas pipeline corridors. By keeping an eye on the five-meter safety buffer zone, the system sends out early warnings when dangerous operations are detected, reducing the chance of accidental pipeline damage. This gives us a practical, responsive way to protect gas pipelines—whether in day-to-day operations or in busy city areas with lots of construction going on [5].

For a long time, gas pipeline safety management has relied on manual on-site inspections and static warning markers [6]. As sensors and computing power get better, intelligent monitoring is getting more research attention. Compared with regular human patrols, AI-driven detection of unauthorized third-party construction responds faster, saves on labor costs, and lowers the risk of missing hidden hazards [7-8].

Deep learning has pushed object detection forward. For image classification, backbones like VGG [9], ResNet [10], and HSVLT [11] have kept getting better at accuracy. For detection, lightweight models like YOLO [12] and SSD [13] can run real-time inference right on embedded devices. Transformer-based architectures have also entered computer-vision research. As a pioneering fully end-to-end detector, DETR [14] removes anchor-box design as well as NMS post-processing steps from its workflow.

Even so, when deployed for construction-activity detection alongside gas pipelines, mainstream algorithms are plagued by two persistent drawbacks: unsatisfactory computational efficiency and poor detection performance for small-scale targets.

Numerous studies have carried out targeted optimizations for comparable construction-site scenarios. Hu et al. [15] integrated RepVGGBlock alongside large-kernel attention modules into YOLOX-S and reported performance gains on construction-site datasets. Meng et al. [16] introduced residual connections into YOLOv3; this modification stabilized model training and boosted detection accuracy for excavator targets. Hao et al. [17] adopted YOLOv5s to monitor field workers and mechanical equipment in real time, achieving a reasonable trade-off between model footprint and prediction precision. Wang et al. [18] combined CBAM attention and dynamic scaling modules in YOLOv5 and reached 86.9 % mAP for pipeline-related excavation detection. Xiang et al. [19] further modified YOLOv5 via feature pyramid networks and RepVGG blocks. Their work improved multi-scale feature fusion and obtained 87.32 % mAP when detecting construction machinery.

When it comes to Transformer-based detection frameworks, Deformable DETR [20] applies deformable attention to strengthen its capability for multi-scale target perception. Meanwhile, Swin Transformer [21] realizes decent accuracy-efficiency trade-offs by adopting hierarchical window-partitioned attention operations.

Existing research delivers meaningful references for analyzing images captured at construction sites. Nevertheless, very few approaches are specially developed for detection tasks around gas pipelines. Such practical scenarios impose multiple strict requirements simultaneously: rapid inference speed, accurate target positioning, and reliable recognition for objects with highly diverse sizes.

To fill this research gap, our detection model combines two sets of competitive advantages: the high-speed inference characteristic of YOLO-series networks, together with Transformers' powerful capacity for global feature modeling. We designed the whole system around the real-world challenges of third-party construction monitoring. The scenes are complex. The targets vary widely in size. And detection has to be stable and fast.

We call this framework TPC-DETR. Its core building block is the Residual Scaled Block (RSB). A plain residual connection just passes features along the shortcut without doing much. The RSB does something different—it adds a scaling mechanism that actively controls how much the features get updated. This turns the shortcut from a passive detour into an active adjuster. The result? Better training stability and stronger feature representation, with zero extra parameters or computation cost.

2. Network Architecture Design

We use RT-DETR [22] as our baseline. Compared to the original DETR, RT-DETR has several improvements that make it more efficient. It uses a hybrid encoder, an Attention-based Intra-scale Feature Interaction (AIFI) module, a CNN-based Cross-scale Feature Fusion (CCFF) unit, and a lightweight design overall. These changes make RT-DETR more practical for real-world tasks that need both accuracy and speed.

We made several targeted improvements to adapt this baseline for detecting third-party construction near gas pipelines. Figure 1 shows the overall network structure. It has three main parts: a backbone, an encoder, and a decoder.

The backbone pulls out multi-scale features from the input construction images. These features then go through the CACG-Layer, which makes them cleaner and more useful. In the fusion stage, we add a re-parameterized RefConv module. This combines shallow and deep features efficiently and cuts down the information loss that often happens in standard fusion. The decoder then takes the fused features and outputs the final detection results through the prediction head.

With these improvements across the backbone, fusion, and decoder, TPC-DETR learns better features and representations. It performs well on our third-party construction datasets. The model offers a reliable option for gas pipeline safety monitoring and risk prevention work.

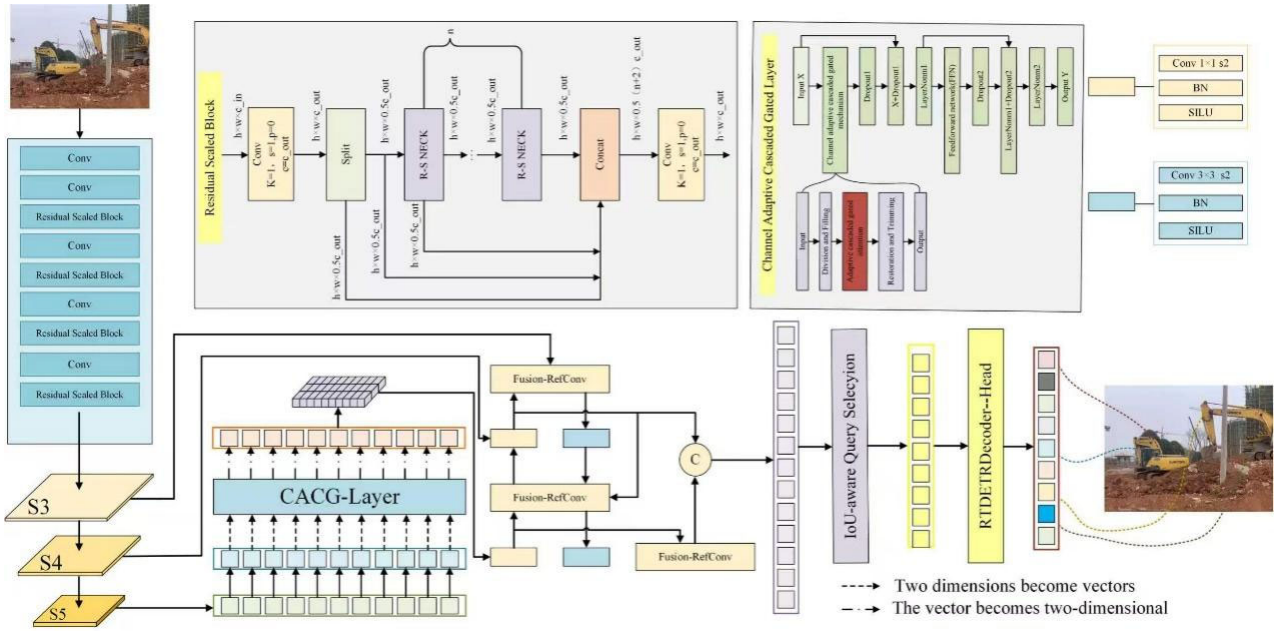


Figure 1. The overall structure of TPC-DETR.

2.1. Residual Scaled Block

Residual connections are everywhere in deep networks these days. Their main job is to stop gradients from vanishing by passing the input along. A standard residual block just adds the transformed features to the original input. This helps preserve information and keeps training stable. But it runs into problems when features go through many layers or get refined over and over. Three issues stand out.

First, the new features stay too similar to the input. So the network doesn't pick up much new information. Second, as signals move from layer to layer, the residual contributions keep building up. This makes feature values drift and distributions become uneven. Third, there's no control over how much new information each layer adds. So the update strength is basically unregulated. High-level detection tasks need more than just preserving features. Networks have to keep pulling out new patterns and gradually build up features that are good at telling things apart. Plain residual shortcuts can't do that. To fix these problems, we propose the Residual Scaled Block (RSB). Unlike a standard residual unit, RSB puts clear limits on how much each update can change things. This turns the passive shortcut into an active branch that picks out useful information and refines it. Figure 2 shows the structure.

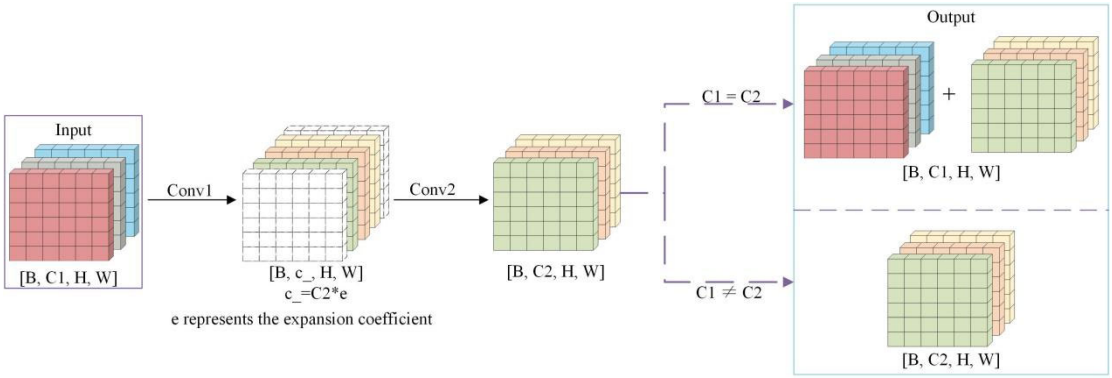


Figure 2. Residual Scaled Block

The input feature map is defined as:(1)

$$x \in \mathrm{R}^{C \times H \times W} \tag{1}$$

The standard residual output is:(2)

$$y = x + \mathrm{F}(x) \tag{2}$$

In our RSB, we add a scaling factor α and change the residual learning to:(3)

$$y = x + \alpha \cdot \mathrm{F}(x), \quad 0 < \alpha < 1 \tag{3}$$

Here, α decides how much the new features affect the final output. It basically sets the step size for each update. The function F(·) is just two convolution layers stacked together (see Eq. 4). Their kernel sizes are k_1 and k_2.

$$\mathrm{F}(x) = Conv_{k_2}(Conv_{k_1}(x)) \tag{4}$$

The RSB keeps the shortcut when input and output channels are the same. When they're different, we skip the shortcut and only use the transformation branch. This makes the module work with different network setups.

Standard residual blocks mainly pass information through. Our RSB does more—it gradually improves features step by step. When you stack several RSB modules, the feature flow follows Eq. (5), where yi is the output of the i-th block.

$$y_i = y_{i-1} + \alpha \cdot \mathrm{F}(y_{i-1}),\ i = 1,2,3,\ldots,N \tag{5}$$

Since α is less than 1, each RSB only makes a small adjustment. It keeps the original features intact while adding new useful information. This helps with both stability and feature variety. The gradient flow through the block comes from Eq. (3), giving us:(6)

$$\frac{\partial y}{\partial x} = \mathrm{I} + \alpha \cdot \frac{\partial \mathrm{F}(x)}{\partial x} \tag{6}$$

In Eq. (6), I is the identity mapping. With α less than 1, the RSB tones down the gradient boost from the transformation branch. This cuts down on unstable updates and makes training more stable as the network gets deeper. So the RSB gives smoother training, whether you stack it in deep networks or use it across multiple feature modeling stages.

2.2. Channel-Adaptive Cascaded Gating (CACG) Layer

Standard Transformer encoders are good at capturing global context. But their multi-head attention costs a lot of computation. The frequent reshaping and element-wise operations push up GFLOPs and slow things down. On top of that, different heads often produce similar feature maps, which wastes computation and makes the model less efficient.

To fix these problems, we propose the Channel-Adaptive Cascaded Gating Attention module (Figure 3). The basic idea is to split the feature sequence into smaller groups and give each group to its own attention head. This grouping approach cuts total computation and encourages each head to learn different patterns. For construction-site detection, this module gives better features and faster inference [23].

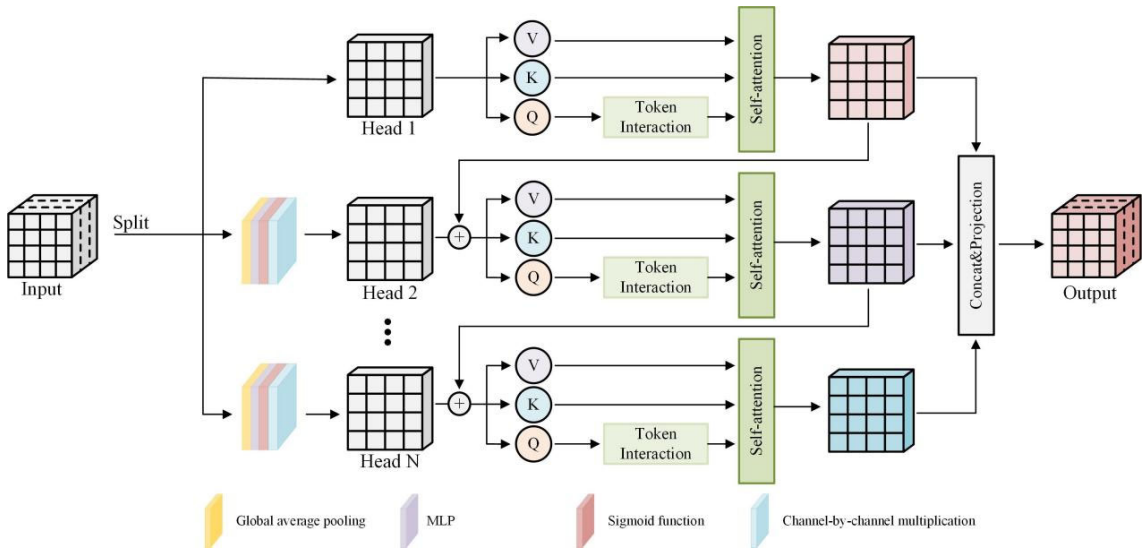


Figure 3. Overview of the Channel-Adaptive Cascaded Gating Attention mechanism

In TPC-DETR, the encoder takes multi-scale features from S3, S4, and S5. It replaces the standard MHA used in the original RT-DETR.

Standard MHA projects the input into Q, K, and V, then works out attention weights. But this design has a few issues. Multiple heads add a lot of computation—especially when you scale up training or run inference. Each head has its own projection matrices, which add parameters and make overfitting more likely. Plus, the attention weights are hard to read. The interactions between heads make it hard to see how the model reaches its decisions.

To get better features and push heads to work together, we replace MHA with our Channel-Adaptive Cascaded Gating mechanism. We split the input feature map (Eq. 7) into H groups along the channel dimension (Eq. 8). Each group (Eq. 9) then goes to its own head for processing.

$$X \epsilon \mathrm{R}^{B\times C\times H\times W} \tag{7}$$

$$X=[X_1,X_2,X_3,\cdots,X_H] \tag{8}$$

$$X_i \epsilon \mathrm{R}^{B\times \frac{C}{H}\times H\times W} \tag{9}$$

In standard multi-head attention, each head works on its own. Our cascaded gating approach connects them. The first head takes the input from Eq. (10). For each later head ii, its input comes from combining the previous head's output with its own grouped feature, as shown in Eq. (11).

$$F_1=X_1 \tag{10}$$

$$F_i=F_{i-1}+\sigma(g_{i-1})\odot X_i,\ \ i=2,3,\ \cdots,\ H \tag{11}$$

Here, $\odot$ means channel-wise multiplication. $\sigma(\cdot)$ is the Sigmoid function. GAP$(\cdot)$ is global average pooling. The MLP is just two light 1×1 convolution layers. Instead of using a fixed number, we get the gating vector g_{i-1} from the current grouped feature (Eq. 12).

$$g_{i-1}=MLP(GAP(X_i)) \tag{12}$$

The gating mechanism adjusts how much information each head takes in, based on what its assigned channels carry. This leads to better feature fusion. For each input Fi, we project it to Q, K, and V. On the query side, we add a depthwise separable convolution to pick up local patterns. Then we compute attention scores using a learnable position bias from 2D relative offsets. After all heads finish, we concatenate their outputs and project them to the final result. This design gives us better features and detection performance, with almost no extra cost.

2.3. RefConv for Parameter Refocusing

RT-DETR has a fusion block that combines features from different levels to handle complex scenes. It takes ideas from two sources: FPN's layered design and Transformers' big-picture view. This helps with two common problems in regular multi-scale fusion—losing details and wasting compute.

To make fusion better, we add RepC3. It uses structural re-parameterization. During training, it runs multi-branch convolutions to learn more diverse patterns. At inference time, these branches fold into one convolution. So we get better performance without slowing down.

Even with those benefits, RepConv's re-parameterization has its limits. It doesn't handle complex non-linear transformations well. We saw clear overfitting when testing it on our small third-party construction dataset.

To fix this, we switch to RefConv [24]. It has a refocusing mechanism that fine-tunes learned kernel weights. Everything can be merged at inference time, so there's no extra cost. It gets more out of the representation power already in the pre-trained parameters. By shifting where kernels pay attention, it pulls out better features and generalizes better when data is limited.

RefConv looks at convolution differently. It doesn't treat kernel weights as fixed templates. Instead, it sees them as learnable parts that a small sub-network can adjust. This sub-network makes changes based on the weights themselves, not on the input image. So the kernel adapts to its own structure. This turns a static kernel into a dynamic one that shifts its focus based on what it has learned. Eq. (13) shows this.

$$W_{final}=W_{base}+\mathrm{F}(W_{base}) \tag{13}$$

Here, W_{base} is the original kernel, F(·) is the refinement function, and W_{final} is the resulting dynamic kernel.

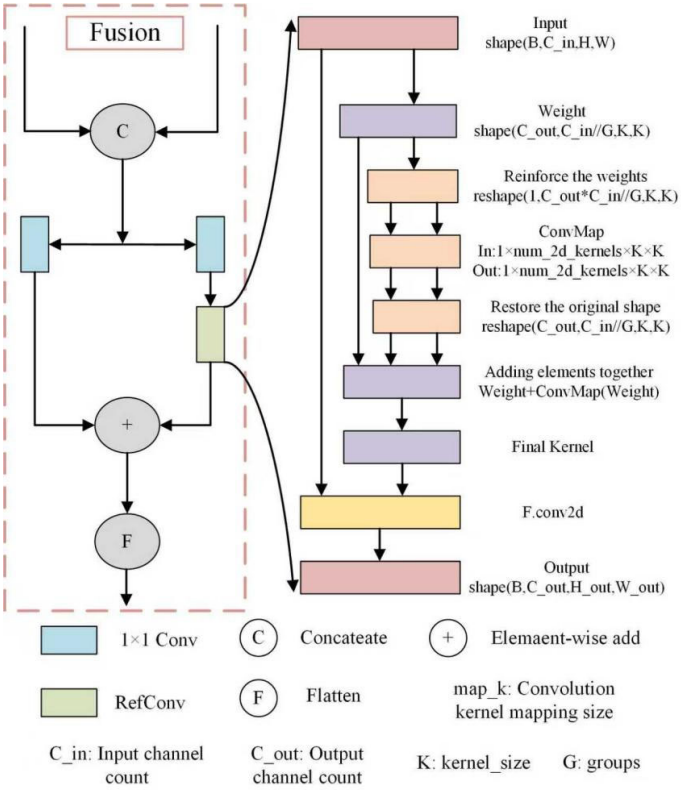


Figure 4. RefConv feature refinement module

Fig. 4 shows the RefConv module. It has two parts. One is the base convolution weights. The other is a lightweight refinement network called ConvMap. Most attention or gating modules look at the input features. ConvMap is different. It only looks at the kernel weights themselves. It never touches the input. This makes inference fast. We also keep ConvMap small. We use group convolutions to do that. So it barely adds any extra parameters. And the whole module is fully differentiable. That means we can train it end-to-end with the rest of the network.

3. Experiments

We run three sets of experiments. First, we compare TPC-DETR with several existing models to see how well it performs. Second, we do ablation studies to see what each new module brings. Third, we look at accuracy, speed, and robustness from different angles. We keep the same settings for training and testing across all models. That way we can compare the results fairly. All experiments run on a Windows 10 machine with an Intel i7-9700 CPU (3.00 GHz) and an NVIDIA Quadro P2200 GPU. We implement everything in PyTorch.

3.1. Dataset Construction

We built our own dataset for third-party construction detection. We gathered images from different sources. We annotated them in PASCAL VOC format, so each image has an XML file. Then we split the whole dataset into training, validation, and test sets. The split ratio is 8:1:1.

3.2. Ablation Study

Table 1 summarizes the ablation results obtained on our dataset. RT-DETR (ResNet-18) serves as the baseline, and three components including the Residual Scaled Block (A), CACG-Layer (B), and RefConv (C) are progressively introduced. Checkmarks denote the activation of corresponding modules. In the case of Module A, model parameters decrease by 22.6 % from 19.9 MB to 15.4 MB, alongside GFLOPs reduced to 53.6 and FPS increased from 52.5 to 61.4. Although mAP@0.5 increases by 0.8 % to 95.9 %, mAP@0.5:0.95 declines by 4.6 % to 76.0 %. These results indicate that Module A shrinks model volume and accelerates inference at the cost of partial localization precision. When Module B is enabled, mAP@0.5 and mAP@0.5:0.95 rise to 96.5 % (+1.4 %) and 82.3 % (+1.7 %) respectively. Its accuracy improvement comes with a drop in FPS to 43, while GFLOPs remain nearly unchanged, which suggests higher computational overhead. Module C brings parameters down to 15.5 MB (−22.1 %) and GFLOPs to 44.4, with FPS elevated to 58. Detection accuracy remains stable, reaching 95.8 % for mAP@0.5 and 80.4 % for mAP@0.5:0.95. Hence, Module C achieves efficiency gains without performance degradation. We further evaluate different module combinations. The A+B combination yields 15.2 MB parameters, 96.6 % mAP@0.5 (+1.5 %), 60 FPS and 80.1 % mAP@0.5:0.95. By integrating all three modules, the complete TPC-DETR achieves the most comprehensive performance: 97.0 % mAP@0.5 (+1.9 %), 81.6 % mAP@0.5:0.95 (+1.0 %), 69 FPS, 43.8 GFLOPs and 15.2 MB parameters.

From the above comparison, Modules A and C mainly contribute to efficiency optimization by reducing parameter size and computational consumption. Module B is responsible for accuracy enhancement on both evaluation metrics. The A-C combination already achieves competitive efficiency with 55 FPS and 43.7 GFLOPs. Incorporating

Module B causes a slight drop in inference speed yet brings obvious accuracy improvements. Finally, our full model achieves desirable trade-offs among model size, inference speed and detection accuracy.

Table 1. Ablation Experiments

Baseline	A	B	C	Parameters	$mAP_{0.5}$/%	$mAP_{0.5:\ 0.95}$/%	FPS	GFLOPs	Weights
√	×	×	×	19.9MB	95.1	80.6	52.5	57	40.4MB
√	√	×	×	15.4MB	95.9	76	61.4	53.6	31.4MB
√	×	√	×	19.7MB	96.5	82.3	43	57	40.1MB
√	×	×	√	15.5MB	95.8	80.4	58	44.4	34.5MB
√	×	√	√	22.03MB	96.5	80.3	38	76.5	42MB
√	√	×	√	15.4MB	95.7	80.7	55	43.7	35MB
√	√	√	×	15.2MB	96.6	80.1	60	52.6	31.2MB
√	√	√	√	15.2MB	97	81.6	69	43.8	34.3MB

3.3. Comparative Experiments

Table 2 compares our model against several state-of-the-art detectors. Here's how it does.

Against RT-DETR-R18, our model is lighter and faster, and it still beats the baseline on accuracy. Parameters drop from 19.9 MB to 15.2 MB—a 23.6% cut. Model weight goes from 40.4 MB down to 34.3 MB, a 15.1% drop. Accuracy? 97.0% mAP@0.5 and 81.6% mAP@0.5:0.95. That's 1.9 and 1.0 points higher than the baseline (95.1% and 80.6%). Speed jumps from 49 FPS to 69 FPS—a 40.8% gain. GFLOPs also drop, from 57 to 43.8 (−23.2%). So we get better accuracy, smaller size, and faster speed all at once. We also compare with YOLO v5 through v11 [25-32]. Our model beats all of them across every metric. The best YOLO we tested is v9-M, with 93.2% mAP@0.5 and 79.5% mAP@0.5:0.95—still below our 97.0% and 81.6%. YOLOv11-M, the newest one, runs a bit faster at 71.7 FPS, but its accuracy is lower: 92.8% and 79.2%. YOLOv5 is the cheapest to run, but its accuracy drops to 91.1% and 78.6%. The gap is clear. YOLO uses CNN backbones with fixed receptive fields and FPN-style fusion. These designs tend to lose global context or fine-grained details when merging features across scales. Our model avoids both issues—it has Transformers' global view, plus our module improvements.

Among DETR-based detectors, ours gives the best balance between accuracy and speed. The original DETR is slightly more accurate, but it runs at only 26 FPS and has a huge model size—not practical. RT-DETR-R50 hits 97.2% (just above our 97.0%), but it needs far more parameters and FLOPs. RT-DETR-R34 is close to us in accuracy, but it uses about twice the parameters and FLOPs, and runs at only 62% of our speed. Bottom line: TPC-DETR matches the top DETR models on accuracy, while running much faster and costing far less to compute.

Table 2. Comparative Experiments

Model	Parameters	GFLOPs	$mAP_{0.5}$/%	$mAP_{0.5:\ 0.95}$/%	FPS	Weights
YOLOv5	7.02MB	15.8	91.1	78.6	117.6	14.4MB
YOLOv7	37.2MB	105.2	93	79.4	40.4	74.8MB
YOLOv8-M	25.8MB	78.7	92.4	78.9	63.9	52.0MB
YOLOv9-M	32.6MB	130.8	93.2	79.5	36.5	66.3MB
YOLOv10-L	25.7MB	126.4	92.6	79	54.9	52.2MB
YOLOv11-M	20.3MB	67.7	92.8	79.2	71.7	40.5MB
DETR	41.3MB	225	94.5	81	26	485.4MB
RT-DETR-R18	19.9MB	57	95.1	80.6	49	40.4MB
RT-DETR-R34	31.1MB	88.8	96.3	81.3	42.9	63.0MB
RT-DETR-R50	41.9MB	129.6	97.2	81.9	32.8	86.1MB
TPC-DETR	15.2MB	43.8	97	81.6	69	34.3MB

We further evaluate generalization and robustness on three public benchmarks: MS COCO, VisDrone2019, and CrowdHuman. Tables 3–5 report the comparisons across models in terms of parameters, FLOPs, mAP, and model size. Across all three datasets, our method consistently outperforms the competitors—especially in cluttered scenes and across varied object distributions, where it shows clear gains in both robustness and adaptability.

Table 3. Comparison Experiments on the MIMICOCO Dataset

MIMICOCO	$mAP_{0.5}$/%	$mAP_{0.5:\ 0.95}$/%	Parameters	GFLOPs	Weights
YOLOv5	44.8	26.9	7.02MB	15.8	14.4MB
YOLOv7	53.2	34.6	37.2MB	105.2	74.8MB
YOLOv8-M	51.2	36.1	25.8MB	78.7	52.0MB
YOLOv9-M	50.7	36.3	32.6MB	130.8	66.3MB
YOLOv10-L	52.2	38.1	25.7MB	126.4	52.2MB
YOLOv11-M	53.6	40.0	20.3MB	67.7	40.5MB
DETR	55.3	39.6	41.3MB	225	485.4MB
TPC-DETR	56.1	39.9	15.2MB	43.8	34.3MB

Table 4. Comparison Experiments on the VisDrone 2019 Dataset

VisDrone2019	$mAP_{0.5}$/%	$mAP_{0.5:\ 0.95}$/%	Parameters	GFLOPs	Weights
YOLOv5	32.7	17.7	7.04MB	15.9	14.5MB
YOLOv7	35.5	18.5	37.3MB	105.2	74.8MB
YOLOv8-M	34.8	20.4	25.8MB	78.7	52.0MB
YOLOv9-M	33.3	19.5	32.6MB	130.8	66.3MB
YOLOv10-L	33.7	19.5	25.7MB	126.4	52.2MB
YOLOv11-M	34.6	20.5	20.3MB	67.7	40.5MB
DETR	35.0	20.8	41.3MB	225	485.4MB
TPC-DETR	35.5	21.1	15.2MB	43.8	34.3MB

Table 5. Comparison Experiments on the CrowdHuman Dataset

CrowdHuman	$mAP_{0.5}$/%	$mAP_{0.5:\ 0.95}$/%	Parameters	GFLOPs	Weights
YOLOv5	65.7	33.1	7.02MB	15.8	14.4MB
YOLOv7	72.1	35.8	37.2MB	105.2	74.8MB
YOLOv8-M	64.8	32.4	25.8MB	78.7	52.0MB
YOLOv9-M	65.2	33.2	32.6MB	130.8	66.3MB
YOLOv10-L	65.8	32.8	25.7MB	126.4	52.2MB
YOLOv11-M	66.9	34.1	20.3MB	67.7	40.5MB
DETR	55.9	30.1	41.3MB	225	485.4MB
TPC-DETR	73.0	36.2	15.2MB	43.8	34.3MB

3.4. Visualization Analysis

We evaluate our model qualitatively on the self-built dataset (Fig. 5), where we compare the baseline RT-DETR-R18 against our TPC-DETR across four representative examples.

Small-object detection. In the first row, the baseline picks up the main construction targets but misses a small construction area. Our model catches it, thanks to its improved sensitivity to small-scale objects—a clear advantage in dense scenes. Localization accuracy. The second row shows that our bounding boxes fit the construction regions much more tightly. The baseline tends to produce looser boxes; ours regresses boundaries more precisely. False-positive suppression. The third row illustrates a common pitfall. The baseline mistakes a normal yellow truck for construction equipment, likely because of the color similarity to nearby construction areas. Our model, with stronger feature discrimination, correctly classifies the truck as non-target—fewer false alarms.

Confidence calibration. In the fourth row, both detectors find the target, but ours assigns higher confidence scores. This suggests that TPC-DETR has learned more distinctive features and makes classification decisions with greater certainty.



Figure 5. Visualization effect

4. Conclusion

We propose TPC-DETR, a detection framework designed specifically for gas pipeline protection against third-party construction damage. The design targets three practical issues: degraded feature quality, inefficient multi-scale fusion, and the computational expense of Transformer encoders. Our solution combines three components—a Residual Scaled Block in the backbone, a Channel-Adaptive Cascaded Gating (CACG) layer for feature connection, and a RefConv module for feature extraction. Together, they improve feature representation while cutting computation and retaining the global context that Transformers are known for.

On our own third-party construction dataset, TPC-DETR achieves 97% mAP@0.5 and 81.6% mAP@0.5:0.95. Comparisons against several state-of-the-art detectors confirm that our approach is both effective and well-matched to the task.

References

[1] Hong, B., Li, X., Song, S., & Liu, Y. (2020). Optimal planning and modular infrastructure dynamic allocation for shale gas production. Applied Energy, 261, 114439. https://doi.org/10.1016/j.apenergy.2019.114439

[2] Koroteev, D., & Tekic, Z. (2021). Artificial intelligence in oil and gas upstream: Trends, challenges, and scenarios for the future. Energy and AI, 3, 100041. https://doi.org/10.1016/j.egyai.2020.100041

[3] Witek, M., & Uilhoorn, F. E. (2021). Influence of gas transmission network failure on security of supply. Journal of Natural Gas Science and Engineering, 90, 103877. https://doi.org/10.1016/j.jngse.2021.103877

[4] Rusin, A., Stolecka-Antczak, K., Kapusta, K., & Grzelak, M. (2021). Analysis of the effects of failure of a gas pipeline caused by a mechanical damage. Energies, 14(22), 7686. https://doi.org/10.3390/en14227686

[5] Hussain, R., Lee, D., Abbas, M. S., & Ali, S. (2026). Vision-language model-based intelligent assistant for onsite construction safety inspection. Automation in Construction, 182, 106728. https://doi.org/10.1016/j.autcon.2025.106728

[6] Kim, J., Gong, Y., Chi, S., & Park, J. (2026). Towards transparent object detection models for construction sites: explainable AI and error classification. Advanced Engineering Informatics, 71, 104245. https://doi.org/10.1016/j.aei.2025.104245

[7] Awuku, B., Huang, Y., & Yodo, N. (2023). Predicting natural gas pipeline failures caused by natural forces: an artificial intelligence classification approach. Applied Sciences, 13(7), 4322. https://doi.org/10.3390/app13074322

[8] Li, L., Cui, X., Wang, J., & Zhang, H. (2026). Automated Tracking of Worker and Heavy Equipment on Tunnel Construction Sites: Deep-Learning Framework. Journal of Construction Engineering and Management, 152(2), 04025237. https://doi.org/10.1061/JCEMD4.COENG-16807

[9] Vedaldi, A., & Zisserman, A. (2016). Vgg convolutional neural networks practical. University of Oxford.

[10] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778). https://doi.org/10.1109/CVPR.2016.90

[11] Ouyang, S., Wang, H., Niu, Z., & Li, Y. (2023). HSVLT: Hierarchical scale-aware vision-language transformer for multi-label image classification. In Proceedings of the 31st ACM International Conference on Multimedia (pp. 4768-4777). https://doi.org/10.1145/3581783.3612159

[12] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788). https://doi.org/10.48550/arXiv.1506.02640

[13] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), Computer Vision–ECCV 2016 (pp. 21-37). Springer International Publishing. https://doi.org/10.1007/978-3-319-46448-0_2

[14] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-end object detection with transformers. In A. Vedaldi, H. Bischof, T. Brox, & J.-M. Frahm (Eds.), Computer Vision – ECCV 2020 (pp. 213-229). Springer International Publishing. https://doi.org/10.1007/978-3-030-58452-8_13

[15] Hu, H., Guo, F., & Liu, Z. (2023). Target Detection in Construction Scenes Using an Improved YOLOX-S Model. Journal of Frontiers of Computer Science & Technology, 17(5). https://doi.org/10.3778/j.issn.1673-9418.2205012

[16] Meng, L., Peng, Z., Zhou, J., & Liu, T. (2020). Real-time detection of ground objects based on unmanned aerial vehicle remote sensing with deep learning: Application in excavator detection for pipeline safety. Remote Sensing, 12(1), 182. https://doi.org/10.3390/rs12010182

[17] Hao, F., Zhang, T., He, G., & Wang, Q. (2024). CaSnLi-YOLO: construction site multi-target detection method based on improved YOLOv5s. Measurement Science and Technology, 35(8), 085202. https://doi.org/10.1088/1361-6501/ad42c5

[18] Wang, W., He, W., Bai, M., & Liu, S. (2025). Improved detection and reidentification algorithm for natural gas pipeline excavator dynamic tracking based on deep learning. Measurement, 118567. https://doi.org/10.1016/j.measurement.2025.118567

[19] Xiang, Y., Zhao, J., Wu, W., & Li, B. (2022). Automatic object detection of construction workers and machinery based on improved YOLOv5. In Y. Wang (Ed.), International Conference on Green Building, Civil Engineering and Smart City (pp. 741-749). Springer Nature Singapore. https://doi.org/10.1007/978-981-19-5217-3_74

[20] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 10012-10022). https://doi.org/10.48550/arXiv.2103.14030

[21] Zhu, X., Su, W., Lu, L., Li, B., Wang, X., & Dai, J. (2020). Deformable detr: Deformable transformers for end-to-end object detection [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2010.04159

[22] Zhao, Y., Lv, W., Xu, S., & Li, J. (2024). Detrs beat yolos on real-time object detection. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 16965-16974). https://doi.org/10.48550/arXiv.2304.08069

[23] Liu, X., Peng, H., Zheng, N., & Yang, Y. (2023). Efficientvit: Memory efficient vision transformer with cascaded group attention. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 14420-14430). https://doi.org/10.48550/arXiv.2305.07027

[24] Cai, Z., Ding, X., Shen, Q., & Wang, Y. (2025). Refconv: Reparameterized refocusing convolution for powerful convnets. IEEE Transactions on Neural Networks and Learning Systems. https://doi.org/10.1109/tnnls.2025.3552654

[25] Jocher, G., Chaurasia, A., Stoken, A., & Qiu, J. (2022). ultralytics/yolov5: v7.0-yolov5 sota realtime instance segmentation [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3908559

[26] Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2023). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 7464-7475). https://doi.org/10.48550/arXiv.2207.02696

[27] Reis, D., Nguyen, H., & Choi, J. (2023). Real-Time Flying Object Detection with YOLOv8 [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2305.09972

[28] Lyu, C., Zhang, W., Huang, H., Zhou, Y., Wang, Y., Liu, Y., Zhang, S., & Chen, K. (2022). Rtmdet: An empirical study of designing real-time object detectors [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2212.07784

[29] Xu, X., Jiang, Y., Chen, W., Zhang, Y., & Sun, X. (2022). Damo-yolo: A report on real-time object detection design [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2211.15444

[30] Wang, C. Y., Yeh, I. H., & Mark Liao, H. Y. (2025). Yolov9: Learning what you want to learn using programmable gradient information. In A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, & G. Varol (Eds.), Computer Vision – ECCV 2024 (pp. 1-21). Springer, Cham. https://doi.org/10.1007/978-3-031-72751-1_1

[31] Wang, A., Chen, H., Liu, L., Ding, G., Han, J., & Lin, Z. (2024). Yolov10: Real-time end-to-end object detection. In Advances in Neural Information Processing Systems 37 (pp. 107984-108011). https://doi.org/10.52202/079017-3429

[32] Khanam, R., & Hussain, M. (2024). Yolov11: An overview of the key architectural enhancements [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2410.17725