**J. Meas. Sci. Instrum.**, 2026, 17(2): 278-296. http://jmsi.publish.founderss.cn
**DOI**: 10.62756/jmsi.1674-8042.2026024 jmsi@nuc.edu.cn

# Real-time lightweight self-supervised monocular depth estimation

YANG Tianxiang[1], MENG Lingjun[1*], JIN Hong[2], FENG Wenjie[1], LIU Xinhao[3]

*1. School of Instrument and Electronics, North University of China, Taiyuan 030051, China;*
*2. School of Electrical and Control Engineering, North University of China, Taiyuan 030051, China;*
*3. School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 201900, China*

***Corresponding author:** MENG Lingjun (menglingjun@nuc.edu.cn)

**Received:** February 23, 2026 **Revised:** April 16, 2026 **Accepted:** May 15, 2026

**Abstract:** Monocular depth estimation aims to predict depth information within a scene from a single RGB image, but many models remain computationally intensive for real-time inference on resource-constrained edge devices. This paper presents a lightweight self-supervised monocular depth estimation network that balances accuracy and efficiency through targeted encoder–decoder design. The encoder employed a synergistic modeling approach combining decomposable large-kernel convolutions and local depthwise convolutions to capture both long-range context and local details with low computational overhead. The decoder utilized cross-scale feature differences as guidance to dynamically fuse multi-scale features, enhancing detail recovery and geometric consistency under lightweight constraints. In addition, a temporal soft fusion reprojection loss was employed to better leverage the complementary information of forward and backward frames, improving the robustness of self-supervised training. The model contained 3.0 M parameters and required 3.5 GFLOPs of computation. On KITTI, it achieves Abs Rel=0.105 and $\delta_1$=0.892. On Make3D, it achieves Abs Rel=0.308 in a zero-shot setting. On a Rockchip RK3588S, a hybrid-quantized multi-thread implementation runs at 67 frames/s. The results demonstrated that the proposed method achieved a favorable accuracy–efficiency balance on edge devices, making it suitable for real-time monocular depth estimation tasks.

**Key words:** monocular depth estimation; deep learning; self-supervised learning; large-kernel attention; differential-driven dynamic fusion; lightweight network; RK3588S

## 0 Introduction

Monocular depth estimation has become a key component in autonomous driving, robot navigation[1], and 3D reconstruction. Supervised methods[2-5] rely on dense ground-truth depths acquired by sensors such as LiDAR. However, data collection is costly and scenario coverage is limited, which hinders large-scale training and cross-domain generalization. Therefore, recent research has gradually shifted towards self-supervised methods that do not require ground truth depth data.

Self-supervised monocular depth estimation is commonly formulated as a task of view synthesis. In this framework, a depth network predicts the depth or disparity of the target frame, while a pose network estimates the relative camera motion between adjacent frames. Through differentiable geometric projection, the source frame is warped to the target view, and losses are constructed using photometric consistency and geometric constraints to achieve end-to-end optimization. This paradigm has established a well-developed training and evaluation workflow, becoming one of the key research directions in monocular depth estimation.

Despite significant progress in self-supervised methods, deploying them on embedded platforms such as in-vehicle devices, drones, and robots still requires a better balance between accuracy and inference efficiency. On the one hand, under lightweight constraints, the encoder must ensure operator-friendliness and low computational overhead while possessing sufficient long-range context modeling capabilities to maintain global geometric consistency. On the other hand, the decoder needs to fully fuse multi-scale features at low decoding cost, enhancing the recovery quality of local details and geometric consistency to reconstruct high-resolution depth maps. To address these challenges, we introduced targeted designs at both the encoding and decoding stages. Specifically, the encoder incorporates the dual-branch large-kernel attention module to achieve low-overhead joint modeling of long-range context and local details, and the decoder follows an adjacent-scale sparse

© The Author(s) 2026. Published by Publishing Center of North University of China. The articles published in this open access journal are distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

fusion principle and further adopts the differential-driven dynamic fusion for adaptive cross-scale interaction. Furthermore, we introduced a temporal soft fusion reprojection loss to fully leverage complementary information between consecutive frames, achieving robust self-supervised training. The main contributions are summarized as follows.

1) We proposed a lightweight self-supervised monocular depth estimation framework targeting real-time deployment. With an operator-friendly design, the framework jointly optimized long-range and local context modeling in the encoder and cross-scale dynamic fusion in the decoder, yielding a favorable accuracy–efficiency trade-off. Furthermore, we introduced a temporal soft fusion reprojection loss to enhance the robustness of self-supervised training.

2) We designed a dual-branch large-kernel attention (DB-LKA) block tailored for monocular depth estimation. It combined a decomposed large-kernel attention branch for long-range dependencies and a depthwise convolution branch for local details, achieving efficient joint modeling of long-range context and local details under lightweight budgets.

3) We proposed a differential-driven dynamic fusion (DDF) module. Guided by differences in features across scales, the module performed adaptive filtering on shallow features and generated residual signals to mitigate semantic discrepancies across scales, achieving dynamic cross-scale feature fusion while keeping low decoding overhead.

4) We conducted comprehensive comparisons and ablation studies on KITTI and Make3D, and further evaluated deployment performance on a RK3588S embedded platform. Results showed that the proposed method achieved competitive accuracy with 3.0 M parameters and 3.5 GFLOPs. Multi-threaded inference using hybrid quantization reached 67 frame/s, demonstrating strong practicality for edge deployment.

## 1 Related work

### 1.1 Self-supervised monocular depth estimation

Early self-supervised approaches were built on stereo geometry constraints and trained with view reconstruction and photometric consistency losses[6]. Godard et al.[7] further introduced left–right consistency by predicting disparities for both views and enforcing cross-view reconstruction. Zhou et al.[8] extended the framework to monocular video sequences, jointly optimizing depth networks and pose networks using motion constraints between adjacent frames. To mitigate occlusions, static pixels, and dynamic objects, Godard et al.[9] proposed the minimum reprojection loss together with an auto-masking strategy, which stabilizes training and has become a widely adopted baseline.

To improve accuracy, researchers have explored geometric priors, temporal modeling, and feature interaction. Guizilini et al.[10] introduced a 3D packing prior to improve geometric consistency and detail recovery. Watson et al.[11] exploited multi-frame temporal aggregation to enhance robustness to occlusions. Zhou et al.[12] proposed recurrent multi-scale feature modulation to suppress error propagation and improve predictions for boundaries and small objects. Lü et al.[13] redesigned the skip connections and proposed the fSE fusion module to reduce the semantic gap across resolutions, thereby improving boundary and thin-structure reconstruction. To strengthen long-range dependency modeling, several works[14-15] incorporated transformers into depth estimation and achieved notable accuracy gains. However, these approaches often increase architectural complexity and computational cost.

Driven by real-time deployment requirements, lightweight self-supervised monocular depth estimation has received growing attention. Zhang et al.[16] and Zhao et al.[17] combined CNN with channel-wise self-attention mechanisms[18] to build lightweight networks that balanced speed and accuracy. Feng et al.[19] constrained both the encoder and decoder to construct an efficient framework that traded off between accuracy and latency. Recent studies have further advanced this direction. Zhang et al.[20] proposed Repmono, which improves inference efficiency through large kernel convolution and structural reparameterization while maintaining competitive depth estimation performance. Zhao et al.[21] introduced LDA-Mono, which enhances scene-structure perception and local detail aggregation through a lightweight dual-aggregation design. Zhang et al.[22] proposed LEDepth, a lightweight self-supervised monocular depth estimation network that combined CNN and transformer, further improving the accuracy–efficiency balance for real-time and resource-constrained deployment.

In summary, existing methods have made encouraging progress. However, achieving effective long-range context modeling at low cost and efficient cross-scale fusion in decoding remains challenging under stringent edge-resource budgets.

### 1.2 Long-range modeling mechanisms

In dense prediction tasks such as depth estimation, modeling long-range dependencies is crucial for predicting structure boundaries, distant objects, and textureless regions. Conventional convolutional networks typically

enlarge receptive fields by stacking small kernels or deepening networks, which increases computation and still exhibits strong local inductive biases. Transformer[23] can capture global relations, but the attention computation and memory overhead are substantial, hindering real-time on-device deployment. Compared to transformers, XCiT[18] shifts attention computation to the channel dimension, significantly reducing computational load. However, it remains a substantial burden for resource-constrained devices. To obtain a large effective receptive field at a lower cost, Guo et al.[24] proposed large kernel attention (LKA) in VAN by decomposing large-kernel spatial modeling into a combination of depthwise and dilated depthwise convolutions. Although LKA provides a favorable trade-off between efficiency and receptive field, its large-kernel nature may smooth local high-frequency details (e.g., edges), especially under lightweight constraints. Therefore, it is important to design a low-cost mechanism that coordinates long-range context aggregation and local detail preservation.

### 1.3 Cross-scale fusion strategies

In monocular depth estimation networks, the decoder progressively upsamples and fuses multi-scale encoder features to recover high-resolution geometric details. Prior studies have shown that effective cross-scale fusion not only improves boundary and structural reconstruction but also increases decoding efficiency. HR-Depth[13] employs dense skip connections and the fSE module to alleviate semantic discrepancies across scales. DNA-Depth[25] performs neighborhood feature aggregation to strengthen information interaction in the decoder. However, lightweight decoders[16-17,19,22] typically employ sparse fusion strategies to enhance decoding efficiency by reducing the number of fusion nodes. Moreover, they usually aggregate features at fusion nodes through static addition or concatenation operations, followed by further fusion through small-kernel convolutions. Under the constraint of limited feature interactions, this static fusion approach may result in insufficient geometric consistency and detail recovery capabilities. Therefore, designing more adaptive and efficient fusion mechanisms remains a key challenge in decoder design.

## 2 Method

### 2.1 Motivation

For real-time on-device applications, limited compute and memory necessitate lightweight models. However, depth prediction requires both global geometric consistency and fine-grained local details. Lightweight encoders generally encounter difficulties in capturing global constraints for distant structures and textureless regions, while lightweight decoders typically rely on static cross-scale fusion, which may result in detail loss or inconsistent depth across geometric regions.

To address these issues, we introduced DB-LKA in the encoder to achieve low-overhead collaborative modeling of long-range and local information, and employed DDF in the decoder for adaptive fusion. This design improved depth estimation quality while retaining real-time performance on edge devices.

### 2.2 Depth estimation network

#### 2.2.1 Overall architecture

The proposed depth estimation network adopts a standard encoder–decoder architecture (Fig.1).

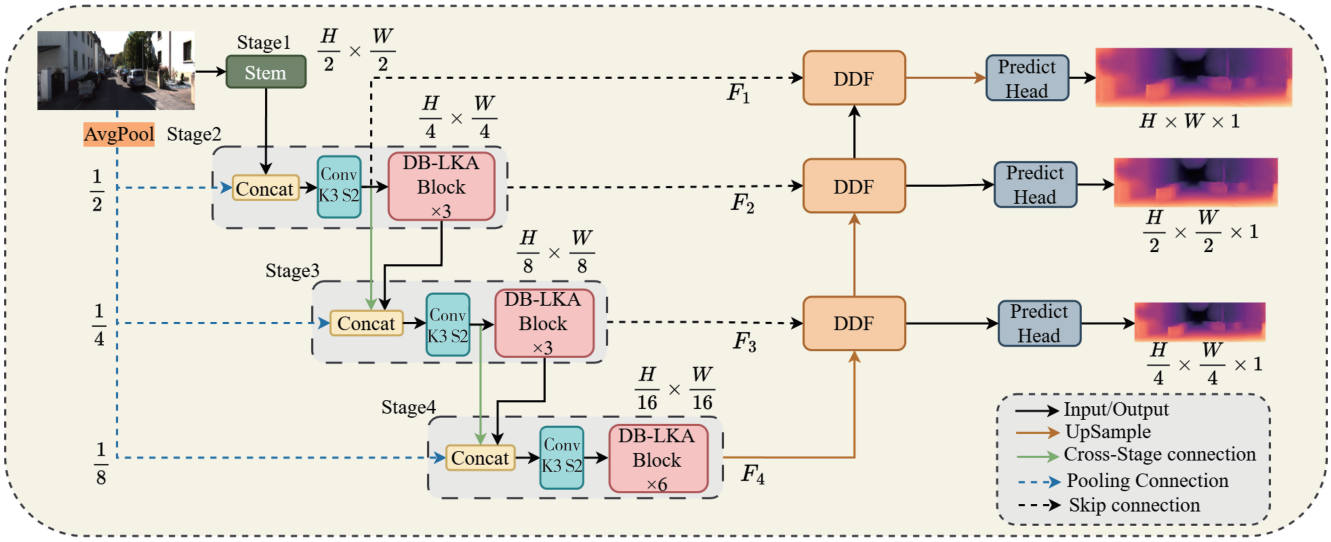



**Fig. 1 Depth estimation network architecture**

The encoder forms a four-stage feature pyramid (Table 1). In the first stage, the input image is processed by a convolutional stem for initial downsampling and feature extraction. The stem consists of a 3×3 convolution with a stride of 2 and two depthwise convolutions (DWConv). Inspired by prior work[16], we introduced an additional pooling branch during downsampling and concatenated it with the main-branch features along the channel dimension to alleviate the loss of spatial information. The concatenated features are then processed by subsequent stages that follow the pattern of downsampling with a 3×3 stride-2 convolution and feature extraction with DB-LKA blocks. Finally, the encoder outputs four multi-scale feature maps $\{F_1, F_2, F_3, F_4\}$.

**Table 1 Encoder configuration ($k$ is kernel size; $s$ is stride)**

| Layer | Input ($H\times W\times C$) | Output ($H\times W\times C$) |
|---|---|---|
| Stem | (192×640×3) | (96×320×32) |
| Avgpool | (192×640×3) | (96×320×3) |
| Avgpool | (96×320×3) | (48×160×3) |
| Avgpool | (48×160×3) | (24×80×3) |
| Conv $k$=3, $s$=2 | (96×320×35) | (48×160×48) |
| DB-LKA×3 | (48×160×48) | (48×160×48) |
| Conv $k$=3, $s$=2 | (48×160×99) | (24×80×96) |
| DB-LKA×3 | (24×80×96) | (24×80×96) |
| Conv $k$=3, $s$=2 | (24×80×195) | (12×40×160) |
| DB-LKA×6 | (12×40×160) | (12×40×160) |

Following the adjacent-scale sparse fusion principle, the decoder (Table 2) propagates information only between neighboring scales to reduce redundant computation.

The deepest feature $F_4$ is first upsampled by bilinear interpolation to the resolution of $F_3$ and fused with $F_3$ using the DDF module, generating the decoded feature $U_3$. Then, it is upsampled to the 1/4 scale and fused with $F_2$ to obtain the decoded feature $U_2$. To reduce computational overhead, $U_2$ is not fused with the encoded features at the 1/2 scale. Instead, it is fused with the shallow feature $F_1$ at the 1/4 scale and then upsampled to obtain the decoded feature $U_1$.

**Table 2 Decoder configuration**

| Layer | Input ($H\times W\times C$) | Output ($H\times W\times C$) |
|---|---|---|
| UpSample | (12×40×160) | (24×80×160) |
| DDF | (24×80×96), (24×80×160) | (24×80×96) |
| UpSample | (24×80×96) | (48×160×96) |
| DDF | (48×160×48), (48×160×96) | (48×160×48) |
| DDF | (48×160×32), (48×160×48) | (48×160×32) |
| UpSample | (48×160×32) | (96×320×32) |
| Predict head | (24×80×96) | (48×160×1) |
| Predict head | (48×160×48) | (96×320×1) |
| Predict head | (96×320×32) | (192×640×1) |

The process can be written as

$$U_3 = \mathrm{DDF}(F_3, \mathrm{UpSample}(F_4)), \tag{1}$$

$$U_2 = \mathrm{DDF}(F_2, \mathrm{UpSample}(U_3)), \tag{2}$$

$$U_1 = \mathrm{UpSample}(\mathrm{DDF}(F_1, U_2)). \tag{3}$$

Finally, the decoder outputs three decoded features at different scales $\{U_1, U_2, U_3\}$. Each is fed into a prediction head (a 3×3 convolution followed by bilinear upsampling and a sigmoid activation) to generate multi-scale single-channel disparity maps.

### 2.2.2 Dual-branch large-kernel attention

In monocular depth estimation, reliable depth inference requires both high-frequency local cues (e.g., object boundaries and texture variations) to recover sharp structural edges and low-frequency global geometry (e.g., scene layout and perspective) to maintain spatial consistency. To jointly model local and long-range spatial relations under lightweight constraints, we design a dual-branch large-kernel attention (DB-LKA) block based on LKA[24]. As shown in Fig.2, the input feature $F_{in}$ is first normalized by a BatchNorm layer for stable training and then fed into a long–short range interaction (LSRI) unit.

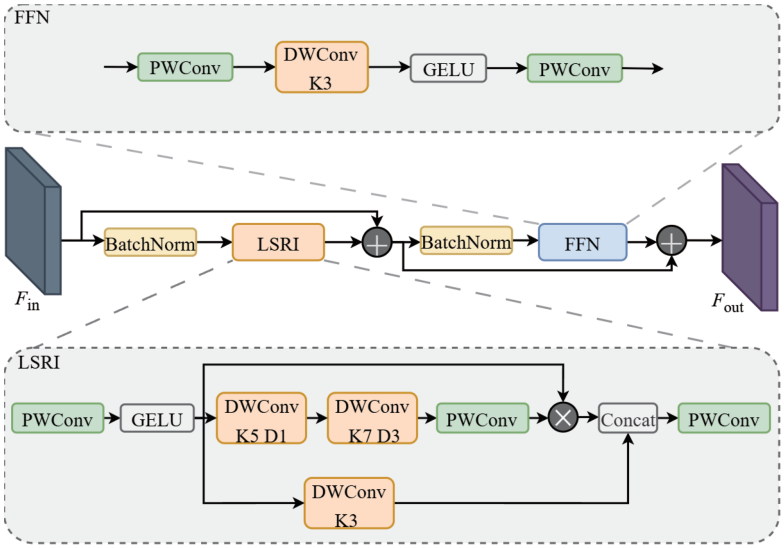



**Fig. 2 Dual-branch large-kernel attention block**

Next, the LSRI output is added to $F_{in}$ through a residual connection to generate an intermediate feature. After another BatchNorm layer, the feature passes through a feedforward network (FFN) unit, and the final output feature $F_{out}$ is obtained through a further residual addition.

The standard LKA extracts local information with a 5×5 depthwise convolution and expands the receptive field through a 7×7 depthwise convolution with a dilation rate of 3, thereby enabling long-range spatial modeling. However, this single pathway may result in local information being overly smoothed by large convolution kernels. Therefore, we explicitly introduce a lightweight parallel local branch to complement the LKA's long-range branch, allowing the network to integrate both local details and long-range context. Specifically, the unit adopts a dual-branch design. In the long-range branch, a 23×23 convolution is factorized into a 5×5 depthwise convolution, a 7×7 dilated depthwise convolution with dilation rate 3, and a pointwise convolution, achieving an effective large receptive field with low overhead. In the local branch, a 3×3 depthwise convolution is used to capture fine-grained details. The computation is given by

$$\boldsymbol{X}=\psi(\mathrm{PWConv}(\boldsymbol{X}_{in})), \tag{4}$$

$$\boldsymbol{X}_{Attn}=\mathrm{PWConv}(\mathrm{DWConv}_{k=7}^{d=3}(\mathrm{DWConv}_{k=5}^{d=1}(\boldsymbol{X}))), \tag{5}$$

$$\boldsymbol{X}_{lka}=\boldsymbol{X}_{Attn}\odot\boldsymbol{X}, \tag{6}$$

$$\boldsymbol{X}_{local}=\mathrm{DWConv}_{k=3}^{d=1}(\boldsymbol{X}), \tag{7}$$

where $X_{in}$ denotes the input of the unit, $\psi(\cdot)$ denotes the GELU activation, $\odot$ is the element-wise multiplication, $k$ is the kernel size, and $d$ is the dilation rate. Finally, the two branch features are concatenated and fused through a pointwise convolution.

$$\boldsymbol{X}_{out}=\mathrm{PWConv}(\mathrm{Concat}(\boldsymbol{X}_{lka},\boldsymbol{X}_{local})), \tag{8}$$

where $\boldsymbol{X}_{out}$ denotes the unit output, and Concat$(\cdot)$ denotes channel-wise concatenation.

FFN is employed to further enhance the feature extraction capabilities of the network, thereby enabling the capture of richer feature information. Firstly, a pointwise convolution expands the channel dimension to $E$ times the original channel size ($E$=4 in this work). Subsequently, 3×3 depthwise convolutions perform local spatial modeling, incorporating the GELU activation function to boost nonlinear expressiveness. Finally, another pointwise convolution projects the channels back to the original dimension. The computation can be written as

$$\boldsymbol{Y}_{out}=\mathrm{PWConv}(\psi(\mathrm{DWConv}_{k=3}^{d=1}(\mathrm{PWConv}(\boldsymbol{Y}_{in})))), \tag{9}$$

where $\boldsymbol{Y}_{in}$ and $\boldsymbol{Y}_{out}$ are the input and output of the FFN, respectively.

### 2.2.3 Differential-driven dynamic fusion

In the monocular depth estimation task, the decoder is required to effectively fuse adjacent-scale encoder features to combine shallow local details with deep global semantics. Many lightweight decoders perform cross-scale interaction through direct addition or channel concatenation and then apply small-kernel convolutions to fuse features. Such static fusion may cause detail loss or introduce geometric inconsistencies. To address this issue, we propose a differential-driven dynamic fusion (DDF) module (Fig.3), which uses cross-scale feature differences as a conditional signal to achieve adaptive fusion between cross-scale features.

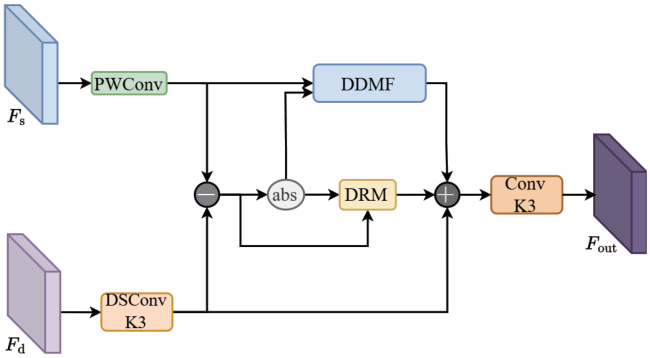



**Fig. 3 Differential-driven dynamic fusion module**

Given the resolution-aligned shallow feature $F_s$ and deep feature $F_d$, we first refine $F_d$ by using 3×3 depthwise separable convolution (DSConv) [26] to mitigate blurring effects potentially caused by upsampling. Then, pointwise convolution is applied to $F_s$ to align channel dimensions, shown as

$$F_d'=\mathrm{DSConv}_{3\times3}(F_d), \tag{10}$$

$$F_s'=\mathrm{PWConv}(F_s). \tag{11}$$

Next, a difference is computed, and the magnitude of this difference is then utilized as a cue of cross-scale inconsistency.

$$F_\Delta=F_s'-F_d', \tag{12}$$

$$F_{|\Delta|}=|F_\Delta|, \tag{13}$$

where $|\cdot|$ denotes element-wise absolute value. Through the differential mechanism, the network can explicitly capture semantic differences between cross-scale features within the same region, thereby providing reliable contextual information for subsequent dynamic fusion.

Based on different cues, we design two components: differential-driven dynamic mix filtering (DDMF) and differential-driven residual modulation (DRM). They are employed to regulate the injection intensity of shallow features and difference features, respectively.

The shallow feature $F_s$ typically contains rich high-

frequency details such as edges and textures. However, our objective is to preserve smoothness in flat regions, enhance edge clarity in boundary areas, and mitigate artifacts in textured regions. To enhance the adaptive adjustment capability of the fusion process, we introduce $N_g$ groups of $3\times3$ depthwise convolution filters ($N_g=3$ in this work) and predict pixel-wise mixing weights from the difference magnitude, enabling appropriate filters to dominate processing in different regions. DDMF is shown as Fig.4.

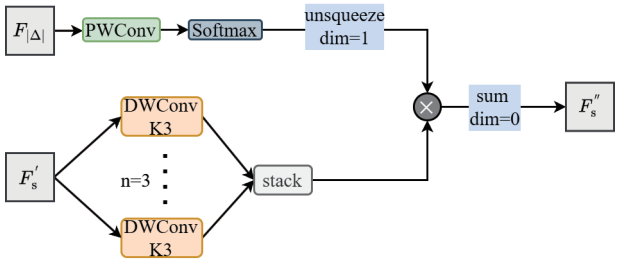



**Fig. 4 Differential-driven dynamic mix filtering**

In the initial stage of the process, parallel $N_g$ groups of depthwise convolutions are implemented to filter shallow features.

$$F_s^n = \mathrm{DWConv}_{3\times3}^{(n)}(F_s') \quad n=1,\cdots,N_g. \tag{14}$$

Next, the mixing weights are predicted from the difference magnitude.

$$\boldsymbol{w} = \mathrm{Softmax}_{N_g}(\mathrm{PWConv}(F_{|\Delta|})) \in \mathbf{R}^{N_g\times H\times W}. \tag{15}$$

Then, the filtered outputs are fused by the mixing weights.

$$F_s'' = \sum_{n=1}^{N_g} \boldsymbol{w}_n \odot F_s^n, \tag{16}$$

where $\boldsymbol{w}_n$ denotes the weight map for the $n$th group, $F_s^n$ denotes the shallow feature after dynamic mix filtering.

To further reduce semantic discrepancies between shallow and deep features, we design a residual modulation mechanism. DRM is shown in Fig.5.

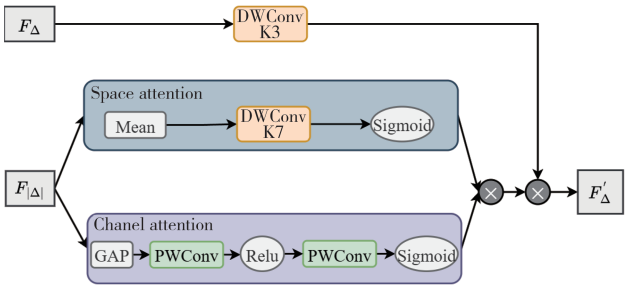



**Fig. 5 Differential-driven residual modulation**

Specifically, DRM extracts a spatial gate weight $\boldsymbol{g}_s$ and a channel gate weight $\boldsymbol{g}_c$ from the differential magnitude and combines them into a mixed gate weight $\boldsymbol{g}$. Meanwhile, a depthwise convolution encodes the raw difference feature map into a structured residual signal, enabling the model to capture local spatial context around inconsistent regions. Subsequently, the residual signal is reweighted using the mixed gate weight to adjust the injection strength at each location. The computation is

$$\boldsymbol{g}_s = \sigma(\mathrm{DWConv}_{7\times7}(\mathrm{Mean}_c(F_{|\Delta|}))), \tag{17}$$

$$\boldsymbol{g}_c = \sigma(\mathrm{PWConv}(\varphi(\mathrm{PWConv}(\mathrm{GAP}(F_{|\Delta|}))))), \tag{18}$$

$$\boldsymbol{g} = \boldsymbol{g}_s \cdot \boldsymbol{g}_c, \tag{19}$$

$$F_\Delta' = \mathrm{DWConv}_{3\times3}(F_\Delta) \odot \boldsymbol{g}, \tag{20}$$

where $\mathrm{Mean}_c(\cdot)$ denotes channel-wise mean pooling, $\mathrm{GAP}(\cdot)$ denotes global average pooling, $\sigma(\cdot)$ is the sigmoid activation, $\varphi(\cdot)$ is the ReLU activation. $\boldsymbol{g}_s \in \mathbf{R}^{1\times H\times W}$, $\boldsymbol{g}_c \in \mathbf{R}^{C\times1\times1}$, and their multiplication yield the broadcast mixed gate $\boldsymbol{g} \in \mathbf{R}^{C\times H\times W}$. $F_\Delta'$ is the reweighted residual signal.

Finally, we fuse the deep feature $F_d'$, the filtered shallow feature $F_s''$, and the reweighted residual signal $F_\Delta'$ by element-wise addition, followed by a $3\times3$ convolution for feature refinement, generating the fused output feature $F_{out}$.

$$F_{out} = \mathrm{Conv}_{3\times3}(F_d' + F_s'' + \theta \cdot F_\Delta'), \tag{21}$$

where $\theta$ is a learnable scalar that controls the strength of residual injection. It is initialized to $1\times10^{-6}$ for training stability.

## 2.3 Self-supervised learning strategy

### 2.3.1 Self-supervised training framework

We follow the self-supervised training pipeline of MonoDepth2[9] and employ a pose network with ResNet18[27] as the encoder and four convolutional layers as the decoder. We model monocular depth estimation as an image reconstruction task (Fig.6). During training, the depth network predicts the disparity (denoted as disp) of the target frame $\boldsymbol{I}_t$, while the pose network estimates the 6-DoF relative pose matrix $\boldsymbol{P}$ between the target frame $\boldsymbol{I}_t$ and each source frame $\boldsymbol{I}_s$ ($s\in\{t-1, t+1\}$). The disp is converted into depth $\boldsymbol{D}_t$ by

$$\boldsymbol{d}_t = d_{\min} + (d_{\max} - d_{\min}) \cdot \mathrm{disp}, \tag{22}$$

$$\boldsymbol{D}_t = \frac{1}{\boldsymbol{d}_t}, \tag{23}$$

where $d_{\max}=1/D_{\min}$ and $d_{\min}=1/D_{\max}$. We set $D_{\min}=0.1$ and $D_{\max}=100$ in this work. Given $\boldsymbol{D}_t$, $\boldsymbol{P}$, and the camera intrinsic matrix $\boldsymbol{K}$, the source frame is warped to the source frame through differentiable geometric projection to synthesize an image. The view synthesis can be expressed as

$$\hat{\boldsymbol{I}}_{t\leftarrow s} = \pi(\boldsymbol{I}_s, \boldsymbol{P}, \boldsymbol{D}_t, \boldsymbol{K}), \tag{24}$$

where $\pi(\cdot)$ denotes the projection function that maps 3D points to pixel coordinates in the source frame. The training objective is to jointly minimize across multiple scales, the reprojection loss between $\boldsymbol{I}_t$ and the synthesized image $\hat{\boldsymbol{I}}_{t\leftarrow s}$, together with an edge-aware smoothness loss on the predicted depth $\boldsymbol{D}_t$.

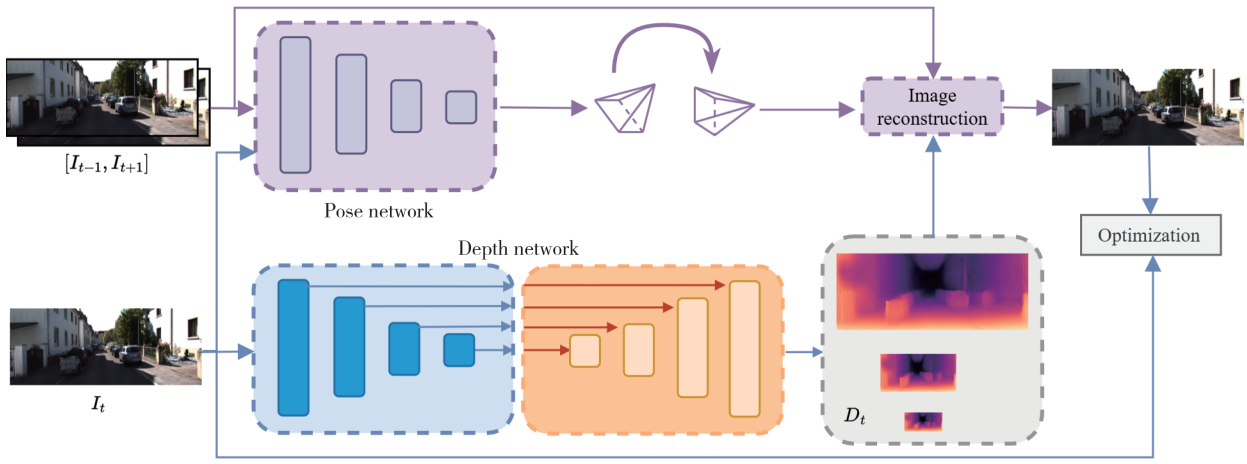



**Fig. 6 Self-supervised training framework**

2.3.2 Loss functions

1) Temporal soft fusion reprojection loss

We define the photometric reprojection loss as a weighted combination of the structural similarity (SSIM)[28] loss $L_{\text{SSIM}}$ and the $L_1$ loss (absolute error).

$$\text{pe}(\hat{\boldsymbol{I}}_{t\leftarrow s},\boldsymbol{I}_t)=\frac{\alpha}{2}L_{\text{SSIM}}(\hat{\boldsymbol{I}}_{t\leftarrow s},\boldsymbol{I}_t)+(1-\alpha)L_1(\hat{\boldsymbol{I}}_{t\leftarrow s},\boldsymbol{I}_t), \tag{25}$$

$$L_{\text{SSIM}}(\hat{\boldsymbol{I}}_{t\leftarrow s},\boldsymbol{I}_t)=1-\text{SSIM}(\hat{\boldsymbol{I}}_{t\leftarrow s},\boldsymbol{I}_t), \tag{26}$$

$$L_1(\hat{\boldsymbol{I}}_{t\leftarrow s},\boldsymbol{I}_t)=\left\|\hat{\boldsymbol{I}}_{t\leftarrow s}-\boldsymbol{I}_t\right\|_1, \tag{27}$$

where $\alpha$ is the weighting coefficient and is set to 0.85.

During self-supervised training, pixels from different positions in consecutive frames often exhibit complementary characteristics after reprojection to the target viewpoint. MonoDepth2[9] employs a minimum reprojection loss to select source frame pixels with minimal error for supervision, thereby enhancing occlusion robustness. However, this strategy constitutes a hard selection, where each pixel receives supervision signals from only a single source frame. It does not explicitly fuse complementary information from both frames, resulting in gradients being propagated back only to the selected branch. To leverage the complementary information of forward and backward frames and improve robustness, we propose a temporal soft fusion reprojection loss (TSFR loss). Specifically, pixel-wise fusion weights are computed according to the reprojection error of each source frame.

$$w_s=\frac{\exp(-\text{pe}(\hat{\boldsymbol{I}}_{t\leftarrow s},\boldsymbol{I}_t)/\tau)}{\sum\limits_{s'\in\{t-1,t+1\}}\exp(-\text{pe}(\hat{\boldsymbol{I}}_{t\leftarrow s'},\boldsymbol{I}_t)/\tau)}, \tag{28}$$

where $w_s$ denotes the pixel-wise fusion weight for source frame $s$, and $\tau$ is the temperature parameter set to 0.1. With $w_s$, the fused reconstruction is given by

$$\hat{\boldsymbol{I}}_t^{\text{fuse}}=\sum_{s\in\{t-1,t+1\}}w_s\hat{\boldsymbol{I}}_{t\leftarrow s}. \tag{29}$$

As $\tau\to 0$, $w_s$ converges to a one-hot distribution, $\hat{\boldsymbol{I}}_t^{fuse}$ approximating a minimum error reconstruction. At this point, the strategy behaves similarly to the minimum reprojection loss.

To further suppress corrupted supervision signals caused by dynamic objects and degenerate cases such as a static camera, we adopt the auto-masking[9] strategy.

$$e_t(x)=\min_{s\in\{t-1,t+1\}}\text{pe}(\boldsymbol{I}_s(x),\boldsymbol{I}_t(x)), \tag{30}$$

$$L_{\text{pe}}=\text{pe}(\hat{\boldsymbol{I}}_t^{fuse},\boldsymbol{I}_t), \tag{31}$$

$$L_{\text{r}}(x)=\min\left\{L_{\text{pe}}(x),e_t(x)\right\}, \tag{32}$$

where $e_t$ denotes the identity error, $L_{\text{pe}}$ is the photometric reprojection loss between the fused reconstruction and the target image, and $L_{\text{r}}$ is the final reprojection loss.

2) Edge-aware smoothness loss

To suppress noise in the depth map while preserving edges, we use the edge-aware smoothness loss, i.e.,

$$L_{\text{smooth}}=\left|\partial x\boldsymbol{d}_t^*\right|e^{-|\partial x\boldsymbol{I}_t|}+\left|\partial y\boldsymbol{d}_t^*\right|e^{-|\partial y\boldsymbol{I}_t|}, \tag{33}$$

where $\partial x$ and $\partial y$ denote image gradients along the horizontal and vertical directions, respectively, and $\boldsymbol{d}_t^*$ denotes the mean-normalized inverse depth map.

3) Total loss

We jointly optimize the reprojection loss and the smoothness loss across multiple scales. The overall loss is

$$L_{\text{total}}=\frac{1}{3}\sum_{s\in\left\{1,\frac{1}{2},\frac{1}{4}\right\}}(L_{\text{r}}+\lambda L_{\text{smooth}}), \tag{34}$$

where $S$ denotes the set of output scales, and $\lambda$ is the weight of the smoothness term, set to $1\times10^{-3}$ in this work.

## 3 Experiments

### 3.1 Datasets

KITTI[29] is one of the most influential public datasets for autonomous driving. We followed the split of Eigen et al.[2] and applied the preprocessing protocol of Zhou et al.[8] to remove static frames, resulting in 39 810 monocular triplets for training, 4 424 image–depth pairs for validation, and 697 raw images for evaluation. The input resolution was set to $192\times640$ ($H=192$, $W=640$).

Make3D[30] is a small outdoor-scene depth dataset containing 400 training images and 134 test images. Following the setting of existing works[9], we performed zero-shot evaluation on the 134 test samples using the model trained on KITTI.

### 3.2 Implementation details

We used PyTorch to build the model. Comparative and ablation experiments were conducted on an NVIDIA GeForce RTX 4090 GPU, while on-device evaluations were conducted on RK3588S using the RKNN framework (details are provided in Section 3.5).

Without pretraining, we trained the model from scratch for 35 epochs. With ImageNet-1K pretraining, the encoder was first pretrained on ImageNet-1K for 100 epochs. Then, the pretrained model was transferred to the depth estimation task. We used AdamW with weight decay 0.01 and a batch size of 12, and adopted a cosine annealing learning-rate schedule with an initial learning rate of $1\times10^{-4}$ and a minimum learning rate of $5\times10^{-6}$. DropPath was applied to the residual branch of DB-LKA with a drop probability linearly increased from 0 to 0.2. For data augmentation, we randomly applied horizontal flipping, brightness adjustment $\pm0.2$, saturation adjustment $\pm0.2$, contrast adjustment $\pm0.2$, and hue jitter $\pm0.1$ with a probability of 50%.

Following MonoDepth2[9], we report Abs Rel, Sq Rel, RMSE, RMSE log, and threshold accuracies $\delta_1$, $\delta_2$, and $\delta_3$, defined as

$$\text{Abs Rel}=\frac{1}{N_{\text{valid}}}\sum_{i=1}^{N_{\text{valid}}}\frac{|p_i-g_i|}{g_i}, \tag{35}$$

$$\text{Sq Rel}=\frac{1}{N_{\text{valid}}}\sum_{i=1}^{N_{\text{valid}}}\frac{\|p_i-g_i\|^2}{g_i}, \tag{36}$$

$$\text{RMSE}=\sqrt{\frac{1}{N_{\text{valid}}}\sum_{i=1}^{N_{\text{valid}}}(p_i-g_i)^2}, \tag{37}$$

$$\text{RMSE log}=\sqrt{\frac{1}{N_{\text{valid}}}\sum_{i=1}^{N_{\text{valid}}}(\log p_i-\log g_i)^2}, \tag{38}$$

$$\delta_k=\frac{1}{N_{\text{valid}}}\sum_{i=1}^{N_{\text{valid}}}1\left\{\max\left(\frac{g_i}{p_i},\frac{p_i}{g_i}\right)\right\}<1.25^k,k\in\{1,2,3\}, \tag{39}$$

where $p_i$ and $g_i$ denote the predicted and ground-truth depth values of pixel $i$, respectively; $N_{\text{valid}}$ is the number of valid depth pixels; $1\{\cdot\}$ denotes the indicator function, which takes the value of 1 when the condition inside the braces is satisfied and 0 otherwise. During evaluation, depth is capped to the range $[0, 80]$ m.

### 3.3 Comparative experiments

#### 3.3.1 Results on KITTI

To validate the effectiveness of the proposed method, we evaluated it on KITTI and compared it with a variety of self-supervised monocular depth estimation approaches. The quantitative results are summarized in Table 3.

**Table 3 Quantitative results on KITTI**

| Method | Data | Abs Rel ↓ | Sq Rel ↓ | RMSE ↓ | RMSE log ↓ | $\delta_1$ ↑ | $\delta_2$ ↑ | $\delta_3$ ↑ | Params/M |
|---|---|---|---|---|---|---|---|---|---|
| MonoDepth2[9] | M † | 0.132 | 1.044 | 5.142 | 0.210 | 0.845 | 0.948 | 0.977 | 14.3 |
| R-MSFM3[12] | M † | 0.128 | 0.965 | 5.019 | 0.207 | 0.853 | 0.951 | 0.977 | 3.5 |
| R-MSFM6[12] | M † | 0.126 | 0.944 | 4.981 | 0.204 | 0.857 | 0.952 | 0.978 | 3.8 |
| Lite-Mono[16] | M † | 0.121 | 0.876 | 4.918 | 0.199 | 0.859 | 0.953 | 0.980 | 3.1 |
| RT-MonoDepth[19] | M † | 0.125 | 0.959 | 4.985 | 0.202 | 0.857 | 0.952 | 0.979 | 2.8 |
| RTIA-Mono[17] | M † | **0.118** | **0.858** | **4.810** | **0.196** | **0.865** | **0.955** | **0.980** | 1.4 |
| Ours | M † | **0.118** | 0.874 | 4.871 | 0.197 | 0.863 | **0.955** | **0.980** | 3.0 |
| MonoDepth2[9] | M | 0.115 | 0.903 | 4.863 | 0.193 | 0.877 | 0.959 | 0.981 | 14.3 |
| R-MSFM3[12] | M | 0.114 | 0.815 | 4.713 | 0.191 | 0.876 | 0.960 | 0.981 | 3.5 |
| R-MSFM6[12] | M | 0.112 | 0.806 | 4.704 | 0.191 | 0.878 | 0.959 | 0.981 | 3.8 |
| HR-Depth[13] | M | 0.109 | 0.792 | 4.632 | 0.185 | 0.884 | 0.962 | 0.983 | 14.7 |
| MonoViT-tiny[14] | M | **0.102** | **0.733** | **4.459** | **0.177** | **0.895** | **0.965** | **0.984** | 10.3 |
| Lite-Mono[16] | M | 0.107 | 0.765 | 4.561 | 0.183 | 0.886 | 0.963 | 0.983 | 3.1 |
| Repmono[20] | M | 0.107 | 0.804 | 4.641 | 0.184 | 0.885 | 0.961 | 0.982 | 2.3 |
| LDA-Mono-L[21] | M | 0.104 | 0.765 | 4.535 | 0.180 | 0.893 | 0.964 | 0.983 | 2.0 |
| LEDepth[22] | M | 0.106 | 0.780 | 4.602 | 0.183 | 0.888 | 0.963 | 0.983 | 3.1 |
| Ours | M | 0.105 | 0.744 | 4.496 | 0.179 | 0.892 | 0.964 | **0.984** | 3.0 |

All input images are resized to 192 × 640. "↓" indicates that lower is better, and "↑" indicates that higher is better. The optimal and second-best methods are highlighted in bold and underlined, respectively. M denotes training with monocular videos, and M † denotes the same setting without ImageNet pretraining.

Additionally, Fig. 7 presents qualitative comparison results.

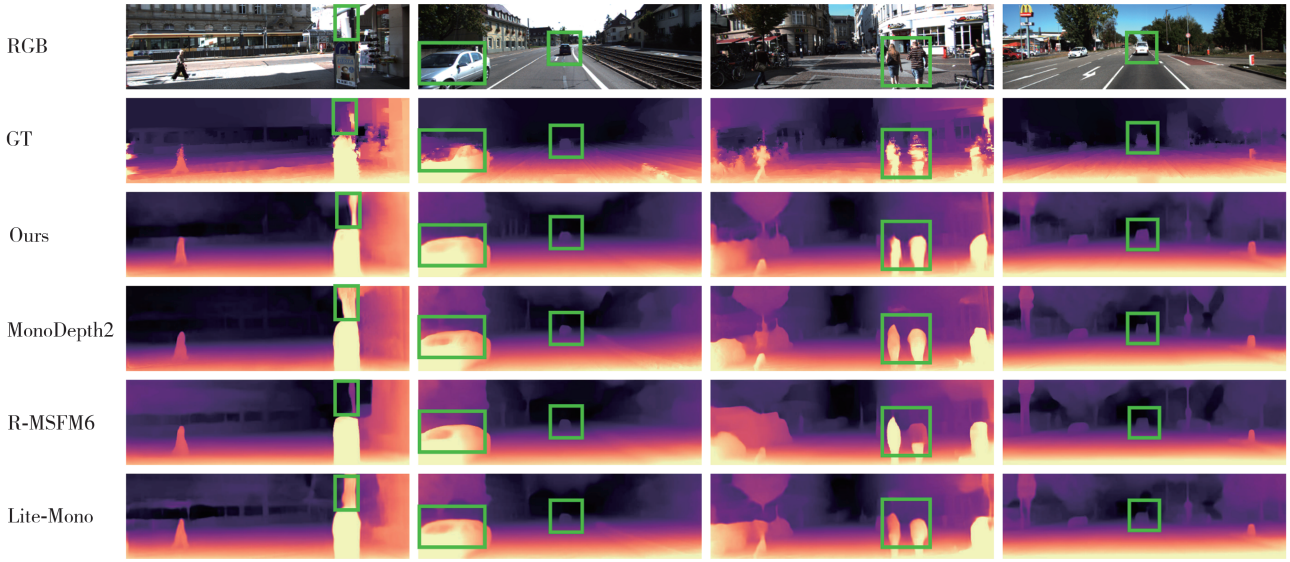



**Fig. 7 Qualitative comparison on KITTI. Green boxes highlight regions with significant differences. Ground truth (GT) is obtained by applying method proposed by Levin to perform depth completion on sparse point clouds**

As shown in Table 3, our model has only 3.0 M parameters while achieving excellent performance on several key metrics. Under the M † setting, our method attains Abs Rel=0.118 and $\delta_1$=0.863, outperforming Lite-Mono[16] and comparable to RTIA-Mono[17]. Under the M setting, our method achieves Abs Rel=0.105 and $\delta_1$=0.892. Compared with the self-supervised baseline MonoDepth2[9], Abs Rel is reduced by 8.7%, with consistent improvements on other error and accuracy metrics. Compared with Lite-Mono[16], our method further reduces the Abs Rel from 0.107 to 0.105. Moreover, compared with the recent lightweight method LEDepth[22], our method achieves competitive performance with a similar parameter budget. Although the transformer-based MonoViT-tiny[14] performs the best, our model achieves comparable accuracy using just 29.1% of its parameters.

Fig. 7 shows a qualitative comparison with GT, MonoDepth2[9], R-MSFM6[12], and Lite-Mono[16] on the KITTI test set. Although GT exhibits noise in some areas, it is evident that the compared methods are prone to depth inconsistencies in the car window regions and suffer from excessive boundary smoothing at thin structures. Meanwhile, on dynamic targets like pedestrians and distant vehicles, other methods generally suffer from depth estimation bias and edge blurring issues. In contrast, the proposed method demonstrates more stable structural recovery capabilities and maintains better depth consistency across geometric regions. These results indicate that the proposed depth estimation model can more accurately restore local details and overall geometric structures in complex scenes, generating higher-quality depth maps.

3.3.2 Results on Make3D

To evaluate generalization, we directly tested the model trained on KITTI on Make3D, and compared it with MonoDepth2[9], R-MSFM6[12], and Lite-Mono[16]. Quantitative and qualitative results are shown in Table 4 and Fig.8.

**Table 4 Quantitative results on Make3D**

| Method | Abs Rel ↓ | Sq Rel ↓ | RMSE ↓ | RMSE log ↓ |
|---|---|---|---|---|
| MonoDepth2[9] | 0.322 | 3.589 | 7.417 | 0.163 |
| R-MSFM6[12] | 0.334 | 3.285 | 7.212 | 0.169 |
| Lite-Mono[16] | **0.305** | <u>3.060</u> | **6.981** | <u>0.158</u> |
| Ours | <u>0.308</u> | **3.020** | <u>7.032</u> | **0.157** |

As shown in Table 4, the proposed method achieves the best performance in terms of Sq Rel and RMSE log metrics, with values of 3.020 and 0.157, respectively. Furthermore, it attains Abs Rel=0.308 and RMSE=7.032, outperforming MonoDepth2 and R-MSFM6, while being slightly worse than Lite-Mono. Fig.8 shows qualitative comparisons, where blue boxes indicate regions with noticeable differences. Our method better recovers the overall scene layout and major structures, demonstrating superior depth consistency across large geometric regions. Overall, the proposed lightweight model demonstrates strong cross-dataset generalization under low parameter and computation budgets.

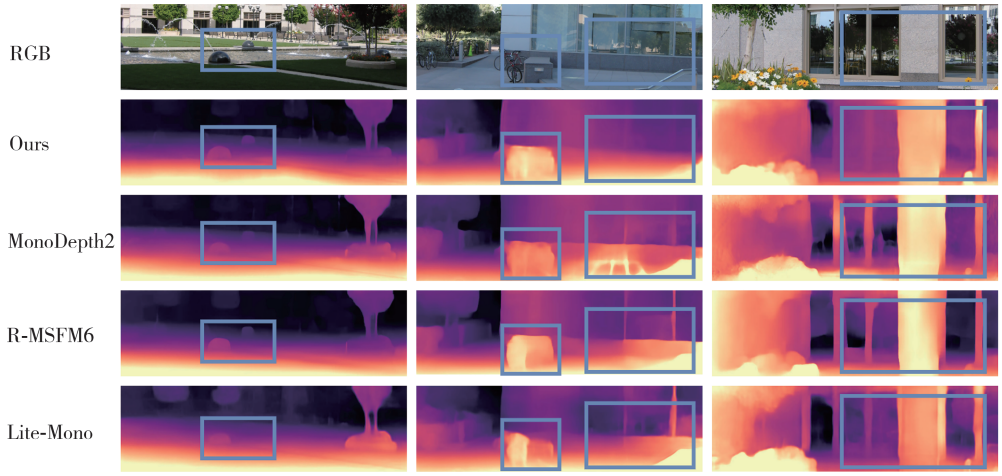



**Fig. 8 Qualitative comparison on Make3D. Blue boxes highlight regions with noticeable differences**

3.3.3 Model complexity and parameter analysis

Table 5 presents a module-wise breakdown of the parameters and FLOPs of the proposed network.

**Table 5 Parameter and complexity allocation within the model**

| Part | Module | Params/M | Parameter ratio/% | FLOPs/G | FLOPs ratio/% |
|---|---|---|---|---|---|
| Encoder | DB-LKA | 2.450 0 | 81.70 | 2.43 | 70.03 |
| | Other layers | 0.383 1 | 12.78 | 0.47 | 13.54 |
| Decoder | DDF | 0.163 9 | 5.47 | 0.56 | 16.14 |
| | Other layers | 0.001 6 | 0.05 | 0.01 | 0.29 |
| Total | | 2.998 6 | 100 | 3.47 | 100 |

Overall, the model contains 2.998 6 M (≈3.0 M) parameters and 3.47 G (≈3.5 G) FLOPs. Among these, the DB-LKA blocks, as the core modeling units in the encoder, contribute 81.70% of the total parameters and 70.03% of the total FLOPs, indicating that most of the computational budget is allocated to long-range and local collaborative representation learning. Within the decoder, DDF is the dominant complexity source, while the remaining decoder layers introduce negligible overhead. These results show that the proposed model concentrates its limited complexity budget on the key modules, rather than on auxiliary layers.

To assess computational efficiency, we benchmarked inference on an NVIDIA GeForce RTX 4090 with an input size of 192×640 and a batch size of 16. After 50 warm-up iterations, we measured the average inference time per image over 100 iterations.

The results are shown in Table 6. Our method runs at 0.98 ms per image, ranking behind only MonoDepth2 (0.65 ms) and RT-MonoDepth (0.59 ms). Compared with the heavier MonoViT-tiny model, our method substantially reduces both parameters (3.0 M) and FLOPs (3.5 G). Compared to Lite-Mono, our method achieves a higher $\delta_1$ value of 0.892 with a similar model size and lower latency. In summary, our method achieves a good balance among complexity, efficiency, and accuracy.

**Table 6 Computational efficiency and complexity analysis**

| Method | Encoder | | Decoder | | Full model | | Latency/ms | $\delta_1$ |
|---|---|---|---|---|---|---|---|---|
| | Params/M | FLOPs/G | Params/M | FLOPs/G | Params/M | FLOPs/G | | |
| MonoDepth2[9] | 11.2 | 4.5 | 3.1 | 3.5 | 14.3 | 8.0 | 0.65 | 0.877 |
| R-MSFM3[12] | 0.7 | 2.4 | 2.8 | 14.1 | 3.5 | 16.5 | 3.81 | 0.876 |
| R-MSFM6[12] | 0.7 | 2.4 | 3.1 | 28.8 | 3.8 | 31.2 | 4.32 | 0.878 |
| MonoViT-tiny[14] | 5.6 | 7.8 | 4.7 | 15.9 | 10.3 | 23.7 | 2.61 | 0.895 |
| Lite-Mono[16] | 2.9 | 4.4 | 0.2 | 0.7 | 3.1 | 5.1 | 1.08 | 0.886 |
| RT-MonoDepth[19] | 2.0 | 2.3 | 0.8 | 7.5 | 2.8 | 9.8 | 0.59 | 0.857 |
| Ours | 2.8 | 2.9 | 0.2 | 0.6 | 3.0 | 3.5 | 0.98 | 0.892 |

## 3.4 Ablation studies

We conducted ablation studies on KITTI to evaluate the contribution of each design component. Since ImageNet pretraining was time-consuming, all ablation experiments were performed without pretrained weights to ensure fair comparison. Additionally, to better reflect the effectiveness of lightweight designs, we measured the average inference latency per frame using an FP16 model under single-threaded conditions on the RK3588S (testing criteria detailed in Section 3.5.1).

3.4.1 Ablation on DB-LKA

To verify the complementarity between the long-range branch (LKA) and the local branch (DWConv) in DB-LKA, we constructed two variants: removing the local branch (w/o DWConv) and removing the long-range branch (w/o LKA), while keeping other settings unchanged. The quantitative results are reported in

Table 7, and the qualitative comparisons are shown in Fig.9.

**Table 7 Ablation results of DB-LKA block**

| Method | Abs Rel ↓ | Sq Rel ↓ | RMSE ↓ | RMSE log ↓ | $\delta_1$ ↑ | $\delta_2$ ↑ | $\delta_3$ ↑ | Params/M | FLOPs/G | Latency/ms |
|---|---|---|---|---|---|---|---|---|---|---|
| w/o DWConv | 0.120 | 0.924 | 4.928 | 0.201 | 0.860 | 0.953 | 0.979 | 2.8 | 3.3 | 54.2 |
| w/o LKA | 0.128 | 0.976 | 5.126 | 0.209 | 0.851 | 0.951 | 0.977 | 2.5 | 2.9 | 47.6 |
| Full DB-LKA | 0.118 | 0.874 | 4.871 | 0.197 | 0.863 | 0.955 | 0.980 | 3.0 | 3.5 | 56.3 |

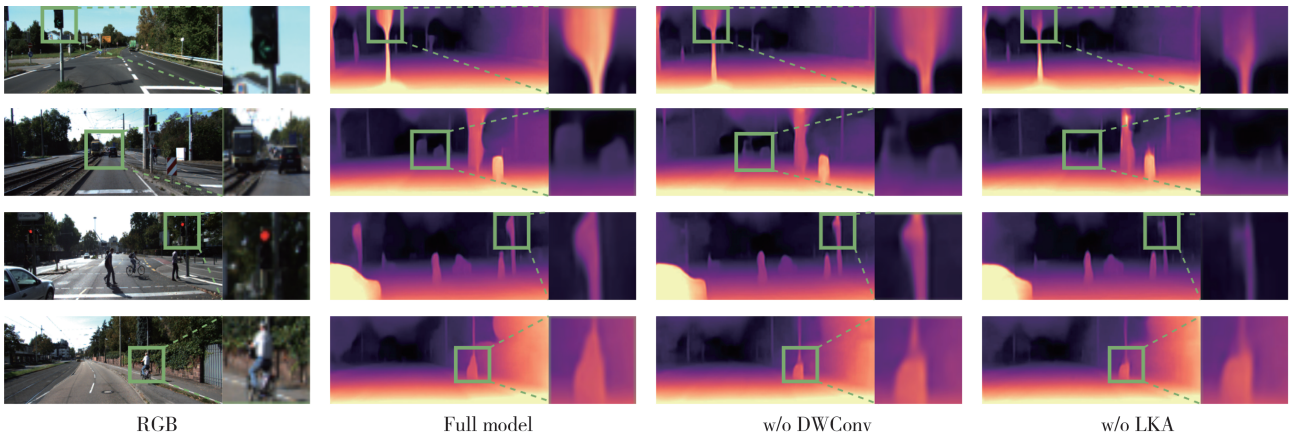



**Fig. 9 Qualitative comparison of different model variants. Green boxes highlight regions where full model performs better**

The full DB-LKA achieves the best performance and predicts more accurate object structures and sharper boundaries. Removing the long-range branch (w/o LKA) reduces parameters and FLOPs but leads to a significant drop in accuracy and is prone to the loss of object structure. Removing the local branch (w/o DWConv) also degrades accuracy, resulting in blurred object edges. These results indicate that the long-range and local branches are complementary and can effectively enhance depth estimation performance.

From the on-device efficiency perspective, w/o LKA yields the lowest average latency (47.6 ms), whereas the full DB-LKA incurs 56.3 ms. Although the full model incurs additional overhead, it delivers significant improvements in accuracy.

3.4.2 Ablation on decoding fusion block

The cross-scale fusion strategy in the decoder directly affects detail restoration quality and decoding cost. To validate the effectiveness of the proposed differential-driven dynamic fusion, we compare DDF with two common fusion methods: Concat+3×3 Conv (Fig.10(a)) and Add+3×3 Conv (Fig.10(b)). In addition to comparing their overall performance on KITTI, we also evaluated the three fusion blocks separately in edge and flat regions.

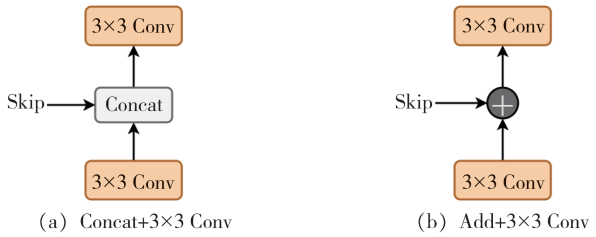



**Fig. 10 Two common fusion methods**

As shown in Table 8, the two static fusion baselines (Add+3×3 Conv and Concat+3×3 Conv) are operator-friendly and have slightly lower average latency, but they exhibit an evident accuracy bottleneck. In contrast, DDF achieves better accuracy with a tolerable additional latency, while maintaining fewer parameters and FLOPs than the static fusion baselines.

**Table 8 Overall ablation results for decoded fusion block**

| Method | Abs Rel ↓ | Sq Rel ↓ | RMSE ↓ | RMSE log ↓ | $\delta_1$ ↑ | $\delta_2$ ↑ | $\delta_3$ ↑ | Params/M | FLOPs/G | Latency/ms |
|---|---|---|---|---|---|---|---|---|---|---|
| Concat+3×3Conv | 0.121 | 0.887 | 4.905 | 0.200 | 0.859 | 0.954 | 0.980 | 3.3 | 4.5 | 52.4 |
| Add+3×3Conv | 0.123 | 0.938 | 4.947 | 0.204 | 0.857 | 0.953 | 0.980 | 3.2 | 4.1 | 51.2 |
| DDF | 0.118 | 0.874 | 4.871 | 0.197 | 0.863 | 0.955 | 0.980 | 3.0 | 3.5 | 56.3 |

We used the Canny edge detection algorithm to extract the edge map from the ground-truth depth map and applied a 3-pixel dilation to the edge map. Then, we partitioned the sparse LiDAR point cloud into edge and flat regions based on the edge map, and finally computed the metrics on the valid points within these two regions. The quantitative results are shown in Table 9. Our method achieved the best performance in both edge regions and flat regions. Fig.11 presents a visual comparison of the three decoding blocks. It can be seen that our method is able to recover clearer

structural edges and exhibits better depth consistency across large geometric regions.

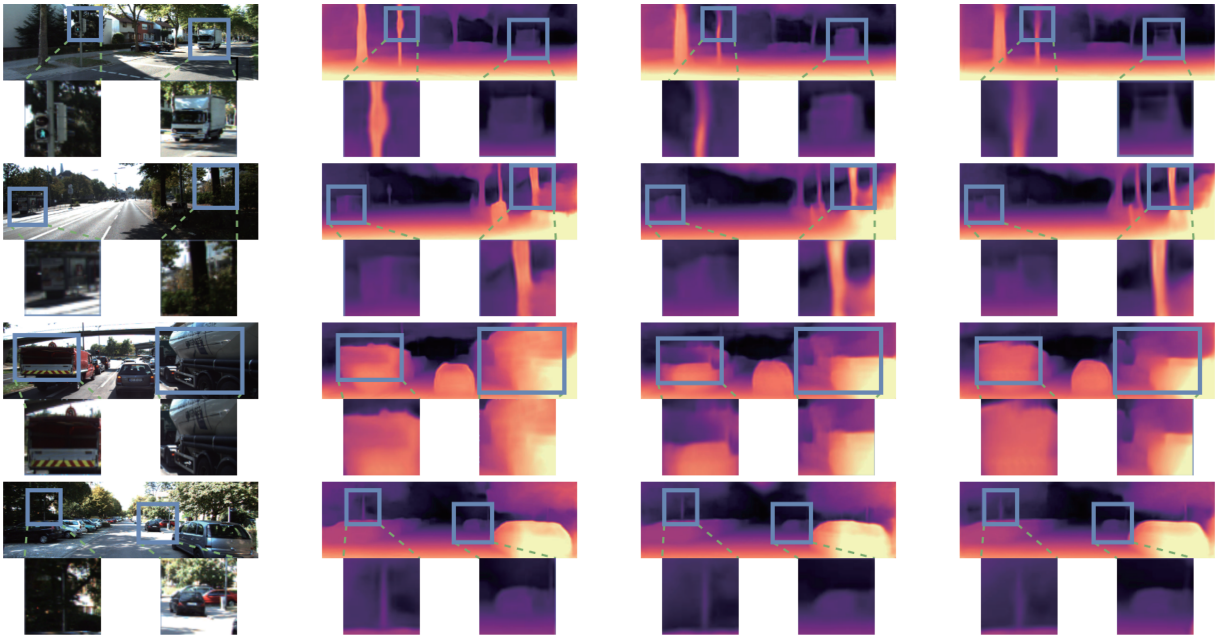



**Fig. 11 Qualitative comparison of different fusion modules. Blue rectangle highlights where DDF performs well**

**Table 9 Performance of decoder fusion blocks in edge and flat regions**

| Method | Edge | | | Flat | | |
|---|---|---|---|---|---|---|
| | Abs Rel ↓ | RMSE ↓ | $\delta_1$ ↑ | Abs Rel ↓ | RMSE ↓ | $\delta_1$ ↑ |
| Concat+3×3Conv | 0.258 | 7.435 | 0.641 | 0.122 | 4.852 | 0.864 |
| Add+3×3Conv | 0.257 | 7.386 | 0.642 | 0.122 | 4.930 | 0.862 |
| DDF | 0.251 | 7.378 | 0.646 | 0.116 | 4.683 | 0.871 |

These results demonstrate that, compared with static addition/concatenation followed by convolution, the proposed DDF achieves a more favorable accuracy–efficiency trade-off through dynamic fusion.

3.4.3 Ablation on DDF module

To further quantify the contribution of key components inside DDF, we individually removed the DDMF branch (w/o DDMF) and the DRM branch (w/o DRM). The results are reported in Table 10. Both removals lead to performance degradation. Without DDMF, Abs Rel increases from 0.118 to 0.124, and $\delta_1$ drops from 0.863 to 0.856. Without DRM, accuracy also decreases (Abs Rel=0.120, $\delta_1$=0.860). This indicates that both DDMF and DRM are crucial to the performance gains. Meanwhile, the full DDF adds only about 1.5–4.7 ms of latency compared with the simplified variants, and it effectively improves prediction accuracy, demonstrating excellent cost-effectiveness.

**Table 10 Ablation results of DDF module**

| Method | Abs Rel ↓ | Sq Rel ↓ | RMSE ↓ | RMSE log ↓ | $\delta_1$ ↑ | $\delta_2$ ↑ | $\delta_3$ ↑ | Params/M | FLOPs/G | Latency/ms |
|---|---|---|---|---|---|---|---|---|---|---|
| w/o DDMF | 0.124 | 0.927 | 4.978 | 0.203 | 0.856 | 0.953 | 0.979 | 3.0 | 3.4 | 51.6 |
| w/o DRM | 0.120 | 0.879 | 4.895 | 0.199 | 0.860 | 0.954 | 0.979 | 3.0 | 3.2 | 54.8 |
| Full DDF | 0.118 | 0.874 | 4.871 | 0.197 | 0.863 | 0.955 | 0.980 | 3.0 | 3.5 | 56.3 |

3.4.4 Ablation on loss functions

To validate the effectiveness of the temporal soft fusion reprojection (TSFR) loss function and explain the rationale behind the selection of the temperature parameter $\tau$, we conducted a sensitivity analysis by varying $\tau \in \{0.001, 0.01, 0.05, 0.1, 0.2, 0.5, 1.0\}$. The results are shown in Table 11. Subsequently, we applied TSFR to MonoDepth2, Lite-Mono, and our proposed network, comparing them against the minimum reprojection (MR) loss as a baseline. The results are shown in Table 12.

**Table 11 Ablation results of temperature parameter $\tau$**

| $\tau$ | Abs Rel ↓ | Sq Rel ↓ | RMSE ↓ | RMSE log ↓ | $\delta_1$ ↑ | $\delta_2$ ↑ | $\delta_3$ ↑ |
|---|---|---|---|---|---|---|---|
| 0.001 | 0.120 | 0.887 | 4.906 | 0.199 | 0.860 | 0.954 | 0.979 |
| 0.01 | 0.119 | 0.926 | 4.877 | 0.200 | 0.859 | 0.953 | 0.979 |
| 0.05 | 0.118 | 0.869 | 4.864 | 0.198 | 0.862 | 0.954 | 0.979 |
| 0.1 | 0.118 | 0.874 | 4.871 | 0.197 | 0.863 | 0.955 | 0.980 |
| 0.2 | 0.120 | 0.914 | 4.928 | 0.199 | 0.856 | 0.953 | 0.978 |
| 0.5 | 0.123 | 0.965 | 5.091 | 0.204 | 0.854 | 0.949 | 0.977 |
| 1 | 0.125 | 0.968 | 5.125 | 0.206 | 0.852 | 0.949 | 0.977 |

In Eq. (28), $\tau$ directly controls the entropy of the fusion weight. When $\tau$ is very small, the weight distribution becomes highly peaked, and TSFR approaches a minimum-

selection strategy, so gradients are dominated by only one source frame. As $\tau$ increases, both the forward and backward frames can participate in supervision, enabling the model to exploit complementary observations from temporally adjacent views. As shown in Table 11, the best results occur within the moderate range of $\tau$ from 0.05 to 0.1. Specifically, when $\tau$=0.1, RMSE log and the three threshold accuracy metrics perform best, while maintaining the same Abs Rel as when $\tau$=0.05. In contrast, when $\tau$=0.001, performance approaches the minimum reprojection baseline of MonoDepth2 (see Table 12), confirming that when $\tau$ is too low, TSFR degenerates into minimum reprojection loss. When $\tau \geqslant 0.2$, the weights become overly smoothed, and high-error source pixels make a non-negligible contribution, leading to increased noise in the supervised signal and a gradual decline in accuracy.

**Table 12 Ablation results of loss functions**

| Loss function | Network | Abs Rel ↓ | Sq Rel ↓ | RMSE ↓ | RMSE log ↓ | $\delta_1$ ↑ | $\delta_2$ ↑ | $\delta_3$ ↑ |
|---|---|---|---|---|---|---|---|---|
| MR | MonoDepth2 | 0.131 | 1.024 | 5.136 | 0.210 | 0.846 | 0.949 | 0.977 |
| | Lite-Mono | 0.121 | 0.881 | 4.926 | 0.200 | 0.858 | 0.953 | 0.979 |
| | Ours | 0.120 | 0.892 | 4.894 | 0.198 | 0.860 | 0.953 | 0.979 |
| TSFR loss $\tau$=0.1 | MonoDepth2 | 0.128 | 0.984 | 5.042 | 0.208 | 0.848 | 0.951 | 0.977 |
| | Lite-Mono | 0.119 | 0.865 | 4.897 | 0.198 | 0.861 | 0.954 | 0.980 |
| | Ours | 0.118 | 0.874 | 4.871 | 0.197 | 0.863 | 0.955 | 0.980 |
| TSFR loss $\tau$=0.001 | MonoDepth2 | 0.130 | 1.046 | 5.131 | 0.208 | 0.848 | 0.950 | 0.977 |
| | Lite-Mono | 0.120 | 0.891 | 4.914 | 0.199 | 0.859 | 0.953 | 0.979 |
| | Ours | 0.120 | 0.887 | 4.906 | 0.199 | 0.860 | 0.954 | 0.979 |

To illustrate the effect of $\tau$ on the weight distribution more intuitively, Fig.12 shows the fusion weight maps for different values of $\tau$. It can be seen that when $\tau$=0.001, the weight map approximates a binary image, similar to the minimum reprojection loss. When $\tau$=1, the weight of each pixel in the weight map is almost equal between the previous and next frames, which is similar to the average reprojection loss. In contrast, when $\tau$=0.1, the weight map is more balanced: in regions such as those obscured by walls or moving vehicles. It suppresses source pixels with larger errors, preventing the introduction of large errors that are difficult to optimize. Whereas in regions such as the road surface where both frames are reconstructed well, it calculates the loss for each frame more uniformly, allowing the losses of both the previous and current frames to effectively participate in the supervised optimization process. These results indicate that TSFR benefits from a balanced soft selection mechanism, rather than extreme hard selection or uniform fusion, thereby enhancing the quality of supervision.

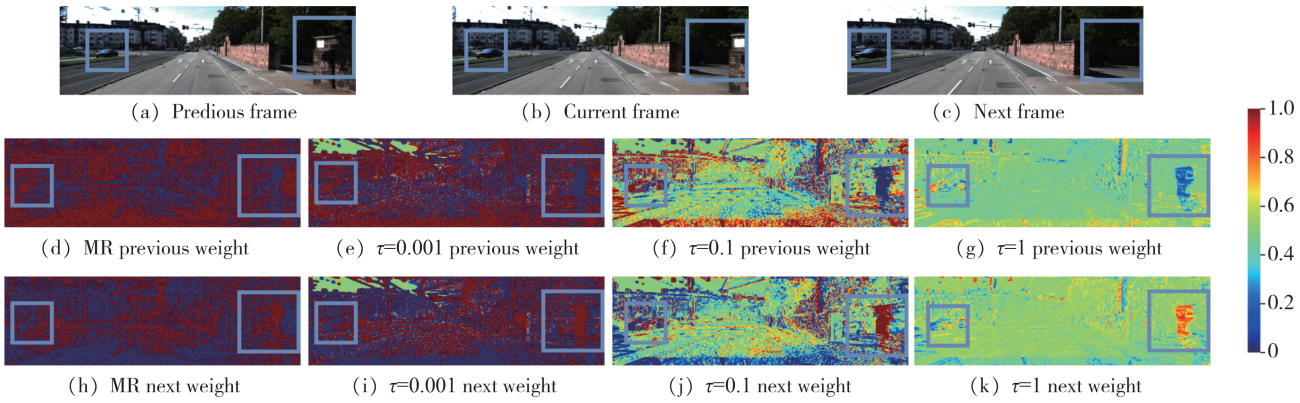



**Fig. 12 Visual comparison of fusion weight maps**

To verify the universality of TSFR, we applied TSFR to MonoDepth2, Lite-Mono, and our proposed network under conditions of $\tau$ =0.1 and $\tau$ =0.001, using the minimum reprojection loss as a benchmark for comparison. As shown in Table 12, the temporal soft fusion strategy consistently improves performance across different network architectures. This indicates that the strategy enhances training robustness and can serve as a generalized enhancement method for self-supervised depth estimation frameworks. When $\tau$=0.001, the performance of all three networks approaches the level achieved using the minimum reprojection loss, which further demonstrates that the temporal soft fusion strategy degenerates into minimum error reconstruction. Furthermore, regardless of whether the minimum reprojection loss function or our proposed temporal soft fusion loss function is used, our network outperforms the baseline models, further validating the effectiveness of the proposed network architecture.

## 3.5 Performance evaluation of RK3588S

### 3.5.1 Experimental setup

To evaluate the deployment performance of the proposed model in real-time on-device scenarios, we chose the Rockchip RK3588S as the test platform. The hardware/

software configuration is summarized in Table 13. As a representative domestic edge AI platform, RK3588S features an 8-core ARM CPU (4×Cortex-A76+4×Cortex-A55) with 8 GB memory and an integrated 3-core NPU (~6 TOPS). It supports various types of precision, such as INT8 and FP16, and provides the RKNN Toolkit 2 and the RKNPU runtime toolchain. These tools enable the conversion and deployment of mainstream deep learning models into the RKNN format. As the proposed model is primarily composed of easily deployable operators such as standard convolution, depthwise convolution, pointwise convolution, bilinear upsampling, pooling, and element-wise operations, it is well suited to RKNN-based conversion and inference workflows. This makes it highly compatible with domestic edge AI platforms such as the RK3588S.

**Table 13 Configuration of RK3588S test environment**

| Item | Configuration/Version |
|---|---|
| Development board | Orange Pi 5 |
| Architecture | aarch64 |
| Operating system | Linux |
| Kernel | 5.10.160-rockchip-rk3588 |
| NPU driver | 0.9.6 |
| IOMMU | Enabled |
| RKNN Runtime | librknnrt.so 2.3.2 |
| RKNN-Toolkit2 | 2.3.2 |
| CPU frequency policy | Performance |
| NPU frequency policy | Performance |

We first exported the PyTorch model to ONNX, then converted it to RKNN using RKNN-Toolkit2, and finally performed inference with the RKNN runtime. The conversion types for RKNN models are categorized into non-quantized and quantized. Non-quantized conversion yields an FP16 RKNN model. Quantized conversion is further divided into hybrid quantization and INT8 quantization. INT8 quantization produces an INT8 R-KNN model. In hybrid quantization, operator layers that suffer an accuracy degradation beyond a predefined threshold after quantization are retained in FP16, while the remaining layers are quantized to INT8. This results in a hybrid model comprising both INT8 and FP16 components. In the RKNN conversion process, we did not introduce custom operator rewrites or manual optimizations for specific modules. Instead, we utilized the user-friendly operators provided by RKNN. Specifically, pointwise convolution, depthwise convolution, and standard convolution were exported as the Conv operators supported by RKNN, while bilinear upsampling and element-wise addition/multiplication were directly mapped to the corresponding operators.

The input resolution is 192×640 and the batch size is 1. We performed 50 warm-up runs, and then measured the average latency $L$ (ms), throughput $T$ (frames/s), parallel efficiency $\eta$ (%), and peak memory $PM$ (MB) over 100 runs. Throughput and parallel efficiency are computed by

$$T(N_{\text{th}}) = N_{\text{th}} \cdot \frac{1\,000}{L}, \tag{40}$$

$$\eta(N_{\text{th}}) = \frac{T(N_{\text{th}})}{N_{\text{th}} \cdot T(1)} \times 100\%, \tag{41}$$

where $N_{\text{th}}$ is the number of threads. During the evaluation process, inference threads are pinned to Cortex-A76 large cores with the objective of reducing scheduling variance, and all three NPU cores are enabled. In addition, to reflect the accuracy impact of model conversion, we computed $\delta_1$ and Abs Rel on the KITTI test set for the RKNN models following the protocol in Section 3.2. Unless otherwise stated, all experiments were conducted under the above evaluation settings.

While $\delta_1$ and Abs Rel reflect the effect of RKNN conversion on the final depth accuracy with respect to ground truth, they do not directly quantify the numerical deviation between the converted RKNN model and the original FP32 model. Therefore, we further introduced five additional metrics: Mean diff, P95, $\gamma_1$, $\gamma_2$, and $\gamma_3$. Let $d_i^{\text{F}}$ and $d_i^{\text{R}}$ denote the depth values predicted by the FP32 model and the RKNN model, respectively, at valid pixel $i$, and let $N_{\text{p}}$ be the number of pixels. These five indicators can be expressed as

$$\text{Mean diff} = \frac{1}{N_{\text{p}}} \sum_{i}^{N_{\text{p}}} \left| d_i^{\text{R}} - d_i^{\text{F}} \right|, \tag{42}$$

$$\text{P95} = Q_{0.95}\left( \left\{ \left| d_i^{\text{R}} - d_i^{\text{F}} \right| \right\}_i^{N_{\text{p}}} \right), \tag{43}$$

$$\gamma_k = \frac{1}{N_{\text{p}}} \sum_{i}^{N_{\text{p}}} 1\left\{ \frac{\left| d_i^{\text{R}} - d_i^{\text{F}} \right|}{d_i^{\text{F}}} < 0.1k \right\}, \quad k \in \{1,2,3\}, \tag{44}$$

where $Q_{0.95}(\cdot)$ denotes the 95th-percentile operator. Mean diff and P95 evaluate the mean value of numerical deviation and the magnitude of tail differences, respectively, while $\gamma_1$, $\gamma_2$, and $\gamma_3$ measure the proportion of valid pixels with relative deviations within the 10%, 20%, and 30% ranges, respectively. For Mean diff and P95, lower values indicate better consistency with the FP32 reference values, whereas for these three relative deviation ratios, higher values are preferable. During evaluation, depth is limited to [0, 80] m.

3.5.2 Multithreading performance

To evaluate the on-device inference performance on the RK3588S, PyTorch models based on this method and several other lightweight approaches were converted to FP16 precision RKNN models. To ensure a fair comparison, optimization techniques such as weight

compression and model pruning were not employed during the conversion process. Finally, the performance of these models was evaluated under the RKNN inference framework across different numbers of threads.

As shown in Fig. 13, throughput increases with the number of threads and saturates after three threads, while the average latency increases. At higher thread counts, parallel efficiency declines and peak memory (PM) grows approximately monotonically. Considering throughput, latency, and memory footprint jointly, we chose three threads as the default configuration on RK3588S, and all subsequent evaluations are reported under this setting.

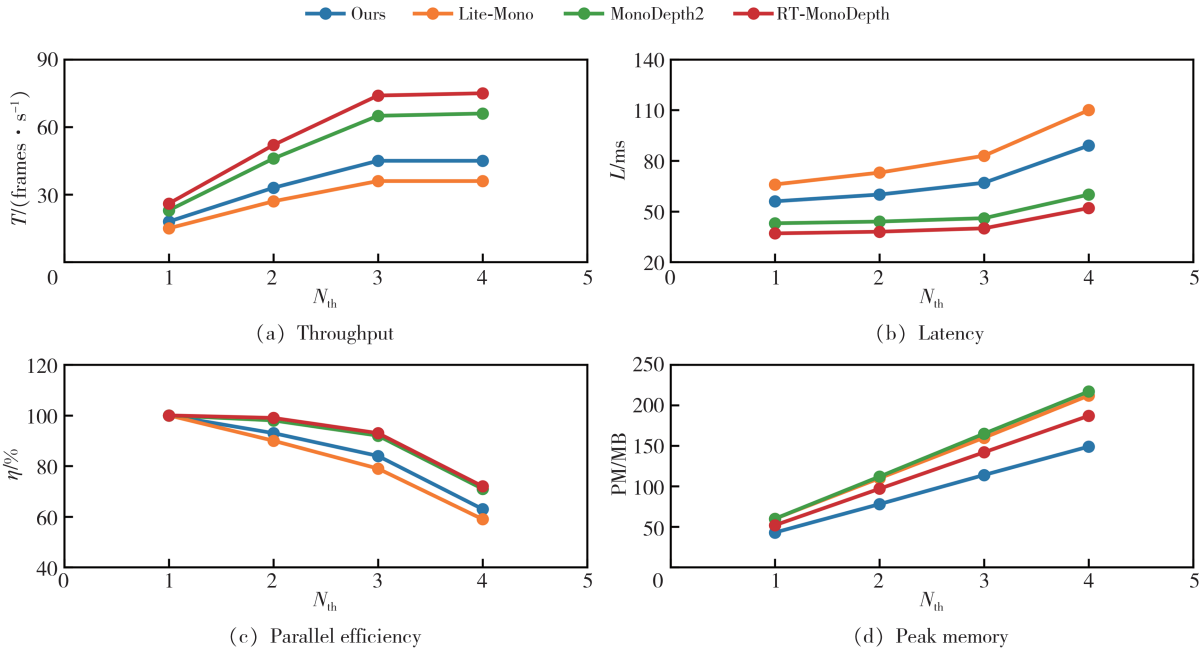



**Fig. 13 Multi-thread inference performance and resource usage on RK3588S**

Table 14 reports the performance of each method on RK3588S with three threads. Our method consumes only 113 MB of peak memory, significantly lower than other approaches. Although its throughput is lower than MonoDepth2 and RT-MonoDepth, it comfortably meets the real-time requirement of 30 frames/s (achieving 45 frames/s) while offering better accuracy. Our method provides a more balanced trade-off among accuracy, throughput, and memory, making it more suitable for real-time edge deployment.

**Table 14 On-device performance on RK3588S with three threads**

| Method | $L$/ms | $T$/(frames•s$^{-1}$) | $\eta$/% | PM/MB | $\delta_1$ | Abs Rel | Model size/MB |
|---|---|---|---|---|---|---|---|
| MonoDepth2 | 46.7 | 65 | 92.4 | 164 | 0.877 | 0.115 | 28.2 |
| Lite-Mono | 83.0 | 36 | 79.3 | 161 | 0.886 | 0.107 | 12.8 |
| RT-MonoDepth | 40.1 | 75 | 93.4 | 142 | 0.857 | 0.125 | 6.3 |
| Ours | 66.7 | 45 | 84.1 | 113 | 0.892 | 0.105 | 8.6 |

3.5.3 Quantized deployment performance

To further improve on-device inference efficiency, we conducted quantization experiments. The PyTorch model of our method was converted into RKNN models with three different numerical precisions: FP16, hybrid, and INT8. For hybrid and INT8 quantization, we selected 200 images from diverse scenarios in the KITTI training dataset as calibration data and used the KL-divergence algorithm. For hybrid quantization, we set the cosine-similarity threshold to 0.999. Operators below the threshold are kept in FP16. Finally, based on these three models, we evaluated the performance and resource consumption of each model under three threads, analyzed the pixel differences resulting from the RKNN conversion, and further examined the performance and resource consumption of the hybrid model under different thread configurations.

Table 15 summarizes the performance of RKNN models with different precisions on RK3588S (RKNN does not support FP32 inference). Therefore, FP32 results are reported only in terms of $\delta_1$, Abs Rel, and model size). The FP16 model preserves the same accuracy as FP32 while reducing the model size by about 27.1%, and runs at 45 frames/s. The INT8 model further compresses the model to 5.1 MB and reaches 67 frames/s, but suffers a noticeable accuracy drop. The hybrid model achieves the same inference speed of 67 frames/s with a model size

comparable to INT8 (5.2 MB) while maintaining nearly lossless accuracy (Abs Rel increases by only 0.001). Hybrid quantization provides the best balance among accuracy, throughput, and model size.

**Table 15 Inference performance of different RKNN models on RK3588S with 3 threads**

| Model | $L$/ms | $T$/(frames•s$^{-1}$) | $\eta$/% | PM/MB | $\delta_1$ | Abs Rel | Model size/MB |
|---|---|---|---|---|---|---|---|
| FP32 | — | — | — | — | 0.892 | 0.105 | 11.8 |
| FP16 | 66.7 | 45 | 84.1 | 113 | 0.892 | 0.105 | 8.6 |
| Hybrid | 44.8 | 67 | 92.9 | 110 | 0.891 | 0.106 | 5.2 |
| INT8 | 44.5 | 67 | 92.9 | 104 | 0.886 | 0.113 | 5.1 |

The $\delta_1$ and Abs Rel metrics on the KITTI dataset reflect the impact of the RKNN conversion on accuracy, but do not intuitively illustrate the differences between the RKNN models and the FP32 model. To address this, we conducted further experiments to evaluate these differences. Table 16 presents the results of a quantitative comparison between the three RKNN models and the FP32 model on the KITTI dataset.

The deviation between FP16 and FP32 is minimal. Although the Hybrid model performs slightly worse than FP16, it is still significantly better than the INT8 model. To provide a more intuitive comparison of the prediction stability among the three RKNN models, we conducted a visual comparison of prediction results across multiple real-world scenarios. Fig. 14 shows a comparison of depth prediction results in real-world scenarios.

**Table 16 Results of difference analysis**

| Model | Mean diff | P95 | $\gamma_1$ | $\gamma_2$ | $\gamma_3$ |
|---|---|---|---|---|---|
| FP16 | 0.258 | 0.589 | 0.698 | 0.833 | 0.888 |
| Hybrid | 0.264 | 0.633 | 0.667 | 0.816 | 0.876 |
| INT8 | 0.271 | 0.675 | 0.629 | 0.784 | 0.860 |

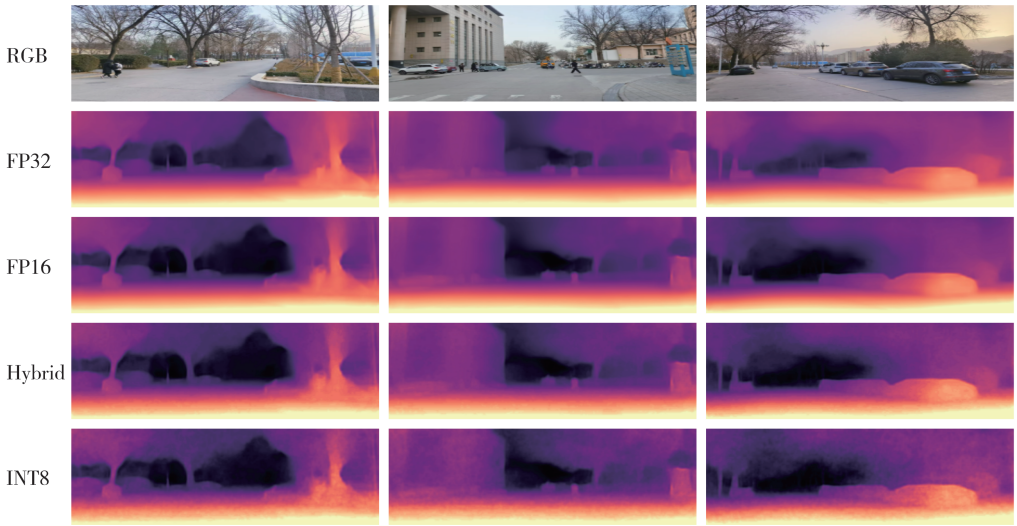



**Fig. 14 Qualitative comparison of different precisions in real scenes**

The depth maps generated by the FP16 and hybrid quantization models are of a quality close to that of FP32, whereas the INT8 model introduces noticeable noise. Overall, the hybrid quantization model maintains the quality of the depth map well, whereas the INT8 model introduces more scattered local deviations. These results validate that hybrid quantization preserves the accuracy of depth predictions better than INT8 quantization, achieving a good balance between accuracy and efficiency.

To further evaluate the practical deployment performance of the hybrid quantization model on RK3588S, we analyzed the performance and resource consumption under different thread configurations. The results are shown in Table 17. As the number of threads is increased from 1 to 3, the throughput is improved substantially from 24 frames/s to 67 frames/s, indicating that multi-thread execution can effectively enhance inference parallelism on the target platform. Meanwhile, the average latency is increased only slightly from 41.1 ms to 44.8 ms, and the parallel efficiency is remained high at 92.9%, suggesting that the hardware resources are still efficiently utilized under the 3-thread setting. However, when the thread number is further increased to 4, the throughput is improved only marginally from 67 frames/s to 68 frames/s, whereas the average latency is increased significantly to 58.1 ms, the parallel efficiency is dropped to 70.8%, and the peak memory usage is rose from 110 MB to 145 MB. This indicated that the RK3588S platform was already close to saturation at 3 threads, and the additional thread mainly introduced extra scheduling overhead and memory-access contention rather than effective performance gain. In terms of power consumption, the average power is increased from

1.6 W in the idle state as the thread number is increased, reaching 4.8 W under both 3-thread and 4-thread settings. This suggested that the compute and memory subsystems had already approached their practical operating limit at 3 threads, so further increasing the thread number did not significantly raise the effective computational intensity. Therefore, considering throughput, latency, parallel efficiency, memory usage, and power consumption together, the 3-thread configuration provides the best overall trade-off for edge deployment on RK3588S.

**Table 17 Comparison of inference performance and resource consumption for hybrid quantization models across different threads on RK3588S platform**

| State | $L$/ms | $T$/(frames·s$^{-1}$) | $\eta$/% | PM/MB | Avg power/W |
|---|---|---|---|---|---|
| Idle | — | — | — | — | 1.6 |
| 1 thread | 41.1 | 24 | 100.0 | 40 | 3.2 |
| 2 thread | 42.4 | 47 | 97.9 | 75 | 4.0 |
| 3 thread | 44.8 | 67 | 92.9 | 110 | 4.8 |
| 4 thread | 58.1 | 68 | 70.8 | 145 | 4.8 |

3.5.4 Quantization-sensitive operator analysis

To further investigate the sources of performance gains from hybrid quantization, we used RKNN-Toolkit2 to identify the network layers in the hybrid precision quantization model that were retained for FP16 computation, and then measured the inference latency of each module. The results are shown in Tables 18 and 19.

**Table 18 Distribution of FP16 operators in hybrid quantization models**

| Module | Original operator | RKNN operator | Output($H$, $W$, $C$) |
|---|---|---|---|
| Stem | DWConv K3 | Conv | (96,320,48) |
| | DWConv K3 | Conv | (96,320,48) |
| DDF(F4→F3) | Mul(Eq.20) | Mul | (24,80,96) |
| DDF(F3→F2) | Add(Eq.16) | Add | (48,160,48) |
| | Add(Eq.16) | Add | (48,160,48) |
| | Mul(Eq.20) | Mul | (48,160,48) |
| DDF(F2→F1) | Mul(Eq.20) | Mul | (48,160,32) |

**Table 19 Module-wise latency breakdown of RKNN models (single-thread/module-wise profiling)**

| Part | Module | FP16 latency/ms | Hybrid latency/ms | INT8 latency/ms |
|---|---|---|---|---|
| Encoder | Stem | 3.61 | 3.32 | 2.78 |
| | DB-LKA | 33.35 | 20.65 | 20.74 |
| | Pool layers | 1.37 | 0.81 | 0.74 |
| | DownSamples | 1.22 | 1.05 | 1.05 |
| Decoder | DDF | 13.74 | 12.08 | 10.92 |
| | UpSamples | 1.76 | 1.29 | 1.38 |
| | Predict head | 2.55 | 2.44 | 2.51 |
| Total | | 57.60 | 41.64 | 40.12 |

Table 18 lists the operators that remain in FP16 under hybrid quantization. Rather than being uniformly distributed, these operators are concentrated in two locations: the two depthwise convolutions in the high-resolution stem and several element-wise Add/Mul operators inside DDF. This distribution reveals the practical quantization strategy adopted by the hybrid model: the convolutional layers that dominate arithmetic cost are quantized to INT8 to maximize throughput, whereas a small set of accuracy-sensitive operators is preserved in FP16 to protect shallow detail extraction and pixel-wise fusion weights.

The module-wise latency breakdown in Table 19 further clarifies the performance impact of this strategy. Compared with FP16, total latency is reduced from 57.60 ms to 41.64 ms after hybrid quantization, and DB-LKA alone contributes 79.6% of this reduction (33.35 ms to 20.65 ms), showing that the encoder remains the primary runtime bottleneck even after quantization. By contrast, the total latency gap between hybrid and INT8 is only 1.52 ms, primarily attributable to DDF (1.16 ms) and the stem (0.54 ms), which is consistent with the locations of the retained FP16 operators in Table 18, while the remaining modules show only negligible differences. Therefore, the hybrid model achieves a speed similar to INT8 with virtually no loss of precision, because only a minimal subset of sensitive operators retain FP16 computation. It is worth noting that the inference latency of the DB-LKA module accounts for the highest proportion across all three RKNN models. Thus, this module is the primary runtime bottleneck and the main target for subsequent performance optimization.

3.5.5 Quantization algorithm analysis

RKNN-Toolkit2 supports three quantization algorithms: normal (min–max quantization), KL-Divergence, and MMSE (minimum mean square error) quantization. To select an appropriate scheme, we conduct ablation experiments, as summarized in Table 20.

**Table 20 Ablation of quantization algorithms under hybrid quantization**

| Algorithm | $\delta_1$ | Abs Rel | Model size/MB | Time/min |
|---|---|---|---|---|
| Normal | 0.888 | 0.110 | 5.2 | 9 |
| KL-Divergence | 0.891 | 0.106 | 5.2 | 13 |
| MMSE | 0.891 | 0.106 | 5.2 | 367 |

Under hybrid quantization, KL-Divergence and MMSE achieve the best accuracy ($\delta_1$=0.891, Abs Rel =0.106), while normal is slightly worse. However, MMSE relies on brute-force iterative search for an approximate optimum, resulting in a long quantization time (367 min) and making it impractical for engineering iteration. In contrast, KL-Divergence requires only 13 min and offers the best cost–performance. Therefore, we adopted KL-Divergence as the quantization algorithm for our model.

## 4 Conclusions

This paper presents a lightweight self-supervised monocular depth estimation network for real-time edge deployment. By introducing the DB-LKA module in the encoder, the model achieved low-overhead long-range context modeling while preserving local details. In the decoder, an adjacent-scale sparse fusion strategy was combined with the proposed DDF module to enable adaptive cross-scale interaction, improving detail recovery and geometric consistency. Furthermore, the robustness of self-supervised training was enhanced by incorporating a temporal soft fusion reprojection loss. Experiments show that the model attains Abs Rel=0.105 and $\delta_1$=0.892 on KITTI, and Abs Rel=0.308 in zero-shot evaluation on Make3D, under a budget of 3.0 M parameters and 3.5 GFLOPs. Moreover, on an RK3588S platform, a hybrid-quantized multi-thread implementation reaches 67 frames/s, demonstrating a strong balance among accuracy, throughput, and resource consumption. Future work will further improve cross-domain generalization and deployment on other edge devices for real-world applications.

## Acknowledgement

This work was supported by the National Natural Science Foundation of China General Program (No. 52475575) and the 2024 Shanxi Provincial Basic Research Program (No.202403021211076).

## Declaration of conflicting interests

The authors have no conflicts of interest related to this publication.

## References

[1] LI L Q, CHANG M, HOU X Y, et al. CEP-SLAM based on RGB-D data coupling errors processing. Journal of North University of China (Natural Science Edition), 2024, 45(5): 614-627.

[2] EIGEN D, PUHRSCH C, FERGUS R. Depth map prediction from a single image using a multi-scale deep network//Advances in Neural Information Processing Systems 27 (NIPS 2014), Montréal, QC, Canada, 2014: 2366-2374.

[3] LAINA I, RUPPRECHT C, BELAGIANNIS V, et al. Deeper depth prediction with fully convolutional residual networks//2016 Fourth International Conference on 3D Vision, October 25-28, 2016, Stanford, CA, USA. New York: IEEE, 2016: 239-248.

[4] WOFK D, MA F C, YANG T J, et al. FastDepth: fast monocular depth estimation on embedded systems//2019 International Conference on Robotics and Automation, May 20-24, 2019. Montreal, QC, Canada. New York: IEEE, 2019: 6101-6108.

[5] PAPA L, ALATI E, RUSSO P, et al. SPEED: separable pyramidal pooling EncodEr-decoder for real-time monocular depth estimation on low-resource settings. IEEE Access, 2022, 10: 44881-44890.

[6] GARG R, B G V K, CARNEIRO G, et al. Unsupervised CNN for single view depth estimation: geometry to the rescue. Computer Vision-ECCV 2016. Cham: Springer, 2016: 740-756.

[7] GODARD C, MAC AODHA O, BROSTOW G J. Unsupervised monocular depth estimation with left-right consistency//2017 IEEE Conference on Computer Vision and Pattern Recognition, July 21-26, 2017, Honolulu, HI, USA. New York: IEEE, 2017: 6602-6611.

[8] ZHOU T H, BROWN M, SNAVELY N, et al. Unsupervised learning of depth and ego-motion from video//2017 IEEE Conference on Computer Vision and Pattern Recognition, July 21-26, 2017, Honolulu, HI, USA. New York: IEEE, 2017: 6612-6619.

[9] GODARD C, MAC AODHA O, FIRMAN M, et al. Digging into self-supervised monocular depth estimation//2019 IEEE/CVF International Conference on Computer Vision, October 27-November 2, 2019. Seoul, Korea. New York: IEEE, 2019: 3827-3837.

[10] GUIZILINI V, AMBRUS R, PILLAI S, et al. 3D packing for self-supervised monocular depth estimation//2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, June 13-19, 2020. Seattle, WA, USA. New York: IEEE, 2020: 2482-2491.

[11] WATSON J, MAC AODHA O, PRISACARIU V, et al. The temporal opportunist: self-supervised multi-frame monocular depth//2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition, June 20-25, 2021. Nashville, TN, USA. New York: IEEE, 2021: 1164-1174.

[12] ZHOU Z K, FAN X N, SHI P F, et al. R-MSFM: recurrent multi-scale feature modulation for monocular depth estimating//2021 IEEE/CVF International Conference on Computer Vision, October 10-17, 2021, Montreal, QC, Canada. New York: IEEE, 2022: 12757-12766.

[13] LÜ X Y, LIU L, WANG M M, et al. HR-depth: high resolution self-supervised monocular depth estimation. Proceedings of the AAAI Conference on Artificial Intelligence, 2021, 35(3): 2294-2301.

[14] ZHAO C Q, ZHANG Y M, POGGI M, et al. MonoViT: self-supervised monocular depth estimation with a vision transformer//2022 International Conference on 3D Vision, September 12-16, 2022, Prague, Czech Republic. New York: IEEE, 2023: 668-678.

[15] BAE J, MOON S, IM S. Deep digging into the generalization of self-supervised monocular depth estimation. 2022: arXiv: 2205. 11083.

[16] ZHANG N, NEX F, VOSSELMAN G, et al. Lite-mono:

a lightweight CNN and transformer architecture for self-supervised monocular depth estimation//2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition, June 17-24, 2023, Vancouver, BC, Canada. New York: IEEE, 2023: 18537-18546.

[17] ZHAO B W, HE H D, XU H, et al. RTIA-Mono: Real-time lightweight self-supervised monocular depth estimation with global-local information aggregation. Digital Signal Processing, 2025, 156: 104769.

[18] EL-NOUBY A, TOUVRON H, CARON M, et al. XCiT: cross-covariance image transformers. 2021: arXiv: 2106.09681.

[19] FENG C, ZHANG C X, CHEN Z, et al. Real-time monocular depth estimation on embedded systems//2024 IEEE International Conference on Image Processing, October 27-30, 2024, Abu Dhabi, United Arab Emirates. New York: IEEE, 2024: 3464-3470.

[20] ZHANG G W, TANG X C, WANG L, et al. Repmono: a lightweight self-supervised monocular depth estimation architecture for high-speed inference. Complex & Intelligent Systems, 2024, 10(6): 7927-7941.

[21] ZHAO B W, HE H D, XU H, et al. LDA-Mono: a lightweight dual aggregation network for self-supervised monocular depth estimation. Knowledge-Based Systems, 2024, 304: 112552.

[22] ZHANG L, LYU C Z, WANG P, et al. LEDepth: a lightweight self-supervised monocular depth estimation network combining CNN and transformer. Signal, Image and Video Processing, 2025, 19(14): 1208.

[23] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need//Advances in Neural Information Processing Systems 30 (NeurIPS 2017), LongBeach, CA, USA, 2017: 5998-6008.

[24] GUO M H, LU C Z, LIU Z N, et al. Visual attention network. Computational Visual Media, 2023, 9(4): 733-752.

[25] WANG B Y, WANG S, YE D, et al. Deep neighbor layer aggregation for lightweight self-supervised monocular depth estimation//ICASSP 2024 IEEE International Conference on Acoustics, Speech and Signal Processing, April 14-19, 2024, Seoul, Korea, Republic of. New York: IEEE, 2024: 4405-4409.

[26] HOWARD A G, ZHU M L, CHEN B, et al. MobileNets: efficient convolutional neural networks for mobile vision applications. 2017: arXiv: 1704.04861.

[27] HE K M, ZHANG X Y, REN S Q, et al. Deep residual learning for image recognition//2016 IEEE Conference on Computer Vision and Pattern Recognition, June 27-30, 2016, Las Vegas, NV, USA. New York: IEEE, 2016: 770-778.

[28] WANG Z, BOVIK A C, SHEIKH H R, et al. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing, 2004, 13(4): 600-612.

[29] GEIGER A, LENZ P, STILLER C, et al. Vision meets robotics: The KITTI dataset. International Journal of Robotics Research, 2013, 32(11): 1231-1237.

[30] SAXENA A, SUN M, NG A Y. Make3D: learning 3D scene structure from a single still image//AAAI Conference on Artificial Intelligence (AAAI), Chicago, IL, USA, 2008: 1571-1576.

# 实时轻量化自监督单目深度估计

杨天祥[1], 孟令军[1*], 靳 鸿[2], 冯文杰[1], 刘昕昊[3]

1. 中北大学 仪器与电子学院, 山西 太原 030051;
2. 中北大学 电气与控制工程学院, 山西 太原 030051;
3. 上海大学 机电工程与自动化学院, 上海 201900

**摘 要：** 单目深度估计旨在从单张RGB图像预测场景中的深度信息，但现有的大多数先进单目深度估计模型计算开销大、难以在资源受限设备上实现实时推理。本文提出一种面向实时应用的轻量自监督单目深度估计网络。编码端通过可分解大核卷积与局部深度卷积协同建模，在低计算开销下同时捕获长程上下文与局部细节；解码端利用跨尺度特征差分作为引导信息实现多尺度特征之间的动态融合，从而在轻量约束下提升细节恢复与几何一致性。此外，引入时序软融合重投影损失，充分利用前后帧的互补信息，增强自监督训练的鲁棒性。模型参数量为3.0 M，计算量为3.5 GFLOPs；在KITTI数据集上取得Abs Rel=0.105，$\delta_1$=0.892，在Make3D数据集零样本测试中取得Abs Rel=0.308。在RK3588S平台上使用混合量化模型进行多线程推理可达67 frames/s。实验结果表明，所提方法在端侧实现了良好的精度-效率平衡，适用于实时单目深度估计任务。

**关键词：** 单目深度估计；深度学习；自监督学习；大核注意力；差分驱动动态融合；轻量网络；RK3588S

**引用格式：** YANG Tianxiang, MENG Lingjun, JIN Hong, et al. Real-time lightweight self-supervised monocular depth estimation. Journal of Measurement Science and Instrumentation, 2026, 17(2): 278-296. DOI: 10.62756/jmsi.1674-8042.2026024