

# Fast-FoundationStereo: Real-Time Zero-Shot Stereo Matching

## Supplementary Material

### 6. Real-time Demo

Please watch our supplemental video which demonstrates real-time demos running on GPU 3090, using the model evaluated in Table 1 (main paper).

### 7. More Details on Cost Filtering

We now elaborate the search space for blockwise neural architecture search for cost filtering module. Within 3D hourglass, the following layer options are considered:

- **3D conv layer.** Output channel dimensions are set to  $0.5\times$ ,  $1\times$  or  $2\times$  of the input channel dimension. Kernel size is set to 3. Stride is set to 2 unless downsampling is performed where it is set to 1. Batch normalization and activation operations are optionally included.
- **3D deconv layer.** This layer is chosen when upsampling the spatial dimensions of the cost volume. The parameters are set following the original counterparts in FoundationStereo [68].
- **APC layer.** Output channel dimensions are set to  $0.5\times$  or  $1\times$  of the input channel dimension. Kernel size of axial-convolution is chosen from 3, 9, 17. Kernel size of planar-convolution is set to 3. Stride is set to 1.
- **Residually connected 3D conv layers.** We follow the Basic Block structure in ResNet [23], where two convolutional layers of kernel size 3 are residually connected. Output channel dimensions are set to  $0.5\times$  or  $1\times$  of the input channel dimension.
- **Feature guided volume excitation.** The multi-level unary features for the left image  $f_l^{(i)} \in \mathbb{R}^{C_i \times \frac{H}{i} \times \frac{W}{i}}$ ,  $i \in \{4, 8, 16, 32\}$  are provided as guidance to excite the relevant geometric features in the cost volume [1].

Within the Disparity Transformer, the self-attention transformer encoder layer repeats from 1 to 6 times. The hidden dimension of feedforward layer is chosen between  $2\times$  or  $4\times$  of the input feature dimension. The number of heads is set to 2 or 4.

In each block the number of layers are chosen to be no more than the number of layers in original teacher’s counterpart. In total we divided cost filtering module into  $N = 8$  blocks. By assembling the different blocks’ candidates, we obtain  $5.5 \times 10^{24}$  number of cost filtering module designs, where one of them is the original cost filtering module from teacher model. By considering only the design choices faster than teacher, we obtain  $5.8 \times 10^{19}$  possible combinations. Nevertheless, with our introduced blockwise distillation (Sec. 3.2 in main paper), only 2584 blocks need to be trained, which can be performed efficiently in terms of both time and memory, allowing ease of parallelization. The

whole blockwise distillation process takes 14 days in total distributed on 128 NVIDIA A100 GPUs. After distillation, the most promising block combinations are identified via ILP which can be efficiently solved under a second. For the model evaluated in Table 1 (main paper), the latency budget was set to  $\Delta\tau = -0.04s$ .

**Cost Filtering Pruning Study.** As an alternative option to our introduced blockwise architecture search, we also experimented with directly applying structured pruning to the cost filtering module. As shown in Fig. 11, such strategy leads to marginal speed gain while substantially compromises the prediction accuracy. This degradation is likely attributable to the inherently small channel dimension (typically below 100) in cost volumes, which offers limited opportunities for effective pruning, in contrast to the significant redundancy in refinement module where structured pruning is more effective.

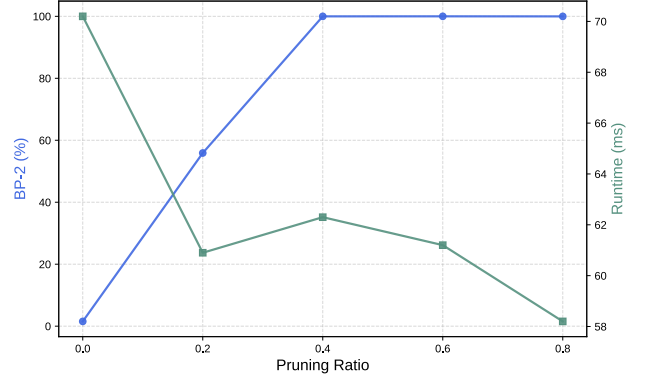


Figure 11. Study of applying structured pruning to cost filtering module, which is an alternative strategy to our introduced blockwise architecture search.

### 8. Effects of Refinement Iterations

Fig. 12 shows the effects of refinement iterations on accuracy and runtime of the complete model, where the results are evaluated on Middlebury-Q dataset. We compare two different versions of refinement module pruned with ratios of 0.6 and 0.8, while the remaining parts of the network (e.g. feature extraction and cost filtering) are the same. As can be observed, under pruning ratio 0.8, the accuracy only obtains marginal improvements with increasing iterations steps, implying the aggressively pruned refinement module has less capacity to benefit from iterative refinement. Under pruning ratio 0.6, the accuracy saturates around 8 iterations. In terms of runtime, when the iteration steps are small, different pruning ratios lead to marginal differences, falling within the magnitude of milliseconds and introducing mea-

surement noise. However, as the refinement iterations increase, higher pruning ratio yields more runtime benefit due to the cumulated effect. For our model evaluated in Table 1 (main paper) pruning ratio is set to 0.6.

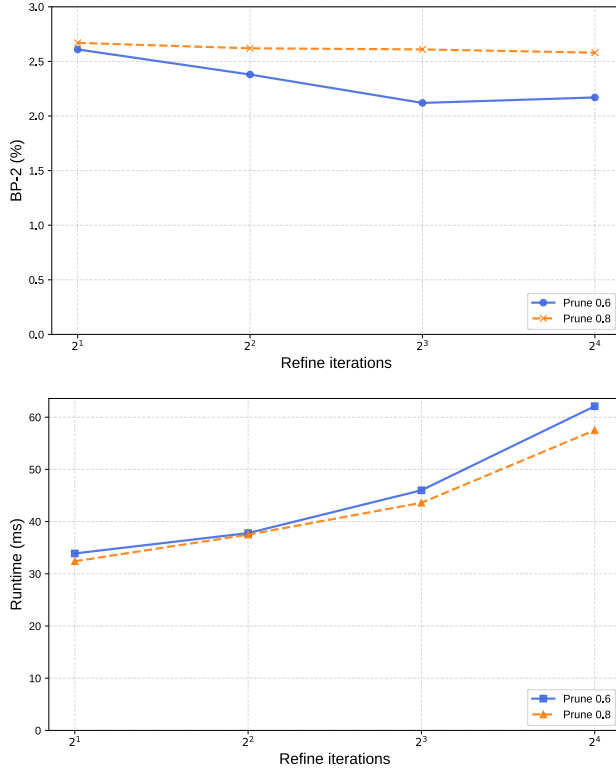


Figure 12. Effects of refinement iterations on accuracy (top) and runtime (bottom) under two different pruning ratios.

## 9. More details on Model Efficiency

Table 5 shows more details of model efficiency analysis including runtime profiled on different NVIDIA GPUs. Our model obtains dramatic reduction in parameters, MACs (Multiply-Accumulate Operations) and runtime across different hardware.

Method	Runtime (ms)			#Param (M)	MACs (G)
	3090	4090	A100		
FoundationStereo [68]	496	295	308	374.5	5413.9
Ours	49	30	41	14.6	309.9

Table 5. Model efficiency analysis including runtime profiled on different NVIDIA GPUs. Ours corresponds to the model evaluated in Table 1 (main paper).

## 10. Efficient GWC Volume Construction

Group-wise correlation (GWC) volume has been widely used in prior works [1, 59, 68, 72–74] to construct discriminative matching cost representations. Its original implementation [21] constructs the GWC volume by iterating

over each disparity level in a Python for-loop: at each disparity  $d$ , the reference feature map is shifted relative to the target by  $d$  pixels along the width axis, and a group-wise normalized dot product is computed between the aligned features, writing one disparity slice of the 5-D volume  $\mathcal{V} \in \mathbb{R}^{B \times G \times D \times H \times W}$  per iteration. This loop incurs significant overhead from repeated GPU kernel launches (one per disparity level) and prevents the compiler from fusing operations across disparity. Our optimized pytorch variant eliminates this explicit loop entirely by first left-padding the target feature map by  $D-1$  pixels and then unfold along the width dimension to extract all  $D$  shifted views as a single strided tensor—a zero-copy operation that creates sliding windows over the padded data without materializing new memory. After a flip and permutation to align the disparity ordering, both the reference (broadcast-expanded via unsqueeze) and the unfolded target volume are reshaped into groups of  $C/G$  channels, and correlated via a single fused element-wise multiply-and-sum. When combined with compilation of Pytorch or TensorRT, this formulation allows the compiler to trace and fuse the entire volume construction and correlation into a minimal number of GPU kernels, substantially reducing kernel launch overhead and improving memory access patterns compared to the original per-disparity-slice implementation. On image resolution of Middlebury-Q, we observe about  $6\times$  runtime reduction, and  $3\times$  memory usage reduction for constructing GWC volume. Our detailed implementation is available at: <https://github.com/NVlabs/Fast-FoundationStereo>

## 11. Model Parameter and Memory Usage

Table 6 details parameter count comparison. Our model’s peak memory usage is 0.63GB when running on Middlebury-Q. This fits into edge compute devices such as NVIDIA Jetson Orin or Thor, which are commonly used in real-time deployment for robotics and self-driving.

Model	BANet-3D	BANet-2D	RT-IGEV	IINet	LightStereo-L	Ours
Param (M)	3.63	5.46	4.17	19.56	24.29	<b>17.65</b>

Table 6. Model parameter comparison.

## 12. More Generalization Results

Fig. 13 and 14 demonstrate more qualitative results of zero-shot inference on out-of-domain stereo images featuring challenges such as textureless regions, translucent surfaces, specular highlights, diverse depth ranges, complex illuminations, varying viewing perspectives and indoor / outdoor scenarios.

### 13. More Details on Pseudo-labeling

Fig. 15 visualizes intermediate results in our pseudo-labeling process. Our data curation process can automatically discover failures on noisy internet data such as images containing subtitle, mosaic and overly challenging samples that are unsuitable for training. The final pseudo-labels can also correct erroneous predictions from FoundationStereo on sky regions. Samples with positive pixels occupying more than 60% (excluding sky regions) on the consistency mask are kept for training, which results in 1.4M stereo pairs in total.

### 14. Limitations

While our method achieves strong generalization, it inevitably inherits certain limitations from its teacher FoundationStereo. Specifically, performance on translucent surfaces remains a challenge (Table 2 in main paper), which can be mitigated by incorporating training datasets enriched with relevant objects.



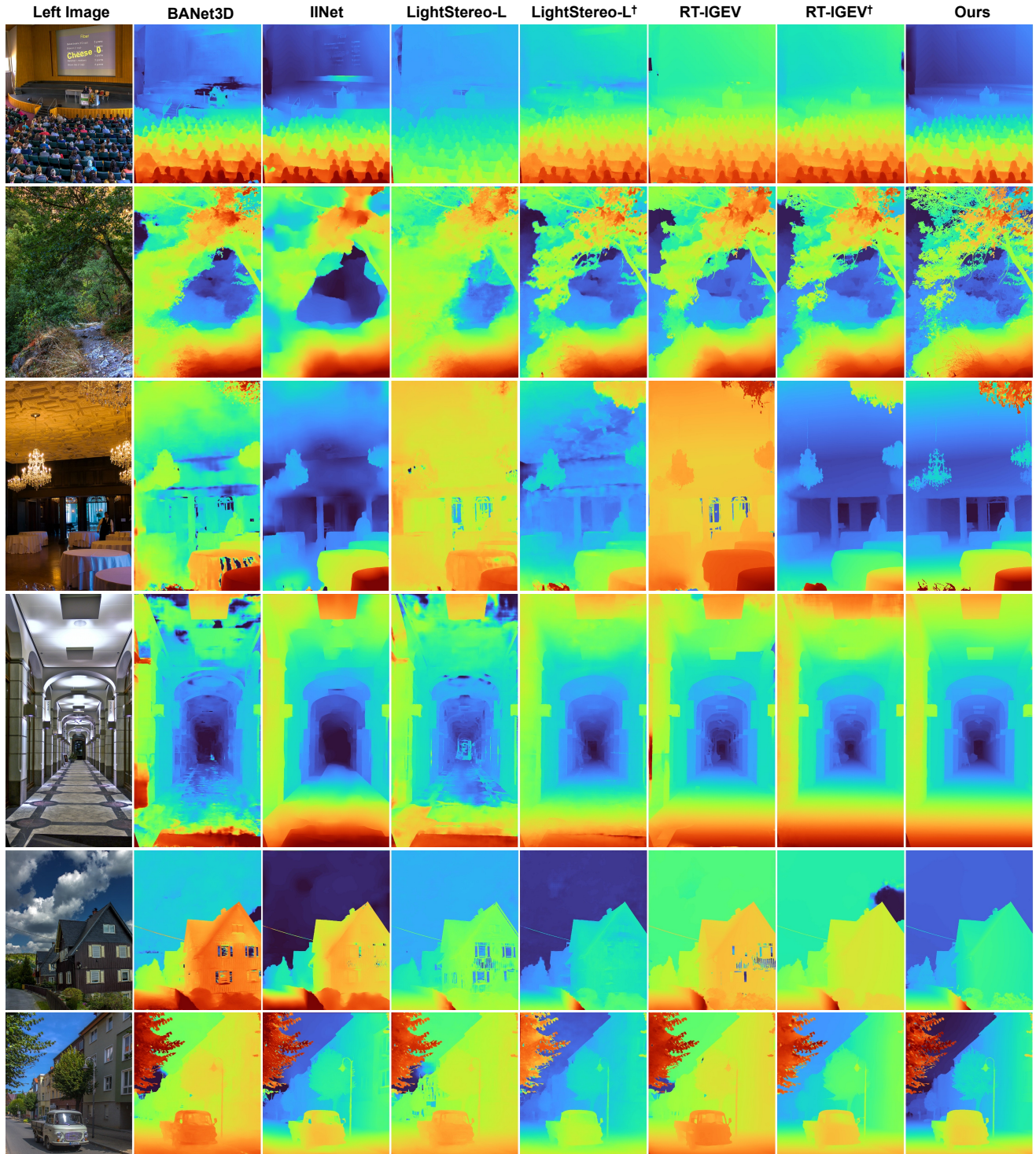


Figure 13. Qualitative comparison among real-time methods. Results are obtained by zero-shot inference without training on target domain [67]. <sup>†</sup>Denotes training on the exact same datasets as ours (including our proposed pseudo-labels). Our pseudo-labeled internet data consistently enhances the generalization across different methods. However, our model demonstrates strongest robustness, validating the effectiveness of both our model design and pseudo-labeling. (Zoom-in on a digital device for better visualization.)



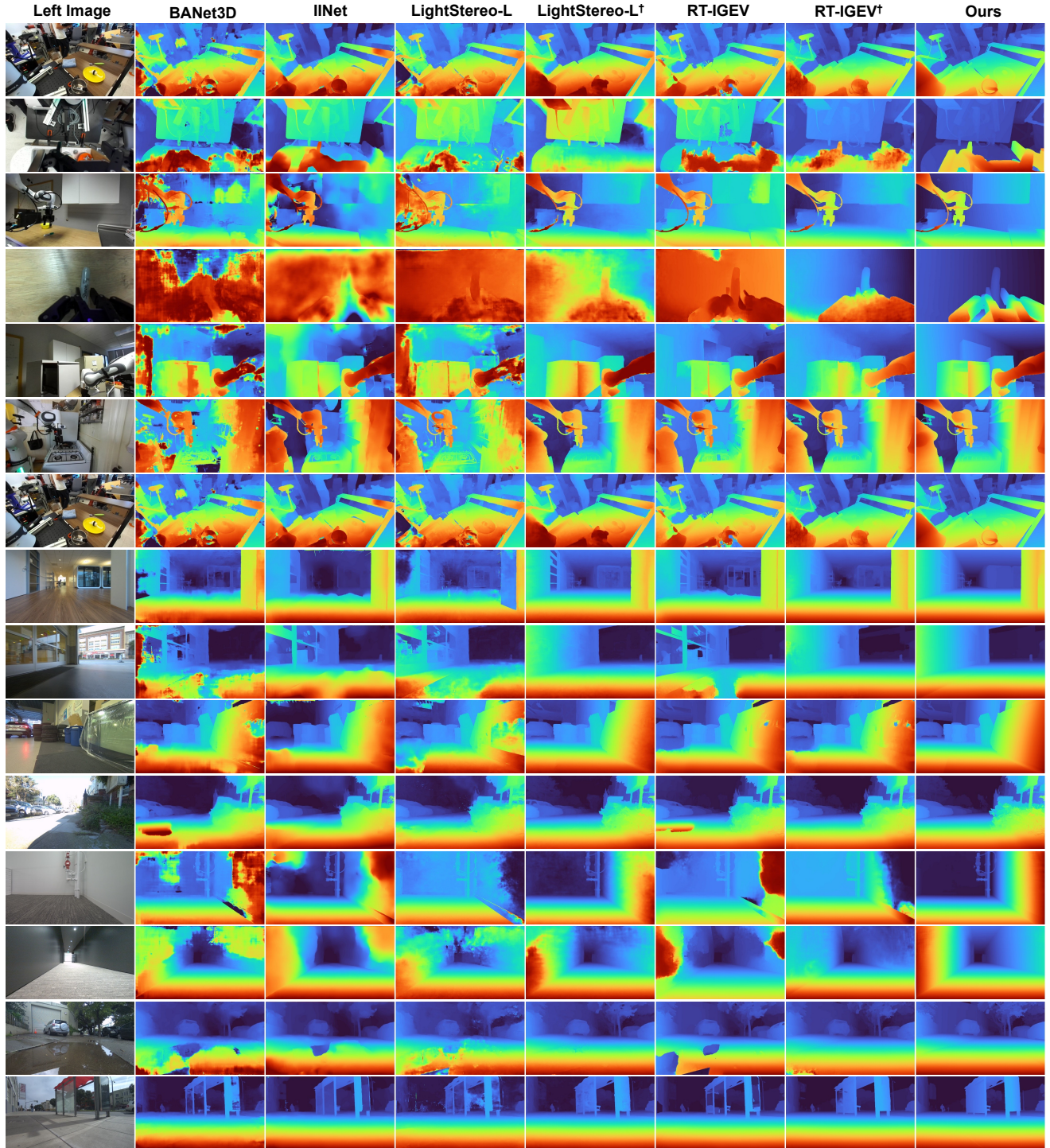


Figure 14. Qualitative comparison among real-time methods. Results are obtained by zero-shot inference without training on target domain (images are from [28] or captured in-the-wild). <sup>†</sup>Denotes training on the exact same datasets as ours (including our proposed pseudo-labels). Our pseudo-labeled internet data consistently enhances the generalization across different methods. However, our model demonstrates strongest robustness, validating the effectiveness of both our model design and pseudo-labeling. (Zoom-in on a digital device for better visualization.)



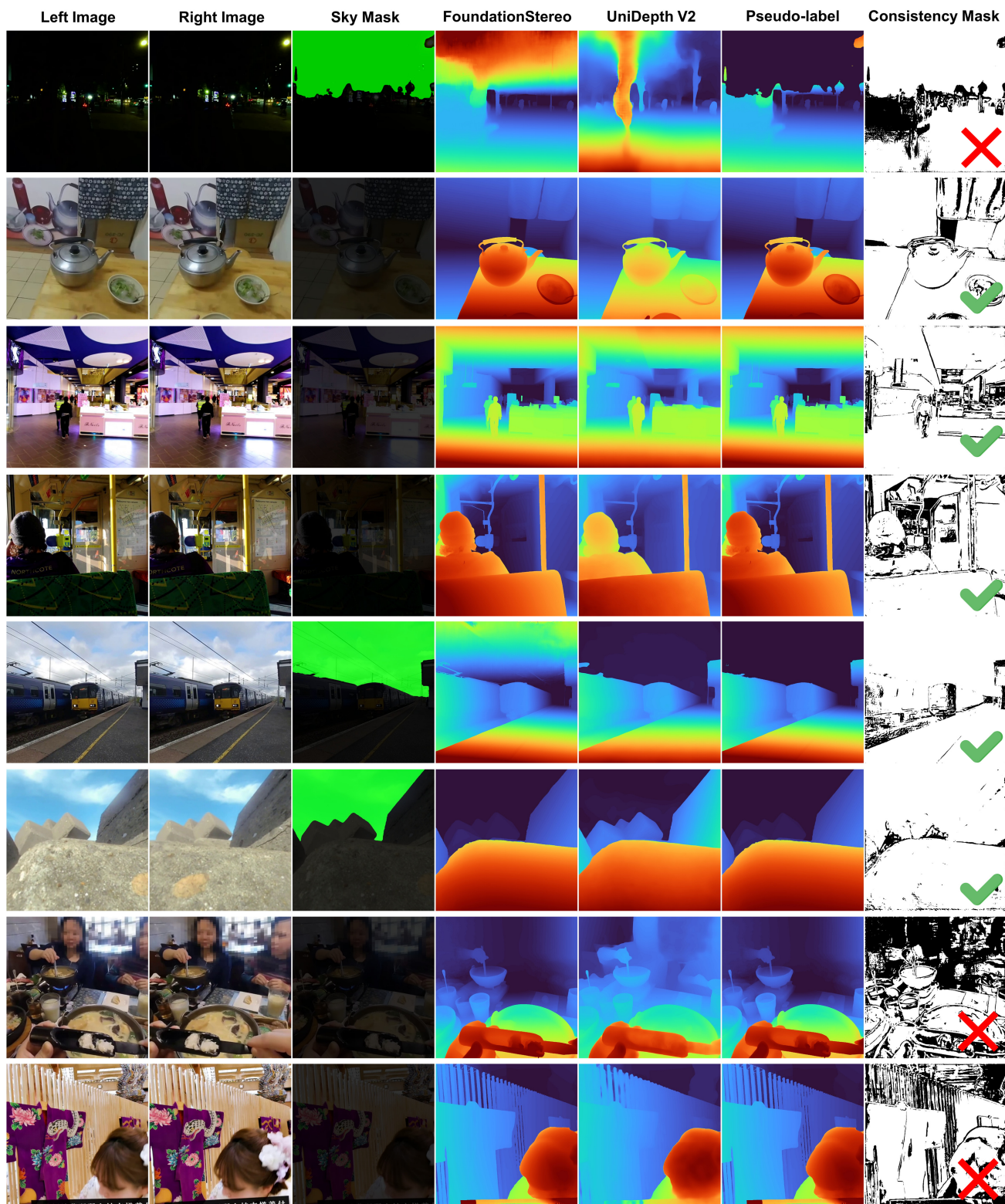


Figure 15. Visualizations of the intermediate results in our pseudo-labeling process. In the rightmost column, ✓ or ✗ denotes whether samples are kept for training or not, based on the percentage of positive pixels in the consistency mask. Our data curation process can automatically discover failures on noisy internet data such as images containing subtitle (bottom), mosaic (2nd last row) and overly challenging samples that are unsuitable for training (top). The final pseudo-labels can also correct erroneous predictions from FoundationStereo on sky regions (5th row). (Zoom-in on a digital device for better visualization.)