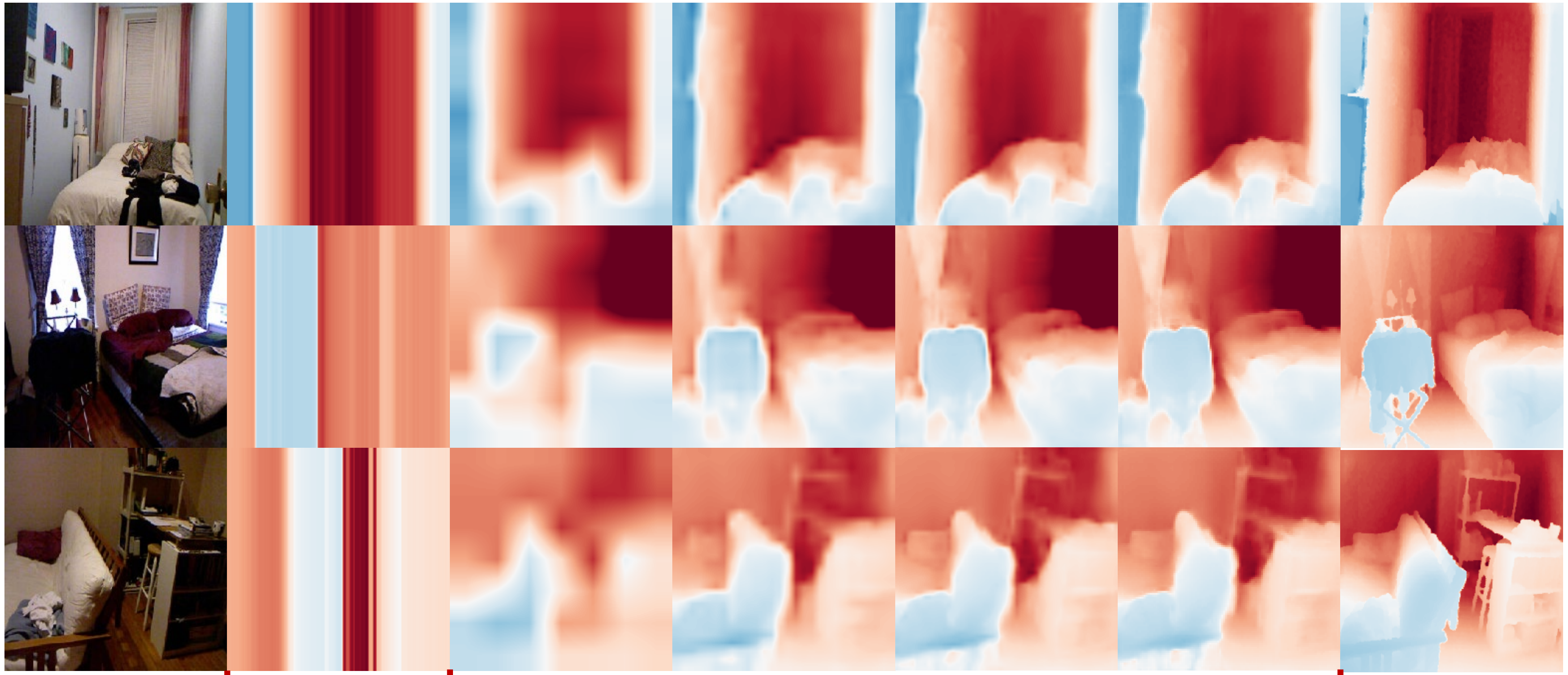
Supplementary Material

ANYTIME DEPTH ESTIMATION WITH LIMITED SENSING AND COMPUTATION CAPABILITIES ON MOBILE DEVICES

S.1

Visualized Prediction Examples

Extending **Figure 4** with additional examples



RGB Image

0.0ms Interpolation

6.4ms

8.0ms

9.8ms

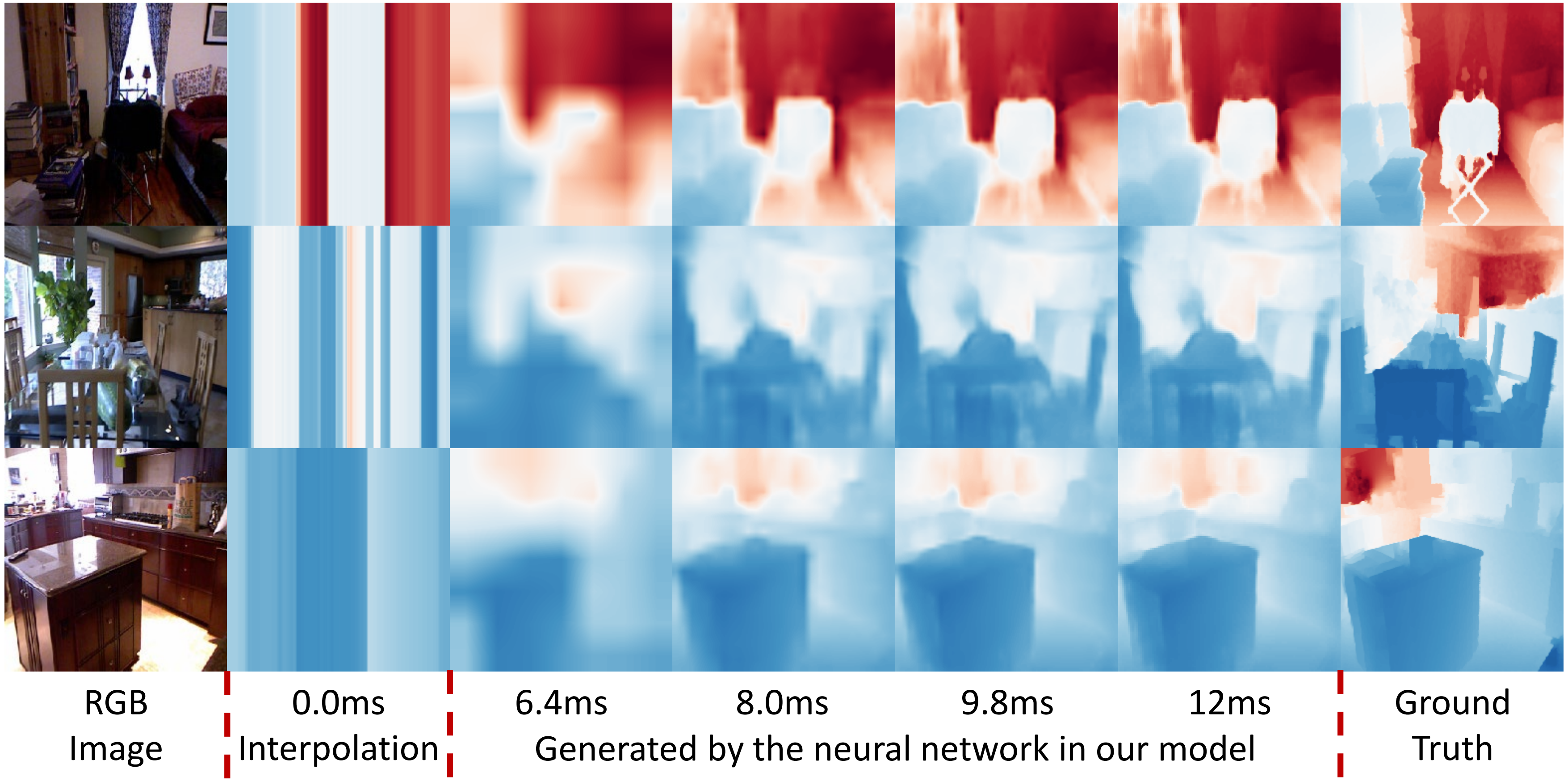
12ms

Generated by the neural network in our model

Ground Truth

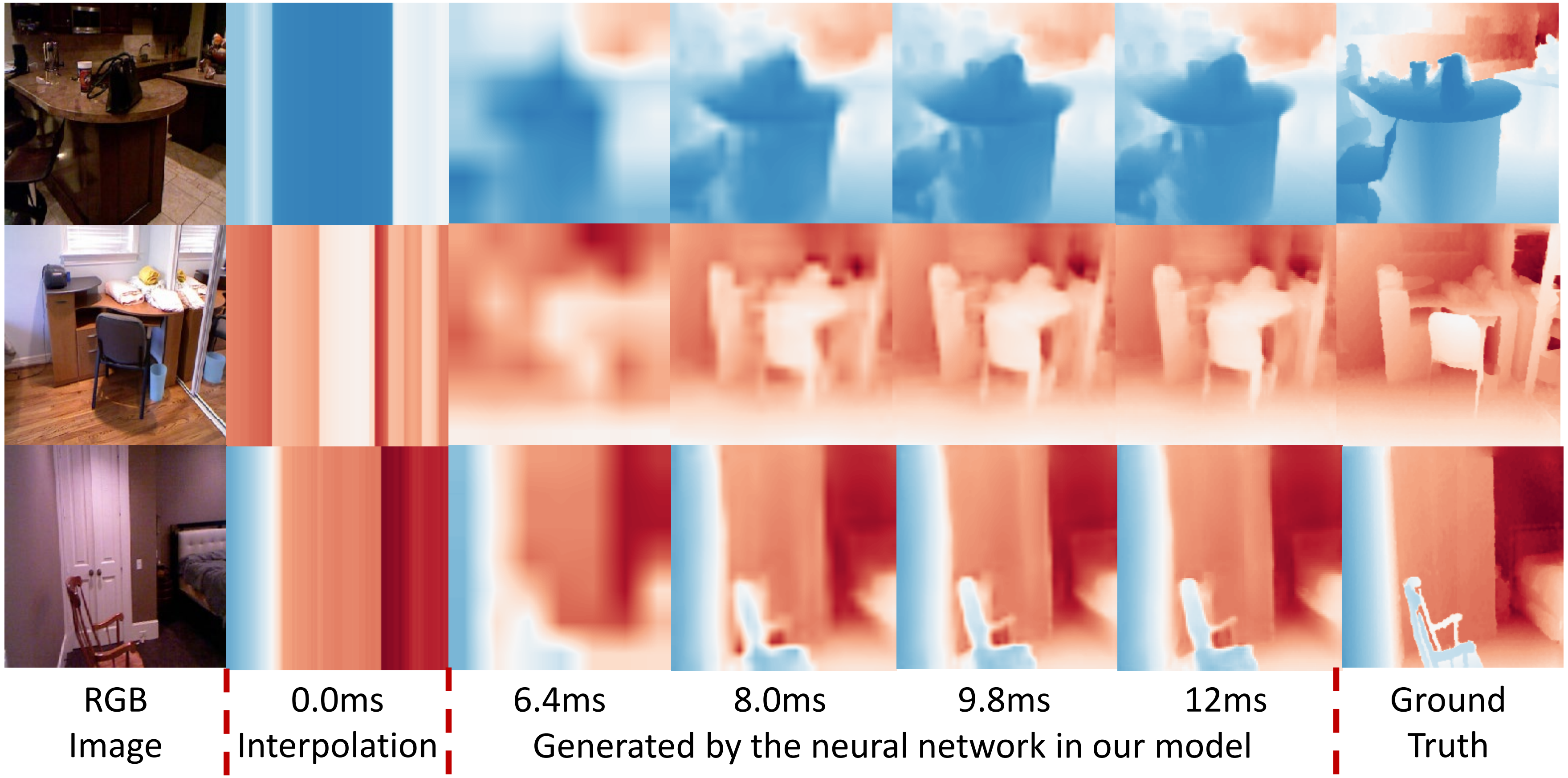
S.1 (cont'd)
Visualized Prediction Examples

Extending **Figure 4** with additional examples



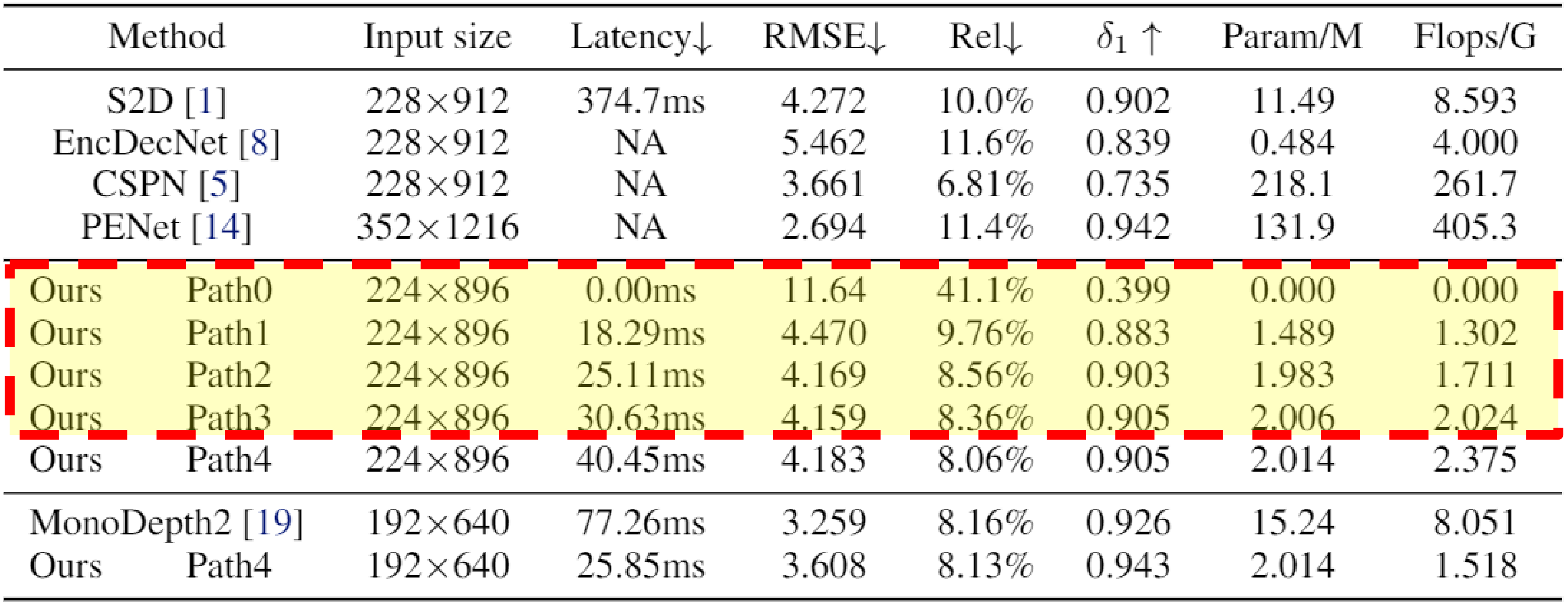
S.1 (cont'd)
Visualized Prediction Examples

Extending **Figure 4** with additional examples



S.2 Complete Evaluation Results for KITTI Odometry Dataset [Table 3]

Path 2 – Path 4 show similar accuracy scores. Considering that most of prior methods use a smaller model on KITTI dataset, e.g., S2D use ResNet-18 based model for KITTI dataset while use ResNet-50 based model for NYU Depth V2, one possible reason is that our model for Path 2 is large enough for the KITTI Odometry Dataset and model for Path 4 is too large for it.



Supervised

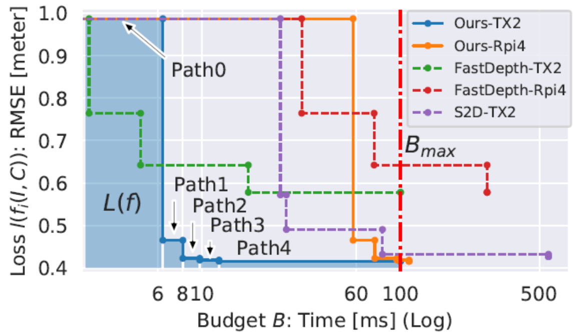
Self Supervised

Method	Input size	Latency↓	RMSE↓	Rel↓	δ_1 ↑	Param/M	Flops/G
S2D [1]	228×912	374.7ms	4.272	10.0%	0.902	11.49	8.593
EncDecNet [8]	228×912	NA	5.462	11.6%	0.839	0.484	4.000
CSPN [5]	228×912	NA	3.661	6.81%	0.735	218.1	261.7
PENet [14]	352×1216	NA	2.694	11.4%	0.942	131.9	405.3
Ours Path0	224×896	0.00ms	11.64	41.1%	0.399	0.000	0.000
Ours Path1	224×896	18.29ms	4.470	9.76%	0.883	1.489	1.302
Ours Path2	224×896	25.11ms	4.169	8.56%	0.903	1.983	1.711
Ours Path3	224×896	30.63ms	4.159	8.36%	0.905	2.006	2.024
Ours Path4	224×896	40.45ms	4.183	8.06%	0.905	2.014	2.375
MonoDepth2 [19]	192×640	77.26ms	3.259	8.16%	0.926	15.24	8.051
Ours Path4	192×640	25.85ms	3.608	8.13%	0.943	2.014	1.518

Table shows that our approach uses a model which is precise, yet small and low-power enough to run on real-time embedded devices.

Extending **Table 3** with additional results (Highlighted)

S.3
Quantitative Results for Figure 5



“RPi4” stands for Raspberry Pi4.

Considering that S2D is too slow and energy-consuming even on TX2 system, we didn’t deploy it on Raspberry Pi4.

Extending Figure 5

New results, not shown in Paper.

Input Size	RMSE	TX2 Latency	TX2 Energy
28 × 38	0.5941	24.87ms	0.269J
56 × 76	0.5004	26.65ms	0.302J
112 × 152	0.4320	81.12ms	0.954J
228 × 304	0.4265	554.95ms	6.244J

S2D

Input Size	RMSE	TX2 Latency	TX2 Energy	RPi4 Latency	RPi4 Energy
56 × 56	0.7643	2.71ms	0.171J	31.81ms	0.052J
112 × 112	0.6422	4.92ms	0.220J	73.63ms	0.247J
224 × 224	0.5780	17.14ms	0.236J	272.65ms	1.09J

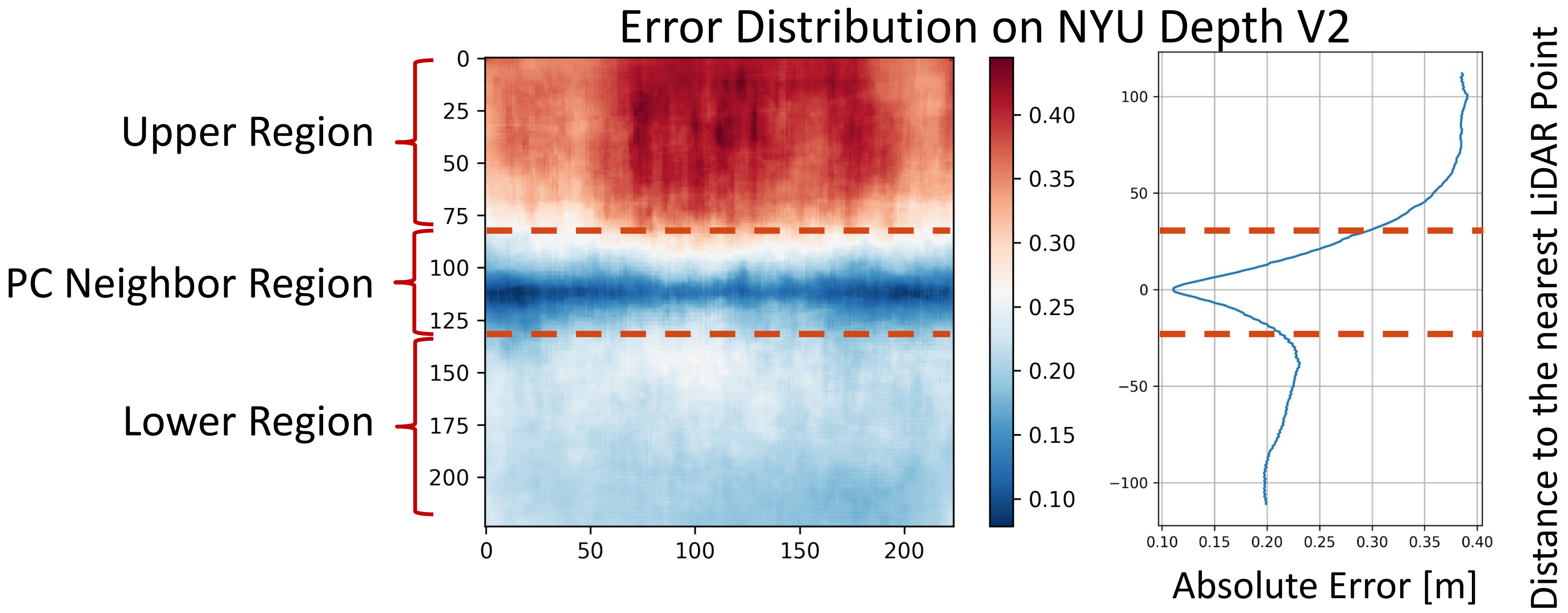
FastDepth

Output Path	Input Size	RMSE	TX2 Latency	TX2 Energy	RPi4 Latency	RPi4 Energy
Path 1	224 × 224	0.4651	6.37ms	0.207J	57.72ms	0.137J
Path 2	224 × 224	0.4225	8.03ms	0.213J	74.23ms	0.222J
Path 3	224 × 224	0.4184	9.77ms	0.263J	96.04ms	0.323J
Path 4	224 × 224	0.4156	12.21ms	0.313J	110.42ms	0.402J

Ours

S.4

Error Distribution of Depth Map



Extending
Section 5 with

additional
experiments
not shown
in Paper

Generally the error map is partitioned into upper, lower and PC neighbor region. Pixels in the PC neighbor region is close to LiDAR points, so that they have the smallest error, which means that our ECU module effectively aligns image features to PC feature. In NYU Depth V2 dataset, lower region mostly covers the ground, which is close to the camera, so the error is smaller comparing with the upper region.

S.5

Power Consumption of 2D and 3D LiDAR

Here we list the power consumption of some commercial 2D and 3D LiDARs.

2D LIDAR

RPLiDAR A2: 2.25W-3W

RPLiDAR A1: 0.5W

YDLiDAR X2: 2W

YDLiDAR G6: 2W-2.4W

CA113: 2W

3D LIDAR

HDL 64E: 60W (Used by the Kitti dataset)

HDL 32E: 12W

OS1-32: 14-20W

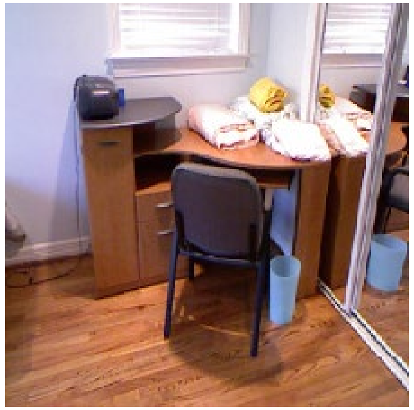
Puck: 8W

Clarification for **Section 1** (Introduction)
Concrete power values for commercial 2D and 3D LiDAR (all links are clickable)

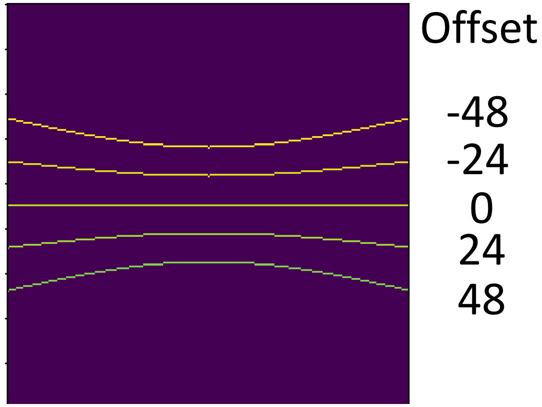
S.6
Accuracy under Different Camera-LiDAR Positions

Upper Region
Tables, Etc.
More Objects

Lower Region
Ground
Fewer Objects



Typical Scene



Projected Positions of 2D Point Cloud

Offset	Path 1	Path 2	Path 3	Path 4
-48	0.4174	0.3702	0.3650	**0.3655**
-24	0.4415	0.3968	0.3919	0.3900
0	0.4651	0.4225	0.4184	**0.4156**
24	0.4846	0.4462	0.4436	0.4420
48	0.5113	0.4758	0.4728	0.4713

RMSE under Different Camera-LiDAR Positions

We evaluate our model under different camera-LiDAR positions. Results with 0 offset are reported in paper. As the projected position of PC moves upward, the accuracy improves significantly. This is because the lower region typically corresponds to the ground, while the upper region corresponds to the objects in the scene, e.g., tables, walls, etc. Thus, the PC in the upper region may bring more information about the environment to the model, which translates into a better prediction accuracy.

Additional Experiments (not shown in Paper)

S.7

Clarifications about the “Budget” Concept

Available Budget vs Required Budget

- All budget references in the main text refer to the *available* budget, e.g., how much computation can we afford for inference. This is determined by the environment and not controllable by the model. We model this budget as a random variable.
- A similar concept is the *required* budget, e.g., how much computation do we actually need for inference. This can be tuned by designing a more efficient model.

The Minimum Possible Budget – Zero

- Zero available computation budget is possible because the preprocessing steps (like image resizing, point cloud transforming) are not included in the model; these preprocessing steps may use up all the computation budget when the system gets busy.
- This is why we need to consider the $[0, B_{max}]$ interval with the minimum possible budget zero.

S.7 (cont’d)
Clarifications about the “Budget” Concept

Fixed Budget vs Random Budget

- Take the time budget as an example. One may think that as long as the required run time of the model is smaller than a fixed time budget, the model is good to be deployed on device. However, this is not true. For example, in the video experiment we provide in supplementary material, though the vanilla model requires only 12ms for inference, it fails to make any prediction in 40ms.
- This is because when one talks about the “fixed time budget”, one may refer to the “fixed wall-clock time” instead of the “real run time”; one of our main motivations for introducing the random budget and random halting is precisely to simulate the gap between the wall clock time and the real run time. The difference between wall clock time and actual run time can be huge.
- In general, one may never know for how much time the OS pauses the model inference to execute other tasks, or for how long the memory accesses stall, or how long the inter-core communication may take. As the system gets busy, a large portion of the time is spent on other things instead of running the inference; this is especially true for embedded systems that have very limited computation capabilities.

Thank you!