Supplementary Material for NERVE: Neighbourhood & Entropy-guided Random-walk for training free open-Vocabulary sEgmentation

Algorithm 1 Truncation-based Segmentation

```
Require: Maximum number of steps $L$
 1: $P \leftarrow G$
 2: for $t = 1, \dots, L$ do
 3:     $Y \leftarrow 0$
 4:     for $h = 1, \dots, H$ do                ▷ Run the loop in parallel
 5:         $M \leftarrow (K^{(h)})^\top P$
 6:         $M \leftarrow \beta \widetilde{Q}^{(h)} M$
 7:         $M \leftarrow M + (1-\beta) S^{(h)}_{\text{local}} P$
 8:         $Y \leftarrow Y + w_h M$
 9:     end for
10:     $P \leftarrow G + \alpha Y$
11: end for
12: $P_L \leftarrow \frac{1-\alpha}{1-\alpha^{L+1}} P$
13: return $P_L$
```

A. Algorithm Details

The segmentation inference based on our truncated random-walk strategy is detailed in Algorithm 1. At every step t of the random-walk and each attention head h, the algorithm computes the global probabilities (lines 5-6) without having to explicitly construct and store the $N \times N$ node-to-node transition matrix. The local probabilities are then added via a sparse matrix multiplication (line 7). While computations for each head are expressed as a loop, each step of this loop can be performed in parallel.

B. Inverse-based Random-walk Computation

As mentioned in Section 3.2, we can use the fact that low-rankness of A_{global} and sparseness of A_{local} to compute the random walk label probabilities $P^{(\infty)}$ efficiently. Following Eq. (3) and (10), this matrix is computed as

$$\begin{aligned} P^{(\infty)} &= (1-\alpha)\Big(I - \alpha\big(\beta S_{\text{global}} + (1-\beta) S_{\text{local}}\big)\Big)^{-1} G \\ &= \frac{\cdot(\alpha-1)}{\alpha\beta}\big(M + \widetilde{Q}K^T\big)^{-1} G, \end{aligned} \tag{23}$$

where

$$M = \frac{1}{\alpha\beta}\big(\alpha(1-\beta)S_{\text{local}} - I\big), \tag{24}$$

and

$$\widetilde{Q} = \text{Diag}(QK^T\mathbb{1})^{-1} Q. \tag{25}$$

Using the Woodbury matrix inverse identity [14], we can change the computation of $P^{(\infty)}$ to

$$P^{(\infty)} = \frac{\cdot(1-\alpha)}{\alpha\beta}\Big(\widetilde{Q}^* \underbrace{\big(I + K^T\widetilde{Q}^*\big)}_{D\times D}^{-1} K^T - I\Big) G^* \tag{26}$$

where $G^* = M^{-1}G$ and $\widetilde{Q}^* = M^{-1}\widetilde{Q}$. Although matrix inversions are still required, these can now be performed efficiently. Hence, since M is sparse, computing G^* and $\widetilde{Q}^*$ can be done quickly using a sparse linear solver, for example based on LU decomposition [14]. Likewise, the center matrix to invert has a fixed size of $D{\times}D$, independent of N.

C. Additional Ablation Experiments

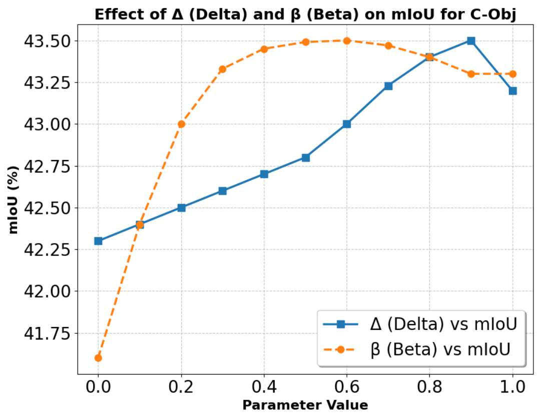

Figure 4. **Ablation on C-Obj dataset**. Effect of varying the random-walk stoppage probability ($\Delta = 1-\alpha$) and the global vs. local weight (β) on segmentation performance (mIoU).

Choice of Hyperparameters. We study the effect of the random-walk stoppage probability ($\Delta = 1$–α) and the global vs. local weight (β) on the C-Obj dataset (Fig. 4). As can be seen, a peak mIoU of 43.5% is reached for a high Δ of 0.9, suggesting that earlier iterations of the random-walk are the most important. Parameter β also has an impact on mIoU performance, rising from 41.6% at $\beta = 0$ to a peak value of 43.5% near $\beta = 0.5$. This demonstrates the usefulness of combining both global and local attention in the random-walk.

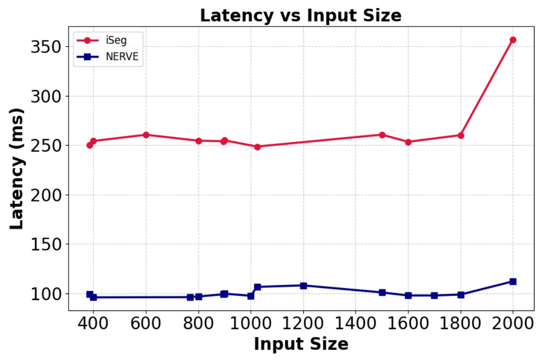

Figure 5. **Latency vs. input size for iSeg and NERVE:** Illustrating the effect of varying image resolution on computational efficiency and runtime behavior.

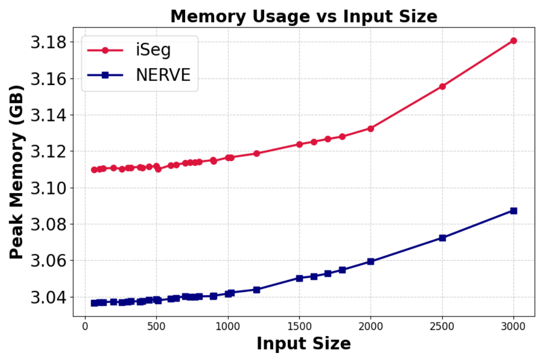

Figure 6. **Memory usage vs. input size for iSeg and NERVE:** Illustrating the effect of varying image resolution on peak GPU memory consumption and scalability.

Computational Efficiency. We benchmarked NERVE against iSeg across a wide range of input sizes (Fig. 5, Fig. 6). In terms of **latency**, NERVE consistently achieves 2–3× speedup, maintaining ∼90–110 ms across resolutions, while iSeg exhibits unstable scaling with spikes exceeding 350 ms at higher resolutions. For **memory**, NERVE starts at 3.04 GB compared to iSeg's 3.11 GB, and the gap widens as resolution increases: at 3K inputs, NERVE uses ∼3.09 GB whereas iSeg exceeds 3.18 GB, corresponding to ∼3% savings in GPU memory. These gains arise from NERVE's hybrid design, where global affinity is represented in a *low-rank* form and local neighborhoods are enforced to be *sparse*, thereby reducing quadratic complexity and avoiding memory blow-up. By contrast, iSeg relies on dense affinity propagation, leading to higher computational cost and scaling variance. Overall, NERVE delivers both *fast inference* and *memory efficiency*, enabling real-time and high-resolution deployment.