A. Appendix

A.1. Formulation of Diffusion Model

We define $\mathbf{x}_0$ as the desired refined sailency latent, and $\mathbf{x}_t$ as the saliency latent distribution by adding Gaussian noise distribution sequentially t times. Following the notion of (Ho et al., 2020; Dhariwal & Nichol, 2021; Nichol & Dhariwal, 2021), this is called the diffusion process. Thus, we could continuously add noise into original $\mathbf{x}_0$ through a Markov process sampling variables $\{\mathbf{x}_0, \mathbf{x}_1, ...\mathbf{x}_{t-1}, \mathbf{x}_t, ..., \mathbf{x}_T\}$ until $\mathbf{x}_T$ becomes a normal noise distribution $p(\mathbf{x}_T) \sim \mathcal{N}(\mathbf{x}_T; 0, I)$. Here, we call the $\mathbf{x}_T$ as the initialization of the Starting from a refined salient latent distribution $\mathbf{x}_0$, we define a forward Markovian noising process q as below. In particular, the added noise is scheduled by the variance $\beta_t(I)$:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) \coloneqq \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{11}$$

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) \coloneqq \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{12}$$

As noted by Ho (Ho et al., 2020) of the sampling properties, we can directly sample data $\mathbf{x}_t$ at an arbitrary timestep t without the need of applying q repeatedly:

$$q(\mathbf{x}_t|\mathbf{x}_0) \coloneqq \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{13}$$

$$\coloneqq \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \epsilon\sqrt{1-\bar{\alpha}_t}, \epsilon(I)\mathcal{N}(0, \mathbf{I}) \tag{14}$$

where $\bar{\alpha}_t \coloneqq \prod_{s=0}^{t} \alpha_s$ and $\alpha_t \coloneqq 1 - \beta_t$ are also a fixed variance coefficient schedule corresponding to β_t. Based on Bayes' theorem, it is found that the posterior $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is a Gaussian distribution as well:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I}) \tag{15}$$

where

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) \coloneqq \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \tag{16}$$

and

$$\tilde{\beta}_t \coloneqq \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{17}$$

are the mean and variance of this Gaussian distribution.

In practice, the representation of $\mathbf{x}_t$ could be obtained by extending the diffusion process defined in Eq. 14 as below.

$$\mathbf{x}_t = \bar{\alpha}_t\mathbf{x}_0 + \alpha_t\bar{\beta}_{t-1}\bar{\boldsymbol{\vartheta}}_{t-1} + \beta_t\boldsymbol{\vartheta}_t \tag{18}$$

where $\boldsymbol{\vartheta_t} \sim \mathcal{N}(0, (I))$ is a gaussian distribution that represents the stochastic property of the diffusion process. It also gives a description of how to represent the diffusion result in $\mathbf{x}_t$ by real sample $\mathbf{x}_0$ and given fixed variance scheduler α_t and β_t. We could get a sample from $q(\mathbf{x}_0)$ by first sampling from $q(\mathbf{x}_T)$ and running the reversing steps $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ until $\mathbf{x}_0$.

Besides, the distribution of $q(\mathbf{x}_T)$ is nearly an isotropic Gaussian distribution with a sufficiently large T and reasonable schedule of β_t ($\beta_t \to 0$), which making it trivial to sample $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$. Moreover, since calculating $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ exactly should depend on the entire data distribution, we could approximate $q_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ using a neural network posterior process, which is optimized to predict a mean μ_θ and a diagonal covariance matrix Σ_θ:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \coloneqq \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_\theta(\mathbf{x}_t, t)) \tag{19}$$

Instead of directly parameterizing $\mu_\theta(\mathbf{x}_t, t)$, Ho (Ho et al., 2020) found learning a network $f_\theta(\mathbf{x}_t, t)$ to predict the ϵ or $\mathbf{x}_0$ from Eq. 14 worked best. We choose to predict $\mathbf{x}_0$ in this work.

However, saliency denoising can't be performed without any given images. We utilize the visual condition c to guide the denoising process. As we assume above, the condition, $\mathbf{c}$ is entirely independent of the denoising and diffusion process. So we can easily change it into conditioned denoising by simply modifying the denoising process as below.

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}) \coloneqq \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t, \mathbf{c}), \sigma_\theta(\mathbf{x}_t, t)) \tag{20}$$

Moreover, since we desired a deterministic process, which means that given an image pairs, the generated saliency map is uniquely deterministic. This is a pre-request for accurate prediction tasks. In that case, we eliminate randomness according to DDIM (Song & Ermon, 2020). Where the random variance $\boldsymbol{\vartheta_t}$ is set to 0.

B. Extended Implementation details

The implementation and the pre-trained model will be open-source after acceptance. Here we release an anonymous link [1] for review. SimpleDiffusion is implemented with the Pytorch (Paszke et al., 2019) framework. We train the entire model with batch size 32 for 150 epochs iterations on a single node with NVIDIA A100 GPUs. We utilize the AdamW optimizer (Kingma & Ba, 2014) with (β_1, β_2, w) = (0.9, 0.999, 0.01), where w is the weight decay. The linear learning rate warm-up strategy is applied for the first 15% iterations. The cosine annealing learning rate strategy is adopted for the learning rate decay from the initial learning rate of $1e^{-3}$ to $1e^{-7}$. We use L1 and L2 pixel-wise loss at the first 50% training iterations as auxiliary subversion.

Augmentation We randomly crop a patch from the original image and its corresponding depth or thermal map and resize them to the desired input size. This helps to avoid overfitting and focus on learning to refine the details of different regions of the image. We randomly adjust the brightness, contrast, saturation, and hue of the input image. This helps to simulate different lighting conditions and make the network invariant with color changes. We randomly flip the input image and its corresponding depth or thermal map horizontally. This helps to augment the data with different orientations and symmetries. Random rotation: We randomly rotate the input image and its corresponding depth or thermal map by a small angle. This helps to augment the data with different angles and perspectives. Here, we use plus or minus 5 degrees as the rotation parameter.

Visual Condition: SimpleDiffusion is compatible with any backbone which could extract multi-scale features. Here, we respectively evaluate our model on the standard convolution-based ResNet (He et al., 2016) backbones and transformer-based PVT backbones. We employ hierarchical aggregation and heterogeneous interaction (HAHI (Li et al., 2022)) neck to enhance features between scales and multi-scale emphasis aggregation to aggregate features into corss-modal salient visual condition. The salient visual condition dimension is equal to the last layer of the neck.

Diffusiong Head: We use the improved sampling process (Song & Ermon, 2020) with 1000 diffusion steps for training and 10 inference steps for inference. The learning rate of the diffusion head is 10 times larger than the backbone parameters. The dimension d of the encoded saliency latent is 16 with shape $\frac{H}{2}, \frac{W}{2}, d$, we conduct detailed ablation to illustrate different inference settings.

B.1. Evaluation Metrics

Suppose the predicted and ground-truth saliency to be $\hat{I} \in \mathbb{R}^{m \times n}$ and $I_{gt} \in \mathbb{R}^{m \times n}$, respectively, and the number of valid pixels to be N. We follow the existing methods (Luo et al., 2024) and utilize the following measures for quantitative evaluation. We listed all potential metrics below: We follow the existing methods (Luo et al., 2024) and utilize the following measures for quantitative evaluation. We list all potential metrics below:

1.Structure Measure (S_m). The structure measure (S_m) evaluates the structural similarity between a predicted saliency map (P) and a ground-truth (G). It is defined as follows:

$$S_m = \alpha \cdot S_o + (1 - \alpha) \cdot S_r$$

where S_o measures the object-aware structural similarity and S_r measures the region-aware structural similarity. α balances the two terms, typically set to 0.5. This measure assesses how well the predicted saliency map preserves structural

[1] https://anonymous.4open.science/r/simple-diffusion

information from the ground-truth. S_m aims to align predictions with ground-truth both globally (region-based) and locally (object-based), making it particularly suitable for saliency detection tasks.

2. Maximum Enhanced-alignment Measure (E_m) The maximum enhanced-alignment measure (E_m) evaluates the alignment between the predicted saliency map (P) and the ground-truth (G) by directly comparing their pixel-level distributions. It is formulated as:

$$E_m = \max_{\beta} E_\beta, \quad E_\beta = \frac{2 \cdot \Phi(P, G)}{\Phi(P, P) + \Phi(G, G)}$$

where Φ denotes the alignment function that computes the similarity between the two distributions. β is a scaling factor applied to the prediction prior to computing E_β. This metric emphasizes precise alignment between the saliency prediction and the ground-truth, making it robust against outlier regions. E_m improves on previous evaluation metrics by enhancing the sensitivity to consistent patterns.

3. Maximum F-measure (F_m) The F-measure is a widely used metric in binary classification problems and is defined as the harmonic mean of precision and recall. For saliency detection, the maximum F-measure is computed as:

$$F_\beta = \frac{(1 + \beta^2) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

and the maximum value across all thresholds is used:

$$F_m = \max_{\tau} F_\beta(\tau)$$

where β^2 is typically set to 0.3 to emphasize precision over recall, and τ is the threshold applied to the predicted saliency map. F_m provides a trade-off between precision and recall, offering a balanced evaluation of the model's capacity to detect salient regions.

4. Mean Absolute Error (MAE) The mean absolute error (MAE) computes the average pixel-wise absolute difference between the predicted saliency map (P) and the ground-truth (G) as:

$$\text{MAE} = \frac{1}{W \cdot H} \sum_{i=1}^{W} \sum_{j=1}^{H} |P(i, j) - G(i, j)|$$

where W and H are the width and height of the saliency map, respectively. MAE provides a straightforward assessment of the overall accuracy of the model's predictions, measuring how much the predicted saliency map differs from the ground-truth on average. These metrics collectively provide a comprehensive evaluation framework, ensuring both structural alignment and pixel-wise accuracy are taken into account.

5. Frames Per Second (FPS) The formula for FPS is:

$$\text{FPS} = \frac{N_{\text{frames}}}{T} \tag{21}$$

where FPS is the number of frames per second, N_{frames} is the total number of frames processed or rendered, and T is the time duration in seconds.

FPS (Frames Per Second) measures how many complete images (frames) are processed, rendered, or displayed by a system in one second. A higher FPS value indicates smoother motion and better rendering or processing performance, which is especially important in applications such as gaming, video playback, or real-time video processing.

6. Giga Floating Point Operations Per Second (GFLOPs) The formula for GFLOPs is:

$$\text{GFLOPs} = \frac{N_{\text{FLOPs}}}{10^9 \times T} \tag{22}$$

where GFLOPs is the number of giga floating point operations per second, N_{FLOPs} is the total number of floating point operations performed, T is the time duration in seconds, and 10^9 is the factor to convert operations to billions.

GFLOPs is a metric used to quantify the computational capability of a processor or accelerator. It represents how many billion floating point operations can be performed in one second. Floating point operations are critical for computations

in areas such as scientific computing, graphics rendering, and machine learning. Higher GFLOPs values indicate greater computational power and efficiency in handling complex calculations.

7. Number of Parameters (Params) The formula for the total number of parameters is:

$$\text{Params} = \sum_{i=1}^{L} p_i \tag{23}$$

where Params is the total number of parameters in the model, L is the number of layers, and p_i represents the number of parameters in the i-th layer.

Params is an indicator used to measure the complexity and capacity of machine learning models, particularly neural networks. It quantifies the total number of learnable weights and biases in the entire model. A larger number of parameters generally means the model has a greater ability to capture complex patterns, but it may also increase memory requirements and the risk of overfitting.

C. Here are some additional clarifications.

our SimpleDiffusion prioritizes lightweight design and efficiency rather than simplicity. The simple encoder-decoder pair essentially functions as a downsampling–upsampling module and is trained jointly. The encoder consists of a 7×7 convolution, a ResNet block, and stacked 3×3 convolutions. The decoder is composed of stacked 1×1 convolutions and upsampling layers, with skip connections for feature fusion. The details can be found in lines 467–473 and 477–493 of simple-diffusion-main\model\net.py in our repository. The original image size is 384×384×3. After FRFM, it becomes 96×96×64. Within ACFCN, the feature resolutions of the Transformer encoder layers are 96×96×64, 48×48×128, 24×24×256, and 12×12×512, respectively. After Multi-Scale Emphasis Aggregation, the Cross-Modal Fusion Feature is 96×96×256. In LDN, it is fused with the noise-added GT (384×384×1), resulting in a 192×192×256 tensor, which is then upsampled and decoded to generate the final iterative masks (384×384×1). We have explored spatially varying similarity and attention mechanisms for multimodal feature alignment. However, they showed no significant performance advantage and drastically increased computation, which goes against our goal of building a lightweight and efficient diffusion-based MSOD method. Although cosine similarity is relatively simple, it is effective and stable. Nevertheless, we will further investigate the alternative strategies you proposed in future work. In Eq. (10), we use the change magnitude between two adjacent denoised predictions as an indirect measure of confidence for each generated mask. During denoising, the changes between masks diminish and stabilize, warranting higher weights for later steps.

In fact, most discriminative MSOD methods (e.g., VSCode, LAFB, MMNet) typically involve two components: (1) the design of complex pre-fusion modules such as multimodal attention mechanisms and multi-branch feature fusion networks to integrate multimodal salient cues, and (2) manually crafted post-refinement strategies to further optimize the saliency maps. In contrast, diffusion-based MSOD methods also consist of two parts: a multimodal salient feature conditioning network and a denoising network. Taking SimpleDiffusion as an example, our Adaptive Cross-Modal Fusion Conditional Network (ACFCN) employs a standard Pyramid Vision Transformer (PVT) combined with a simple cosine similarity mechanism for efficient multimodal fusion and salient feature extraction—without the need for elaborate attention modules or fusion branches. Furthermore, the subsequent Latent Denoising Network (LDN) benefits from the inherent iterative denoising process of diffusion models, allowing progressive refinement of salient objects without requiring manually designed refinement strategies. Thus, our intended meaning is that the combination of these two components inherently eliminates the need for designing additional complex and time-consuming fusion and refinement modules. The improvement in inference time is not solely attributed to the choice of sampler. Instead, SimpleDiffusion achieves efficiency through a combination of strategies: reducing the number of sampling steps, replacing the conventional UNet with a more lightweight denoiser, and performing inference in the latent space to avoid processing high-resolution images directly. This optimization is a result of joint design across multiple components.

Recent SOTA diffusion-based methods, such as SOD-Diffusion (PG 2024), diffSOD (ICASSP 2025), and DiMSOD (AAAI 2025), are all built by fine-tuning Stable Diffusion. Originally proposed by the CompVis in 2022, Stable Diffusion comprises two autoencoders, a UNet denoising network, a CLIP text encoder, and a noise scheduler. One autoencoder compresses images into latent space for noising, while the other reconstructs the image. The UNet denoising network predicts and removes noise during the reverse diffusion process. The CLIP text encoder extracts semantic guidance, and the DDPM/DDIM scheduler controls noise levels at each step to ensure stability and coherence. Pretrained on the massive

LAION-5B dataset (2PB), it embeds rich visual priors and has over 2GB of parameters. Current diffusion-based SOD methods typically modify Stable Diffusion by replacing the CLIP text encoder with a transformer for RGB saliency feature extraction (e.g., SOD-Diffusion), optimizing attention in the denoising network and fusing modality features via ResNet (e.g., DiffSOD), or incorporating ControlNet for multi-modal fusion (e.g., DiMSOD). Despite these adaptations, their backbone remains Stable Diffusion, resulting in large and inefficient models. Our SimpleDiffusion is a lightweight diffusion model built entirely from scratch. We design a pair of simple convolutional encoder-decoder networks to fully replace the autoencoders. We also introduce LDN, a compact denoising module, in place of the heavy UNet. Furthermore, we tailor the architecture to the unique characteristics of the MSOD task by incorporating novel modules such as ACFCN and FRFM, along with new training and sampling strategies (SVA, SSR, WCE) that replace the original scheduler. As a result, our model achieves over 10× reduction in size while delivering superior accuracy and inference speed, making diffusion models more suitable for real-world SOD applications.

As illustrated in Figure 1, during training, SimpleDiffusion employs a pair of lightweight encoders and decoders stacked by convolutional neural networks. The encoder extracts GT features for subsequent noising, and the decoder serves as the mask generator. For the MSOD task, when the input is a binary GT mask, the input channel for the encoder is set to 1, with other settings unchanged. For the SIS task, where the input is instance-level GT mask, the encoder's input channel is set to 3. Similarly, the corresponding decoder (mask generator) follows this configuration. Specifically, in the output layer of the decoder, we change the dimension of the generated mask with a 1×1 convolution. For binary masks, the number of 1×1 kernels is set to 1; for instance-level masks, it is set to 3. These parameters are configurable—in our open-source code, mask_chans and class_num in simple-diffusion-main/config/model.yaml correspond to the input GT channel number and the output mask channel number, and can be flexibly set to 1 or 3 depending on the task (SOD or SIS). The specific code implementation can be found at lines 481–482 of simple-diffusion-main/model/net.py: ConvModule(in_channels=embedding_dim // 2, out_channels=self.num_classes, kernel_size=1, norm_cfg=dict(type='BN', requires_grad=True)), Please refer to our open-source repository, as linked in the main paper, for further details.

D. Results of Other Comparative Experiments.

To demonstrate that our multi-scale feature extraction backbone can be seamlessly replaced with other general-purpose backbones, we replaced the PVT backbone with ResNet50 and conducted extensive comparisons with other methods. The detailed results are presented below.

Table 4. Quantitative Comparison of E-Measure (E_ξ), Weighed F-Measure (F_β^ω), F-Measure (F_β), Mean Absolute Error (MAE) and Frame-Per-Second (FPS) with 17 Different **Non-diffusion-based RGB-D SOD** methods on seven Testing Datasets, Including SIP, DUT-RGBD (i.e., DUT), SSD, STERE, ReDWeb-S, NLPR, and NJUD. The Best, Second Best Results are Marked with Red and Green, Respectively. '-' Indicates the Code or Result Is Not Available.

Methods	Backbone	SIP				DUT				SSD				STERE				ReDWeb-S				NLPR				NJUD				FPS ↑	Param ↓	GFLOPs ↓
		$E_\xi\uparrow$	$F_\beta^\omega\uparrow$	$F_\beta\uparrow$	$MAE\downarrow$	$E_\xi\uparrow$	$F_\beta^\omega\uparrow$	$F_\beta\uparrow$	$MAE\downarrow$	$E_\xi\uparrow$	$F_\beta^\omega\uparrow$	$F_\beta\uparrow$	$MAE\downarrow$	$E_\xi\uparrow$	$F_\beta^\omega\uparrow$	$F_\beta\uparrow$	$MAE\downarrow$	$E_\xi\uparrow$	$F_\beta^\omega\uparrow$	$F_\beta\uparrow$	$MAE\downarrow$	$E_\xi\uparrow$	$F_\beta^\omega\uparrow$	$F_\beta\uparrow$	$MAE\downarrow$	$E_\xi\uparrow$	$F_\beta^\omega\uparrow$	$F_\beta\uparrow$	$MAE\downarrow$			
$HDFNet_{20}$	VGG19	0.920	0.848	0.862	0.048	0.937	0.873	0.887	0.040	0.902	0.823	0.831	0.047	0.924	0.863	0.860	0.041	0.760	0.623	0.704	0.133	0.949	0.873	0.874	0.028	0.911	0.879	0.856	0.039	70	164.19	111.02
$CoNet_{20}$	ResNet101	0.909	0.814	0.842	0.063	0.948	0.896	0.909	0.033	0.896	0.792	0.806	0.059	0.928	0.874	0.885	0.037	0.762	0.618	0.688	0.147	0.934	0.850	0.848	0.031	0.912	0.856	0.873	0.046	54	162.13	13.05
$CCAFNet_{21}$	ResNet50	0.915	0.839	0.864	0.054	0.940	0.884	0.903	0.037	0.915	0.839	0.864	0.054	0.921	0.853	0.869	0.044	0.687	0.515	0.614	0.173	0.951	0.883	0.880	0.026	0.920	0.883	0.896	0.037	68	159.45	189.47
$CDNet_{21}$	VGG16	0.913	0.839	0.870	0.056	0.936	0.878	0.901	0.039	0.849	0.706	0.742	0.073	0.929	0.871	0.884	0.039	0.730	0.588	0.678	0.146	0.951	0.886	0.879	0.025	0.903	0.828	0.856	0.054	66	123.65	72.07
$HAINet_{21}$	VGG16	0.924	0.860	0.883	0.049	0.937	0.887	0.905	0.038	0.843	0.682	0.735	0.101	0.930	0.877	0.888	0.038	0.766	0.654	0.713	0.132	0.951	0.884	0.884	0.025	0.917	0.882	0.895	0.039	11	228.20	181.62
DFM_{21}	MobileNetV2	0.919	0.844	0.871	0.051	0.898	0.795	0.830	0.062	0.871	0.733	0.755	0.076	0.912	0.850	0.858	0.045	0.753	0.610	0.690	0.136	0.945	0.876	0.868	0.026	0.913	0.868	0.885	0.042	52	8.45	5.43
$SPNet_{21}$	ResNet50	0.930	0.873	0.891	0.043	0.876	0.747	0.843	0.085	0.910	0.831	0.852	0.044	0.930	0.879	0.886	0.037	0.759	0.637	0.712	0.129	0.957	0.899	0.898	0.021	0.931	0.909	0.915	0.029	50	573.39	68.10
$RD3D_{21}$	I3DResNet50	0.919	0.852	0.873	0.049	0.949	0.913	0.925	0.031	0.905	0.794	0.812	0.052	0.926	0.877	0.885	0.038	0.701	0.487	0.595	0.177	0.957	0.894	0.890	0.022	0.918	0.890	0.900	0.037	54	110.30	43.51
$DSA2F_{21}$	VGG19	0.908	0.838	0.867	0.057	0.950	0.914	0.926	0.030	0.904	0.836	0.852	0.047	0.928	0.877	0.895	0.038	-	-	-	-	0.950	0.889	0.897	0.024	0.923	0.889	0.901	0.039	-	-	-
DCF_{21}	ResNet50	0.920	0.850	0.877	0.051	0.952	0.913	0.925	0.030	0.898	0.800	0.829	0.053	0.931	0.880	0.890	0.037	0.755	0.632	0.710	0.135	0.956	0.892	0.893	0.023	0.922	0.884	0.897	0.038	57	111.51	59.38
$MobileSal_{21}$	MobileNetV2	0.914	0.837	0.860	0.054	0.936	0.869	0.912	0.044	0.898	0.804	0.815	0.052	0.916	0.865	0.848	0.041	0.671	0.455	0.608	0.186	0.950	0.878	0.872	0.025	0.914	0.874	0.894	0.040	68	24.97	1.96
SSL_{22}	VGG16	0.921	0.851	0.875	0.049	0.927	0.859	0.888	0.046	0.833	0.638	0.696	0.100	0.923	0.864	0.875	0.042	-	-	-	-	0.954	0.885	0.884	0.027	0.881	0.786	0.821	0.065	52	282.95	272.20
$DIGRNet_{22}$	ResNet50	0.918	0.849	0.878	0.053	0.948	0.902	0.919	0.033	0.889	0.804	0.823	0.053	0.927	0.877	0.888	0.038	-	-	-	-	0.955	0.895	0.888	0.023	0.928	0.909	0.916	0.028	33	635.79	68.16
$CIRNet_{22}$	ResNet50	0.917	0.848	0.874	0.053	0.951	0.908	0.926	0.031	0.898	0.791	0.821	0.054	0.921	0.836	0.875	0.049	0.725	0.519	0.630	0.171	0.955	0.889	0.883	0.023	0.922	0.881	0.896	0.040	63	393.50	42.60
$MoADNet_{22}$	MobileNetV3	0.908	0.828	0.850	0.058	0.949	0.911	0.923	0.031	0.894	0.801	0.824	0.057	0.914	0.861	0.868	0.042	-	-	-	-	0.945	0.875	0.874	0.027	0.909	0.881	0.892	0.041	67	19.19	1.32
$LSNet_{23}$	MobileNetV2	0.927	0.856	0.882	0.049	0.891	0.775	0.831	0.074	0.902	0.796	0.820	0.055	0.913	0.827	0.854	0.054	0.691	0.458	0.566	0.193	0.955	0.881	0.882	0.024	0.922	0.885	0.899	0.038	316	17.41	3.04
$CAVER_{23}$	ResNet50	0.927	0.874	0.884	0.043	0.955	0.920	0.919	0.029	0.915	0.826	0.828	0.044	0.931	0.887	0.871	0.034	0.760	0.663	0.724	0.121	0.959	0.899	0.894	0.022	0.922	0.903	0.874	0.032	67	451.8	137.64
Ours	ResNet50	0.939	0.887	0.908	0.040	0.959	0.921	0.932	0.025	0.926	0.845	0.867	0.040	0.933	0.892	0.905	0.035	0.769	0.675	0.738	0.129	0.960	0.912	0.914	0.021	0.927	0.918	0.929	0.028	72	54.04	39.42

Table 5. Quantitative Comparison of E-measure (E_ξ), Weighed F-measure (F_β^ω), F-measure (F_β), Mean Absolute Error (MAE) and Frame-Per-Second (FPS) with 14 Different **Non-diffusion-based RGB-T SOD** Methods on Three Testing Datasets, Including VT5000, VT1000 (Tu et al., 2019) and VT821. The Best, Second Best are Marked with Red and Green, Respectively

Methods	Backbone	VT5000				VT1000				VT821				FPS ↑	Param ↓	GFLOPs ↓
		E_ξ ↑	F_β^ω ↑	F_β ↑	MAE ↓	E_ξ ↑	F_β^ω ↑	F_β ↑	MAE ↓	E_ξ ↑	F_β^ω ↑	F_β ↑	MAE ↓			
$MTMR_{18}$ (Wang et al., 2018)	-	0.795	0.397	0.595	0.114	0.836	0.485	0.715	0.119	0.815	0.462	0.662	0.108	-	-	-
$M3S\text{-}NIR_{19}$	-	0.780	0.327	0.575	0.168	0.827	0.463	0.717	0.145	0.859	0.407	0.734	0.407	-	-	-
$SGDL_{19}$ (Tu et al., 2019)	-	0.824	0.559	0.672	0.089	0.856	0.652	0.764	0.090	0.847	0.583	0.730	0.085	-	-	-
ADF_{20}	VGG16	0.891	0.722	0.778	0.048	0.921	0.804	0.847	0.034	0.842	0.627	0.077	0.716	27	254.67	191.45
$MIDD_{21}$	VGG16	0.897	0.763	0.801	0.043	0.933	0.856	0.882	0.027	0.895	0.760	0.804	0.045	33	200	216.72
$CSRNet_{21}$ (Huo et al., 2021)	ESPNetV2	0.905	0.796	0.811	0.042	0.925	0.878	0.877	0.024	0.909	0.821	0.831	0.038	-	-	-
$CGFNet_{21}$	VGG16	0.922	0.831	0.851	0.035	0.944	0.900	0.906	0.023	0.912	0.829	0.845	0.038	18	266.73	347.78
$MMNet_{21}$	ResNet50	0.887	0.770	0.780	0.043	0.923	0.863	0.861	0.027	0.892	0.783	0.794	0.040	-	-	-
$ECFFNet_{21}$ (Zhou et al., 2021)	ResNet34	0.906	0.802	0.807	0.038	0.93	0.885	0.876	0.021	0.902	0.801	0.810	0.034	-	-	-
$MIA\text{-}DPD_{22}$	ResNet50	0.893	0.780	0.793	0.040	0.926	0.864	0.868	0.025	0.850	0.720	0.741	0.070	-	-	-
$OSRNet_{22}$	ResNet50	0.908	0.807	0.823	0.040	0.935	0.891	0.892	0.022	0.896	0.801	0.814	0.043	42	59.67	51.27
$DCNet_{22}$	VGG16	0.920	0.819	0.847	0.035	0.948	0.902	0.911	0.021	0.912	0.823	0.841	0.033	43	91.57	207.21
$LSNet_{23}$	MobileNetV2	0.915	0.806	0.825	0.037	0.935	0.887	0.885	0.023	0.911	0.809	0.825	0.033	314	17.41	3.04
$CAVER_{23}$ (Pang et al., 2023)	ResNet50	0.924	0.835	0.841	0.032	0.945	0.909	0.903	0.017	0.919	0.835	0.839	0.033	67	451.8	137.68
Ours	ResNet50	0.936	0.857	0.868	0.030	0.948	0.915	0.914	0.020	0.924	0.828	0.863	0.033	72	54.04	39.42

Furthermore, our model achieves outstanding results in RGB-D instance segmentation tasks. To the best of our knowledge, existing models for RGB-D instance segmentation (SIS) are relatively limited. To conduct a comprehensive comparison, we include models from several related tasks, such as S4Net, RDPNet, and OQTR from the SIS field; OSFormer from the camouflaged instance segmentation (CIS) field; and CondInst, SOLOv2, QueryInst, SOTR, Mask Transfiner, SparseInst, and Mask2Former from the general instance segmentation (GIS) field. Additionally, we incorporate RRL and CalibNet from the RGB-D GIS domain. All methods have been uniformly adapted to address RGB-D salient instance segmentation (SIS).

Table 6. Quantitative comparisons with task-related methods on **RGB-D SIS**

Task	Method	COME15K-E			DSIS			SIP		
		AP	AP_{50}	AP_{70}	AP	AP_{50}	AP_{70}	AP	AP_{50}	AP_{70}
SIS	S4Net	43.7	68.0	52.5	58.3	81.9	71.8	49.6	76.0	63.7
	RDPNet	49.8	72.2	59.5	66.1	87.2	80.1	59.0	80.1	74.1
	OQTR	48.7	71.1	58.3	63.1	85.9	77.0	59.9	83.1	76.3
CIS	OSFormer	53.0	71.9	61.3	67.9	86.3	78.3	63.2	80.5	74.6
GIS	CondInst	49.6	72.0	59.5	65.1	86.8	79.1	58.6	79.4	73.3
	SOLOv2	51.1	71.6	59.9	67.4	87.0	80.4	63.4	80.7	74.8
	QueryInst	51.5	73.1	61.1	67.9	87.5	80.6	61.3	81.0	75.1
	SOTR	50.7	70.4	59.0	68.2	86.9	79.3	61.6	78.4	73.0
	MaskTran	48.7	71.0	56.3	67.5	87.6	80.4	57.8	78.9	70.3
	SparseInst	51.3	71.9	58.9	65.0	85.0	75.7	62.8	81.3	75.3
	Mask2For	51.5	67.3	57.6	68.3	85.1	76.7	66.6	79.3	75.1
DSIS	RRL	52.9	71.8	61.8	66.1	87.1	79.6	61.9	79.1	74.6
	CalibNet	58.0	75.8	65.6	69.3	87.8	81.6	72.1	86.6	82.9
	Ours	**66.4**	**82.8**	**75.4**	**74.4**	**90.7**	**88.6**	**79.1**	**92.4**	**89.7**

E. Extensive Generalized Experiments

Building upon previous work, we conducted additional experiments on the RGB-D Salient Instance Segmentation Dataset. Remarkably, without making any modifications to our SimpleDiffusion framework, we only replaced the input GT mask with segmentation masks, and our model achieved excellent instance segmentation results. This further suggests that our multimodal salient object detection framework, SimpleDiffusion, is a versatile solution applicable to a wide range of saliency-related tasks. Please refer to the figure below for more details.

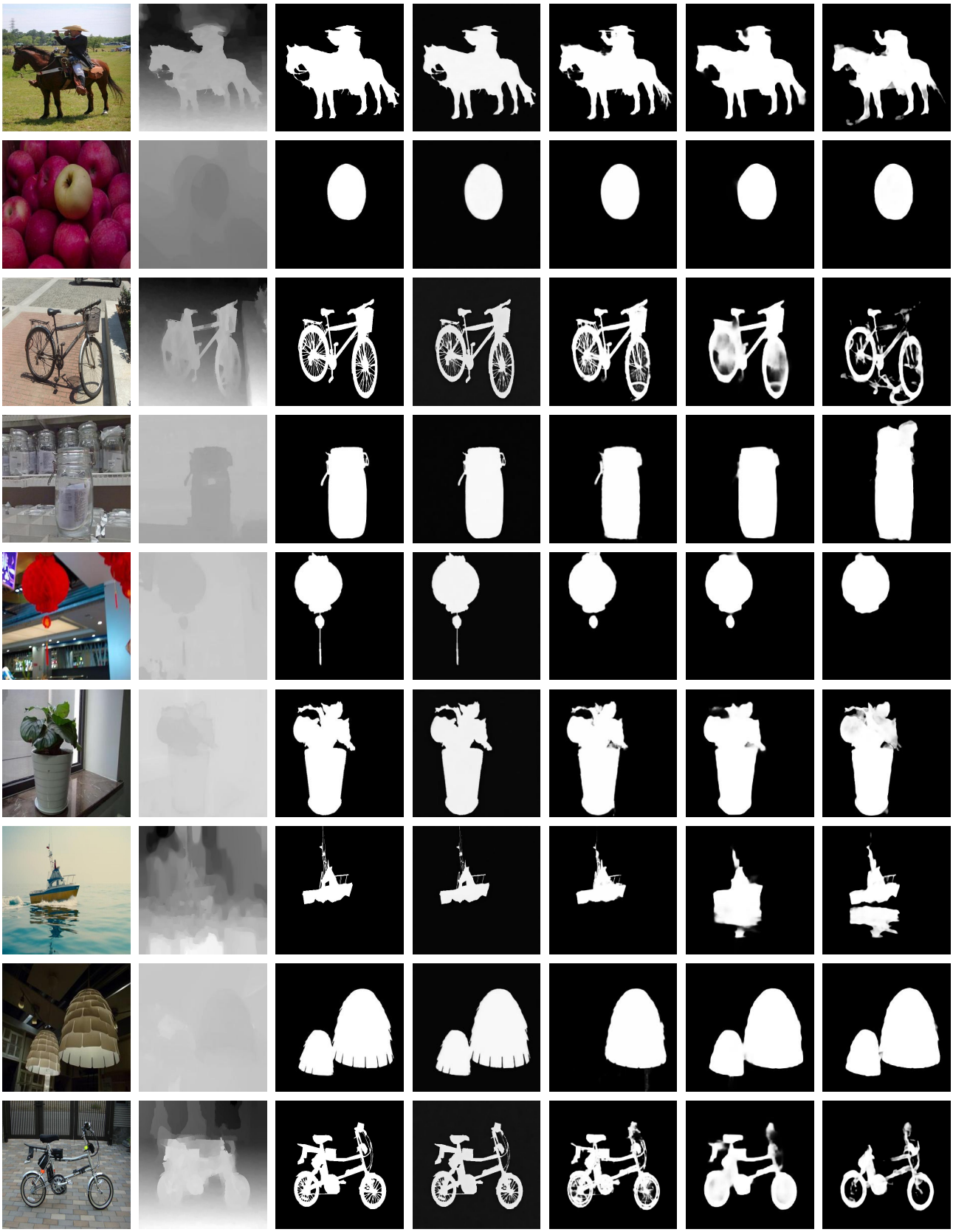

Figure 9. Detailed visual comparison of saliency map results generated by various advanced methods for RGB-D SOD.

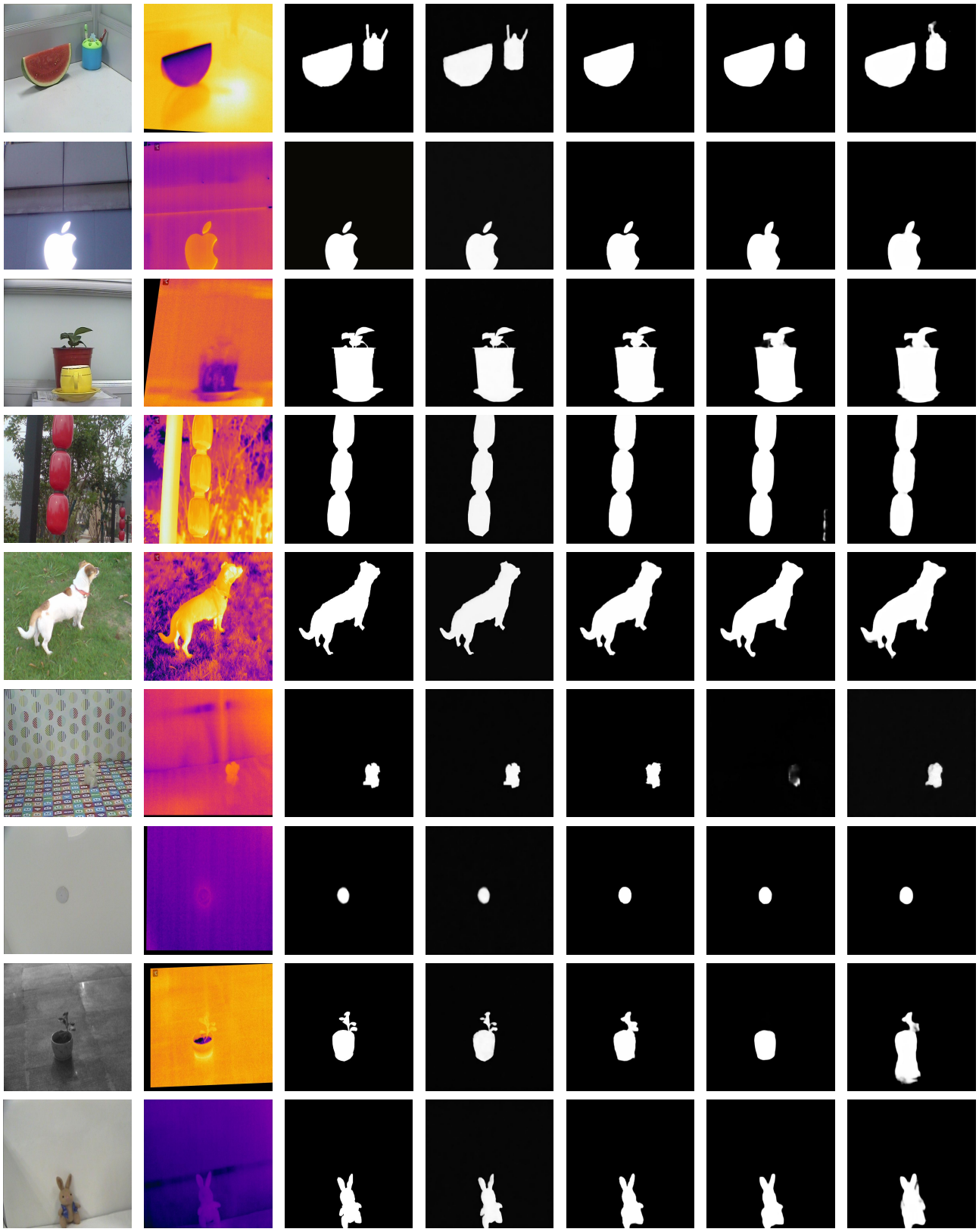

Figure 10. Detailed visual comparison of saliency map results generated by various advanced methods for RGB-T SOD.

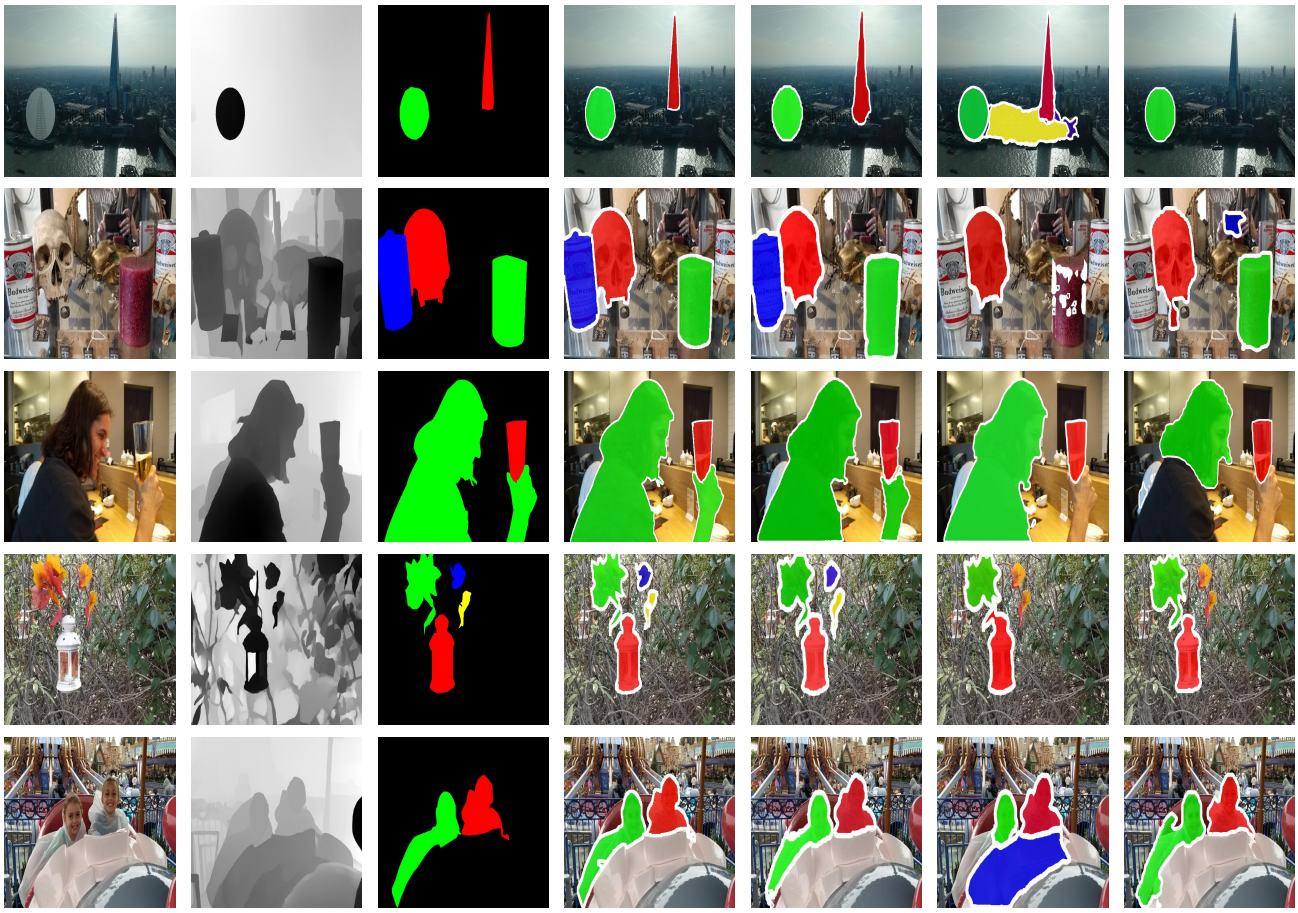

Figure 11. Visual Comparison Results of RGB-D SIS.