Efficient Video Object Segmentation and Tracking with Recurrent Dynamic Submodel

Supplementary Material

In this supplementary material, we provide additional implementation details, analyses, and visualizations to support the main paper. Specifically:

- Section A provides detailed implementation specifics for RDS and the baseline methods.
- Section B analyzes routing patterns under different activation budgets and explains the ranking behavior of I-LoRA.
- Section C reports detailed quantitative results and further motivates the need for dynamic submodels.
- Section D visualizes routing behavior at the frame level, video level, and temporal level.

A. Extended Implementation Details

A.1. RDS Architecture and Implementation

In the actual implementation, the autograd function referred to in code as `RoundSTE` performs Bernoulli sampling in the forward pass and uses a straight-through estimator (STE) in the backward pass. Accordingly, for dynamically routed blocks we interpret $\tilde{w}_t^i$ as an activation probability under the expected routing budget, rather than as a calibrated hard score. When the previous SAM mask logits are unavailable or marked invalid, such as at initialization, the model falls back to the full encoder by setting $\mathbf{r}_t = \mathbf{1}$. At evaluation time, we may either keep stochastic routing, i.e., sample $r_t^i \sim \mathrm{Bernoulli}(\tilde{w}_t^i)$ for each $i \notin \mathcal{K}$, to preserve train-test consistency, or use the deterministic deployment rule $r_t^i = \mathbb{1}[\tilde{w}_t^i \geq \tau]$ for $i \notin \mathcal{K}$, with $r_t^i = 1$ for $i \in \mathcal{K}$. In practice, τ can be calibrated on a validation set, with $\tau = 0.5$ as the default choice.

The router first binarizes the previous SAM mask logits with a zero threshold, resizes the mask to the encoder input resolution, embeds it with a dedicated mask patch embedding layer, and fuses it additively with the current-frame image feature. The fused feature is then processed by a lightweight convolutional router head and an MLP projection to produce block-wise routing logits.

Algorithm 1 Dynamic Block Selection via PAR

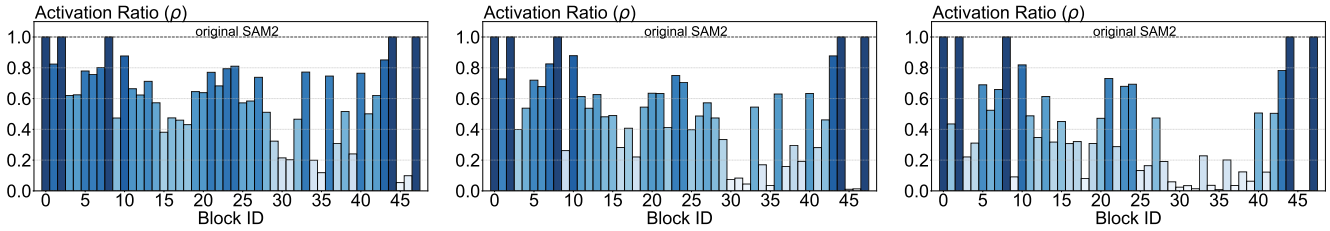
Require: Current-frame feature $X_t \in \mathbb{R}^{H\times W\times C}$; previous segmentation mask M_{t-1}; fusion layers f and g; prediction-aware router ϕ; learnable scalar α; fixed block set K; target activation ratio β; total number of blocks L.

Ensure: Binary routing vector $\mathbf{r}_t \in \{0,1\}^L$ for block activation.

Initialize default routing vector $\mathbf{r}_t \leftarrow \mathbf{1}$
if M_{t-1} is available **then**
  $Y_t \leftarrow \alpha \cdot f(M_{t-1}) + g(X_t)$
  $\mathbf{l}_t \leftarrow \phi(Y_t)$
  $\mathbf{p}_t \leftarrow \sigma(\mathbf{l}_t)$
  $\tilde{\mathbf{w}}_t \leftarrow \min\left(\frac{\beta\cdot L\cdot \mathbf{p}_t}{\sum_{j=1}^{L} p_t^j}, 1.0\right)$ *// Normalize continuous activation probabilities*
  for each block $i = 0, \ldots, L-1$ **do**
    if $i \in K$ **then**
      $r_t^i \leftarrow 1$
    else if training **then**
      $\tilde{r}_t^i \sim \mathrm{Bernoulli}(\tilde{w}_t^i)$
      $r_t^i \leftarrow \mathrm{STE}(\tilde{r}_t^i, \tilde{w}_t^i)$
    else
      $r_t^i \leftarrow \mathbb{1}[\tilde{w}_t^i \geq 0.5]$ *// Deterministic routing during inference*
    end if
  end for
end if
return $\mathbf{r}_t$

A.2. Baseline Implementation Details

- **SlimSAM: A Static Channel Pruning Method**
  SlimSAM is a prominent static pruning method that creates a smaller, fixed architecture by removing network channels. The process consists of three main steps. First, we identify important channels using the disturbed Taylor importance metric on the "sav-000" subset of the SA-V dataset [32]. This training is computationally intensive, performed on 8 Nvidia A100 80G GPUs for 150 epochs. Second, we set a global pruning ratio to create a static model with a computational cost (FLOPs) comparable to our $\mathrm{RDS}_{0.4}$ variant. Third, we fine-tune the pruned model on the "sav-006" subset and DAVIS 2017 [28] with differential learning rates (3e-6 for the backbone, 5e-6 for the head) to maximize its performance.
- **Progressive Block Drop (PBD): A Static Block Pruning Method**
  PBD offers a straightforward static pruning approach by removing entire computational blocks. First, to align with the computational budget of our $\mathrm{RDS}_{0.4}$ variant, we remove the last 24 blocks of the SAM2 encoder, creating a shallower 24-block architecture. Second, the resulting model is extensively fine-tuned on the DAVIS 2017 dataset for 40 epochs. To ensure this baseline reaches its



(a) Target activation ratio 0.6. (b) Target activation ratio 0.5. (c) Target activation ratio 0.4.

Figure 10. **Activation statistics under different target budgets.** As the target activation ratio decreases, the router first suppresses blocks with more sample-specific behavior, while a relatively stable core remains active across all budgets. This visualization also clarifies why low-activation blocks can still be selected by I-LoRA: average activation frequency and adaptation importance are related but not identical. A block may be activated only on difficult frames, yet still produce a large feature transformation or a clear drop in $\mathcal{J}\&\mathcal{F}$ when removed.

peak potential, we integrate I-LoRA adapters into the remaining blocks with a learning rate of 1e-5, making it a optimized static competitor.

- **Dynamic Token Pruning (DyT): A Dynamic Token-level Method**
  DyT provides a dynamic baseline by adaptively pruning redundant image tokens. First, we integrate the DyT decision modules into the SAM2 encoder, calibrating the target token keeping ratio to match the average inference FLOPs of our $\text{RDS}_{0.4}$ variant. Second, the complete DyT-enabled model is fine-tuned on the DAVIS 2017 dataset for 40 epochs. We also integrate I-LoRA adapters into the architecture during this phase with a learning rate of 1e-5 for fair comparison.
- **AdaViT: A Dynamic Block-level Method**
  AdaViT is conceptually the closest baseline, utilizing per-block routers for dynamic block skipping. First, we insert a lightweight decision module before each of the 48 Transformer blocks in the SAM2 encoder, using a sparsity loss to encourage a skipping rate of approximately 50%, aligning its budget with our $\text{RDS}_{0.4}$ variant. Second, the complete AdaViT-enabled architecture is fine-tuned on the DAVIS 2017 dataset for 40 epochs. Crucially, we also apply I-LoRA adapters with a learning rate of 1e-5 during this stage to ensure the performance comparison focuses purely on the dynamic routing strategy.

A.3. Clarification on the DINOv3 Experiment

The DINOv3 result in Table 8 is intended to test whether our routing and parameter-allocation strategy transfers beyond SAM2, rather than to claim a new state-of-the-art video object segmentation model. We follow the official DINOv3 `segmentation_tracking` setting [33], which takes the first-frame mask as supervision, extracts dense DINOv3 features on the reference and target frames, and propagates pixel labels through feature matching.

For this reason, we report the standard semi-supervised VOS metric $\mathcal{J}\&\mathcal{F}$, which directly evaluates mask quality and temporal consistency. Using box-based VOT metrics here would require an additional box conversion step and would not faithfully measure the segmentation quality actually produced by the method. Because DINOv3 is not originally optimized as a dedicated video segmentation backbone, its absolute $\mathcal{J}\&\mathcal{F}$ values are much lower than those of SAM2-based systems. Therefore, Table 8 should be interpreted as evidence of *generalizability and Pareto improvement*: after adding RDS, DINOv3 becomes substantially faster while maintaining or slightly improving segmentation quality under the same evaluation protocol.

B. Analyzing Routing and Adaptation

B.1. Activation Patterns Across Target Budgets

Figure 10 provides a budget-wise view of the learned routing policy. Two observations are consistent across all three settings. First, activation is clearly selective and not strictly monotonic with depth, which suggests that the encoder does not contain a single contiguous "useful prefix" of blocks. Instead, different semantic functions are distributed across the network. Second, when the budget becomes tighter, the model does not uniformly reduce every block's usage. Rather, it retains a stable subset of blocks and suppresses the more specialized ones first.

This behavior directly supports the design of I-LoRA. The router tells us *how often* a block is used, but not necessarily *how important* that block is when it is used. Difficult frames involving appearance change, occlusion, or fast motion may require a block only occasionally, but those rare activations can still be decisive. Consequently, I-LoRA intentionally keeps a few low-frequency yet high-impact blocks in the adapted set, instead of allocating all trainable parameters to the most frequently activated blocks only.

B.2. Detailed I-LoRA Ranking

Figure 11 decomposes the ranking process of I-LoRA into its global, local, and combined components. The global score emphasizes blocks whose removal leads to a large output-quality drop, while the local score emphasizes

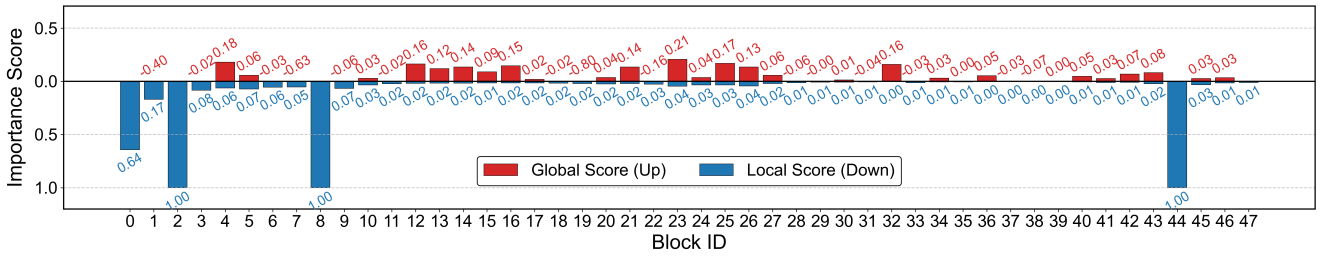


Figure 11. Two Perspectives Importance Score per Block.

blocks that substantially transform the intermediate representation. The final score is a rank aggregation of these two views rather than a simple reliance on activation frequency.

In practice, we set the always-active block set to $\mathcal{K} = \{0, 2, 8, 44, 47\}$. For each $i \in \{0, \ldots, L-1\} \setminus \mathcal{K}$, we compute the descending ranks $r_G(i)$ and $r_L(i)$ from $\mathcal{I}_G^i$ and $\mathcal{I}_L^i$, and define:

$$s(i) = -(r_G(i) + r_L(i)).$$

We then select the top $K_{\text{sel}} - |\mathcal{K}|$ blocks according to $s(i)$ and combine them with $\mathcal{K}$, exactly as in Equation 11 of the main paper. For the SAM2 setting used in the main experiments, we set $K_{\text{sel}} = 24$, yielding the final adapted blocks:

$$\{0, 2, 4, 5, 8, 10, 12\text{–}17, 20, 21, 23\text{–}27, 43\text{–}47\}.$$

This ranking behavior is important for the overall narrative of the paper: the adapted blocks are neither simply the shallowest blocks nor simply the most frequently activated blocks. Instead, they are the ones that remain important under both output-space and feature-space analyses.

C. Extended Quantitative Results and Motivation

C.1. Detailed Quantitative Results

Table 9 provides the exact numerical values corresponding to the performance-efficiency plots in the main paper. These results make the relative trends in Figure 4 explicit and show how the trade-off evolves as each baseline becomes more aggressive.

C.2. Why a Dynamic Submodel Is Necessary

To further illustrate the necessity of dynamic routing, we construct two distinct static architectures, denoted as Static plan A and Static plan B. Both plans operate at an identical activation budget (retaining 24 out of 48 blocks) but utilize different block configurations.

Specifically, Static plan A retains blocks: {0-4,6-8,12-13,15,18,21,25,28,32,34,35,38,39,42-44,47}, while Static plan B retains blocks: {0-5,7-8,13-14,16,19,20,26-28,33,34,36,37,42-44,47}.

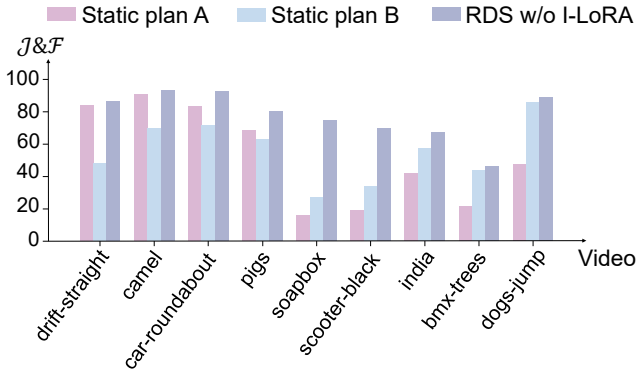


Figure 12. **Performance comparison of static plans and our method across video sequences.** Static plan A and B (same 24-block budget, different configurations) show large performance variance across videos, while our dynamic submodel achieves consistently robust results.

As visualized in Figure 12, we evaluate these static plans across diverse video sequences from the DAVIS 2017 dataset. The results demonstrate that a single static architecture may accidentally align well with one specific video's data distribution while significantly underperforming on another. For instance, Plan A outperforms Plan B on certain sequences, while the reverse is true for others.

This observation extends the intuition introduced in Figure 1(a) to the video and dataset levels. Furthermore, to explicitly illustrate how routing behaviors adapt at a more granular scale, we also provide detailed frame-level visualization comparisons in Figures 23, 24, and 25. Such performance fluctuation is especially pronounced in video object segmentation and tracking tasks, where motion patterns, object scales, background complexities, and appearance changes vary drastically across samples and continuous frames. Therefore, our proposed dynamic submodel is not merely a computational speed optimization, but a crucial mechanism for dynamically matching model capacity to sample diversity, ensuring robust performance across varied video conditions.

D. Visualizing Routing Behaviors

D.1. Video-level Average Activation Profiles

Figures 14, 16, 18, 20, and 22 report the average activation rate of each encoder block over complete videos. Across diverse sequences, several early and late blocks remain consistently active, while many middle blocks vary with scene dynamics, target scale, and background clutter. This complements Figure 9: RDS preserves a stable core pathway but adjusts additional computation in a video-specific manner instead of relying on a single fixed block prefix.

D.2. Per-frame Block Activations

To further illustrate the fine-grained temporal dynamics, we visualize the per-frame block activations alongside their corresponding raw video frames (see Figures 13, 15, 17, 19, and 21). As demonstrated, our Prediction-Aware Router effectively adjusts the network architecture on the fly. Rather than keeping a static architecture, the dynamic submodel instantly reacts to frame-level appearance changes, object deformations, and complex tracking scenarios, ensuring optimal allocation of computational resources over time.

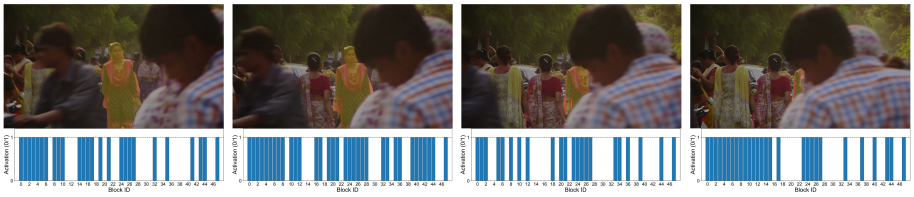


Figure 13. Per-frame block activations in the *india* video.

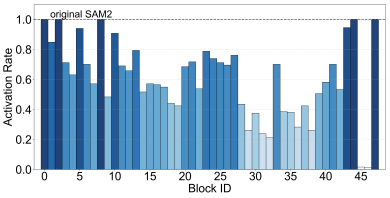


Figure 14. Video-level average block activation rates in the *india* video.

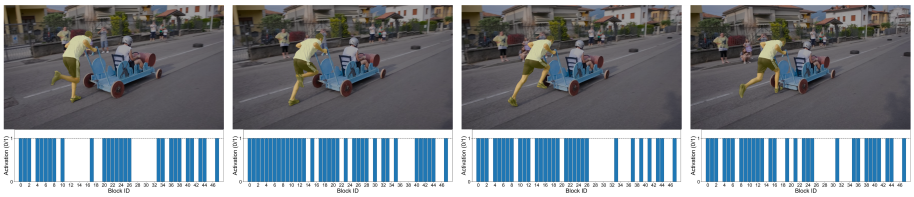


Figure 15. Per-frame block activations in the *soapbox* video.

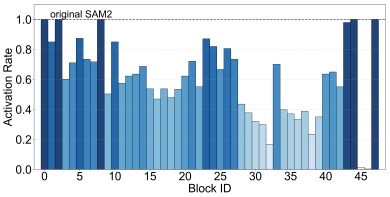


Figure 16. Video-level average block activation rates in the *soapbox* video.

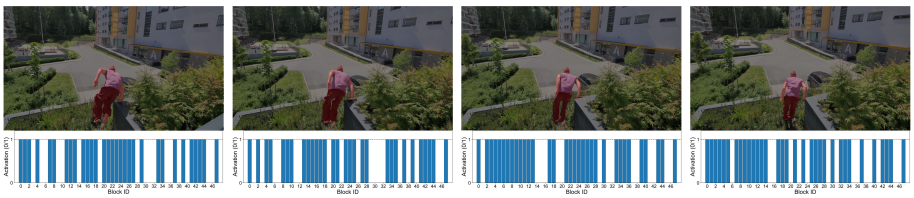


Figure 17. Per-frame block activations in the *parkour* video.

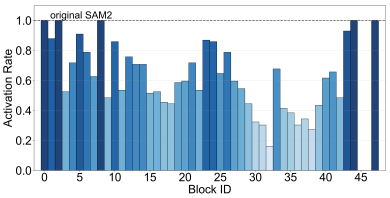


Figure 18. Video-level average block activation rates in the *parkour* video.

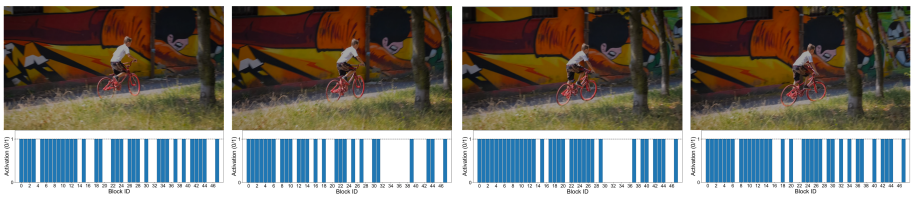


Figure 19. Per-frame block activations in the *bmx-trees* video

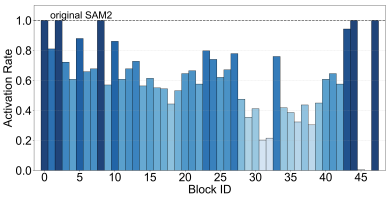


Figure 20. Video-level average block activation rates in the *bmx-trees* video.

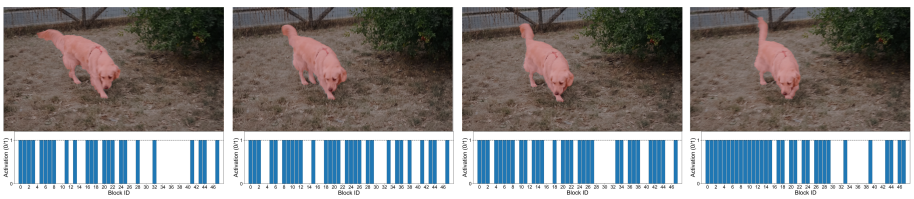


Figure 21. Per-frame block activations in the *dog* video.

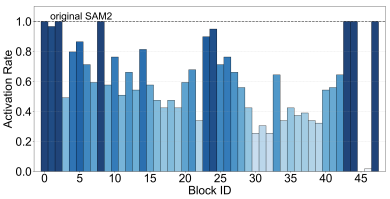


Figure 22. Video-level average block activation rates in the *dog* video.

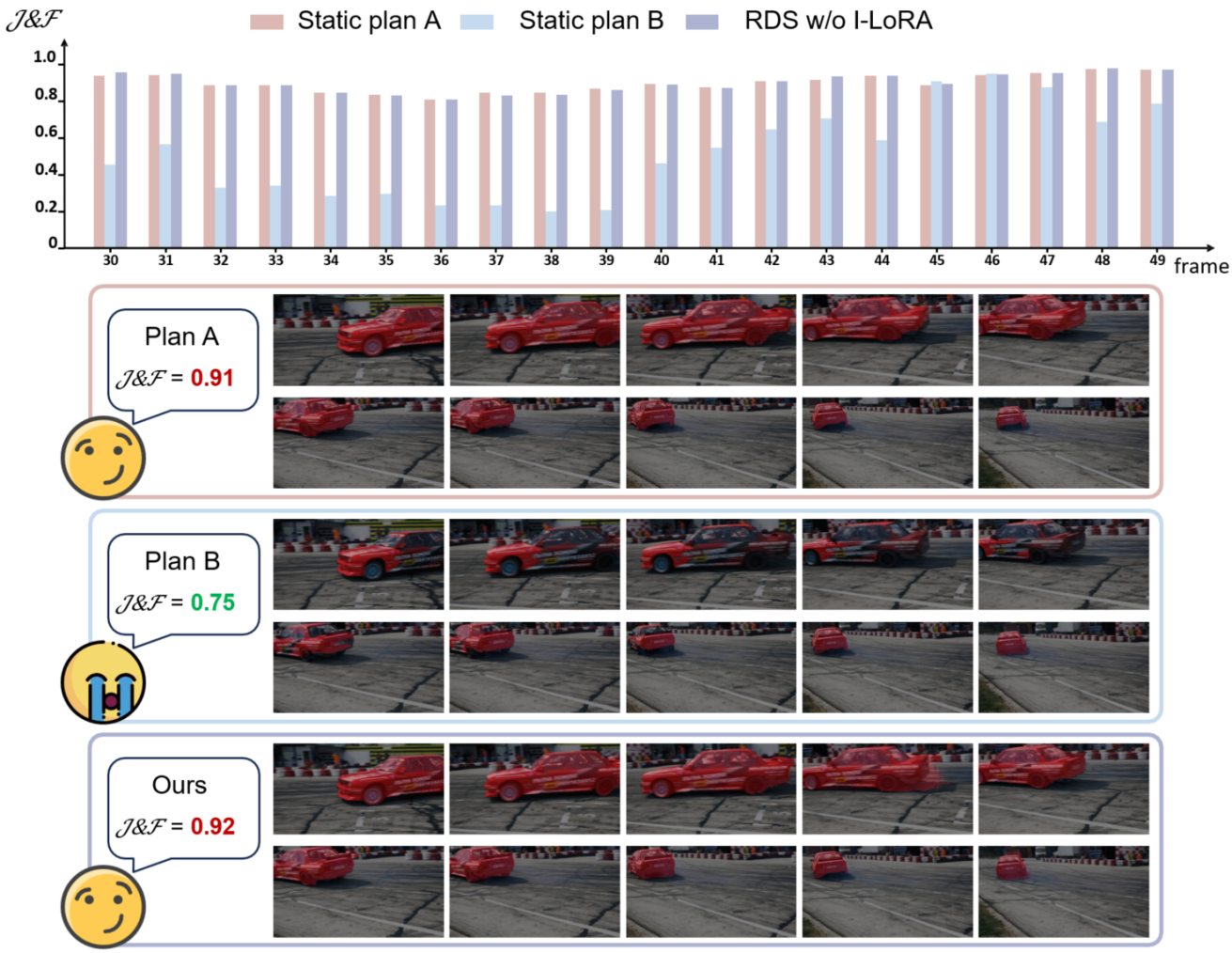


Figure 23. Detailed frame-level visualization comparisons in the *drift-straight* video.

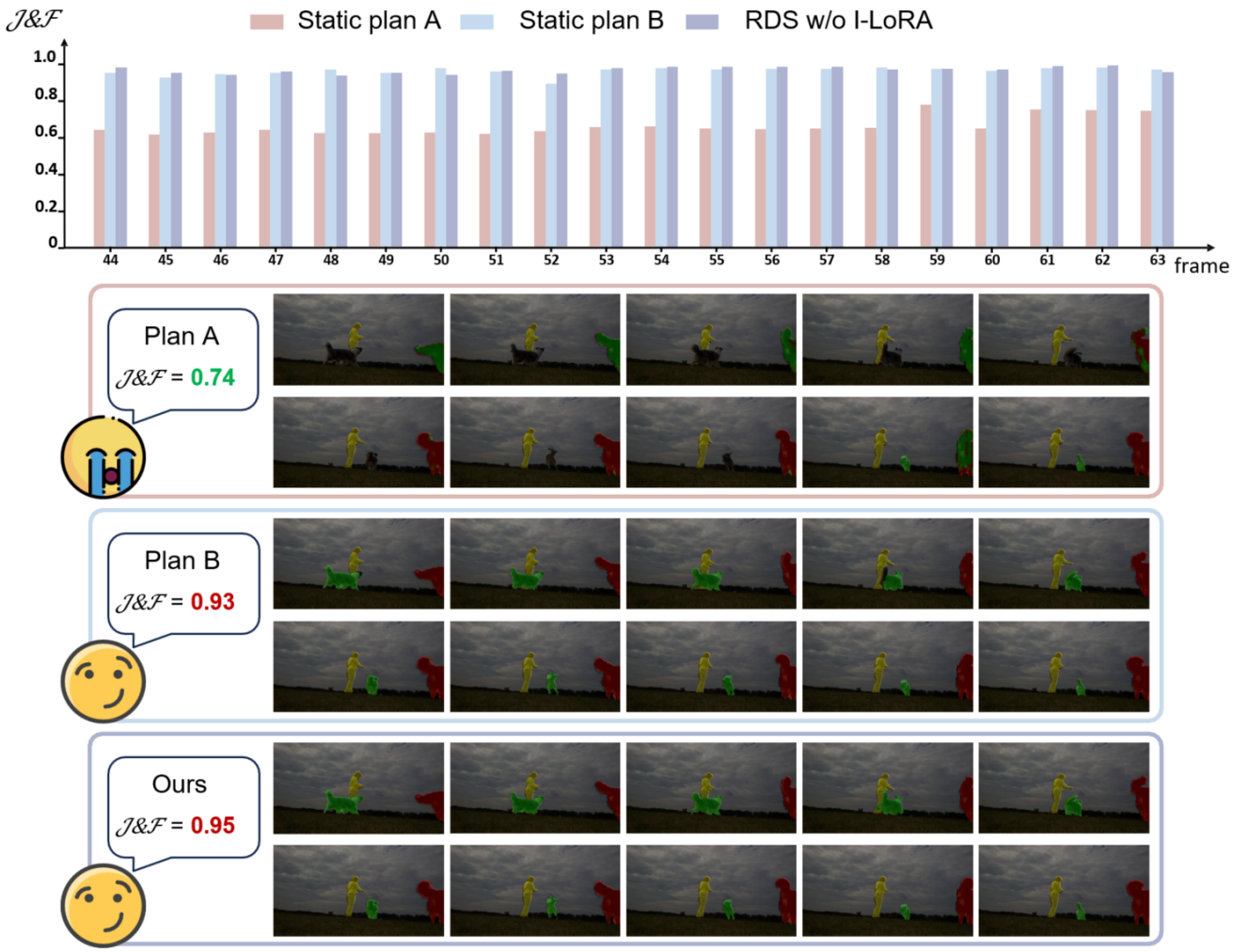


Figure 24. Detailed frame-level visualization comparisons in the *dogs-jump* video.

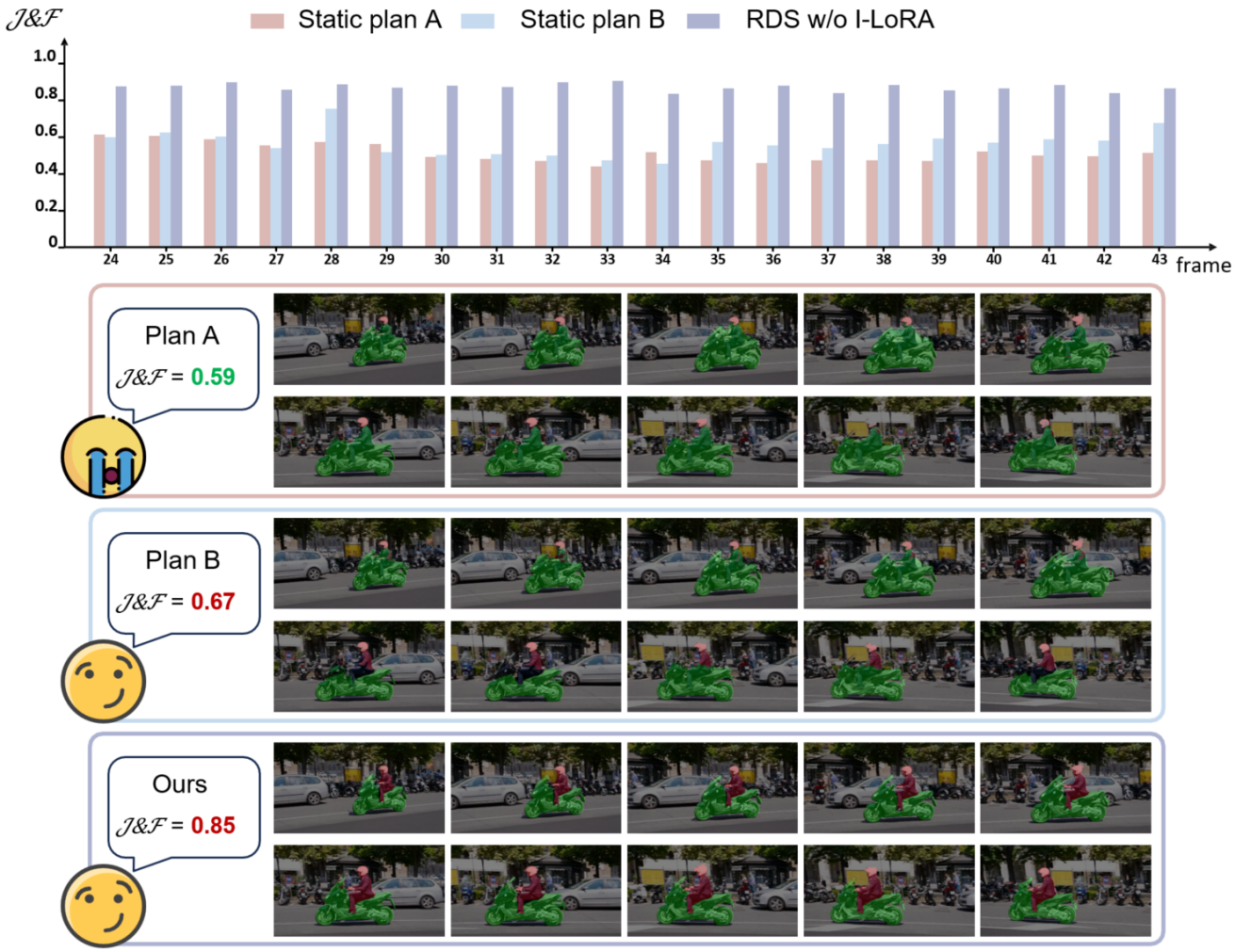


Figure 25. Detailed frame-level visualization comparisons in the *scooter-black* video.

Table 9. **Evaluation of different models on VOS benchmarks.** This table shows the $\mathcal{J}\&\mathcal{F}$, $\mathcal{G}$, and FPS metrics for different model configurations on the DAVIS 2017, SA-V, and YTVOS19 datasets.

Method	$\mathcal{J}\&\mathcal{F}$		$\mathcal{G}$	FPS
	DAVIS 2017	SA-V	YTVOS 2019	
SAM2	90.0	78.5	88.5	29.8
w/ SlimSAM$_{8/9}$	89.5	74.5	86.3	31.1
w/ SlimSAM$_{7/9}$	88.1	70.1	84.1	32.3
w/ SlimSAM$_{6/9}$	87.8	67.9	82.9	33.4
w/ SlimSAM$_{5/9}$	86.1	64.1	81.3	34.4
w/ SlimSAM$_{4/9}$	82.6	57.6	78.8	35.6
w/ PBD$_{0.9}$	90.2	75.5	86.7	32
w/ PBD$_{0.8}$	88.7	72.7	85.5	35.5
w/ PBD$_{0.7}$	87.4	70.1	84.4	36.3
w/ PBD$_{0.6}$	85.6	66.4	83.6	39.8
w/ PBD$_{0.5}$	84.2	62.9	81.9	42.9
w/ PBD$_{0.4}$	81.2	58.1	79.0	46.9
w/ DyT$_{0.9}$	92.6	77.8	88.3	27.8
w/ DyT$_{0.8}$	91.8	77.1	87.9	25.8
w/ DyT$_{0.7}$	90.9	76.0	87.0	24.0
w/ DyT$_{0.6}$	90.2	74.9	86.3	22.4
w/ DyT$_{0.5}$	88.8	72.8	85.1	20.8
w/ DyT$_{0.4}$	87.5	69.1	83.3	19.4
w/ AdaViT$_{0.9}$	91.2	76.0	87.3	29.1
w/ AdaViT$_{0.8}$	90.6	74.7	86.3	28.3
w/ AdaViT$_{0.7}$	89.1	73.9	85.8	27.4
w/ AdaViT$_{0.6}$	88.3	73.1	85.1	26.3
w/ AdaViT$_{0.5}$	86.0	70.4	83.8	25.0
w/ AdaViT$_{0.4}$	83.5	64.5	80.1	23.1
w/ RDS$_{0.9}$	92.0	77.6	88.0	31.4
w/ RDS$_{0.8}$	91.4	76.3	87.6	33.7
w/ RDS$_{0.7}$	90.8	75.6	87.3	35.1
w/ RDS$_{0.6}$	89.6	74.2	86.0	38.2
w/ RDS$_{0.5}$	87.6	70.1	85.2	42.4
w/ RDS$_{0.4}$	84.8	65.4	81.2	46.5