Supplementary Materials of “Graph-based Approximate Nearest Neighbor Search by Deep Reinforcement Routing”

A More Details for Parameter Settings

In this section, we provide more details for the parameter settings of the baseline methods (i.e., HNSW, NSG, SSG, τ-MNG, and LTR). We also conduct additional experiments to investigate the parameter sensitivity of the proposed method, confirming the default settings in the main experiments.

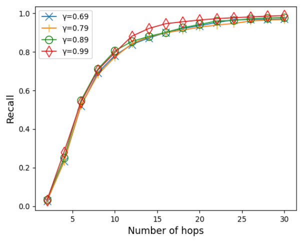
A.1 Baseline Methods

We use the parameters of baseline methods suggested by authors in the original papers to conduct the experiments. Specifically, for the construction of HNSW graph, we set the parameters M=8, M_{max}=16 and $ef_{construction}$=200. M is the maximum out-degree of the vertexes of the layers in graph except the bottom layer. M_{max} is the maximum out-degree of the vertexes of the bottom layer in graph. $ef_{construction}$ is used to control the trade-off between the construction time and graph quality. For the construction of NSG graph, it requires a constructed k-NN graph. As suggested by authors, we use the efanna graph algorithm (Fu and Cai 2016) to build this k-NN graph. Then, we set the parameters L=40, R=50, C=500 to convert the k-NN graph to NSG graph. L is used to control the quality of NSG graph, R controls the index size, and C controls the maximum candidate pool size during NSG construction. For the SSG graph, we construct the k-NN graph using the above efanna graph algorithm. Then, we use the parameter setting of L=100, R=50, $Angle$ = 60 to convert the k-NN graph to SSG graph. L is used to control the quality of SSG graph, and R controls the index size of the graph, where $R < L$. $Angle$ controls the angle between two edges.

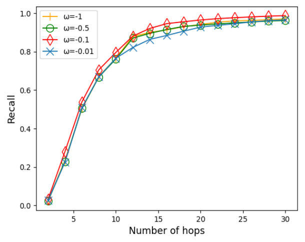
For the τ-MNG graph, as its construction is based on NSG, it also has the three parameters L, R, and C, which are set to be 40, 50, 500. The parameter τ is set as 8, which is used to relax the pruning rule. In terms of the training of LTR method, we set the batch size to be 1024 and use 60,000 iterations to train the LTR model. No dimensionality reduction is performed during the training. The learning rate decays from 0.001 to 0.00001 in 5,000 steps. The maximal number of distance computations (i.e., DCS) during training is set to be 512.

A.2 Our Method

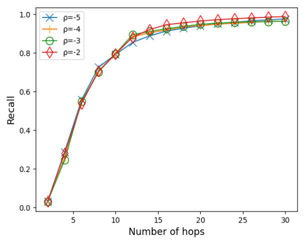
There are three hyper-parameters that needs to be studied for our method, i.e., γ, ω, and ρ. γ is the discount factor of the cumulative reward (Eqs. 1 and 7). ω and ρ are two negative reward values (i.e., penalties) used in the reward function, which are respectively used to handle the cases where the agent consumes a large number of hops or gets trapped in local optima. The experiments are conducted on NSG for Sift100k dataset, where $\gamma = \{0.69, 0.79, 0.89, 0.99\}$, $\omega = \{-1, -0.5, -0.1, -0.01\}$, and $\rho = \{-5, -4, -3, -2\}$. The motivation of ρ is smaller than ω is based on that an agent falling into local optima should receive a larger penalty. The results of recall@1 w.r.t the number of hops are reported in Fig. s1. We can see that our model is not very sensitive to the changes of parameters. The settings of



(a) Parameter γ



(b) Parameter ω



(c) Parameter ρ

Figure s1: The impact of parameters of our method on NSG for Sift100k dataset

Table s1: Training Time and GPU Memory Overhead

Graph	Sift100K				Deep100K			
	Ours		LTR		Ours		LTR	
HNSW	7h	2.9G	11h	2.0G	7h	2.0G	11h	1.9G
NSG	8h	3.2G	12h	2.2G	6h	2.2G	10h	1.9G

$\gamma = 0.99$, $\omega = -0.1$ and $\rho = -2$ perform slightly better than others, which therefore are chosen as the default settings. These settings already enable the model to perform well on all datasets.

B Training Time and GPU Memory Overhead

In this section, we present the training time and GPU memory overhead of our model and the LTR method for a recall of 0.9 on the Sift100K and Deep100K datasets. The results are reported in Table s1. We can see that our training time is 33%−40% less than that of LTR, which confirms the training efficiency of our reinforcement routing model. This is mainly because LTR needs to compute the optimal number of hops (ground truths) of all training queries for model training, while our method avoids. The GPU memory costs of both methods are relatively small, i.e., less than 3GB on most of datasets.

Table s2: Performance Results by using Multi-start Greedy Search on HNSW Graph

The number of starting point	Hops	Sift100K		Deep100K	
		Ours	Original	Ours	Original
2	10	**0.503**	0.315	**0.668**	0.458
	20	**0.849**	0.779	**0.906**	0.828
	30	**0.929**	0.885	**0.939**	0.920
4	10	**0.512**	0.364	**0.693**	0.551
	20	**0.855**	0.789	**0.909**	0.833
	30	**0.934**	0.901	**0.941**	0.922

C Multi-start Greedy Search

In this section, we compare our proposed reinforcement routing with the original routing by using the multi-start greedy search scheme on HNSW graph. Multi-start search is that the algorithm performs the search from multiple starting points separately, and then aggregates the results corresponding to each starting point to generate the final search results. Note that both our method and the original routing perform the search at the bottom layer of HNSW graph. The performance results of Recall@1 with respect to hops budget are reported in Table s2, where the number of starting points is set to be 2 and 4, respectively. The hops budget is applied separately to each starting point for better performance. From the results, we can observe that our method still outperforms the original routing in terms of the multi-start setting. The large margin gap can be seen in the area with small hops budget (i.e., 10). As the hops budget increases, the recall performance of both methods gradually improves. Nevertheless, our method still beats the original routing. The results show that our learned reinforcement model has a strong robustness, and can find the nearest neighbors with fewer hops even in the multi-start setting.