Appendix

A EXTENDED EXPERIMENTAL RESULTS

A.1 Spatial Distributions of the Datasets

The spatial distributions of the MBR center points for all datasets (GAU, Zipf, UNI, CHN, IND) are visualized in Fig. 1. All data coordinates were first linearly normalized to the $[0, 1]$ interval. For the synthetic GAU, Zipf, and UNI datasets, each side of a hyper-rectangle is generated with a length drawn from a uniform distribution $\mathcal{U}[a, 5a]$, meaning the longest side can be at most five times the shortest. The parameter a is configured such that the expected total area of the hyper-rectangles equals half the area of the entire space.

A.2 Effect of Representative Points and Φ

A key improvement of KEAG over heuristic-based R-tree variants and RLR-tree lies in KEAG's incorporation of contextual representative points and candidate representative points as part of its features, along with the intelligent kernel-based point embedder Φ. To investigate the effect of these representative points and Φ, we construct two additional KEAG variants with modified structures. The first variant, KEAG\S, uses only the geometric properties of all candidates as input features, while the second, KEAG\Φ, removes the embedder Φ entirely and feeds normalized points directly into the attention block. We then assess the relative I/O performance of these three KEAG variants on the Cold Start workload of the GAU dataset. The results on the other datasets and other workloads are similar and thus omitted. Table 1 presents the comparative results.

Table 1: Relative I/O cost comparisons of KEAG variants

Method	2M	10M	25M	50M	100M
KEAG\S	0.5979	0.5156	0.4328	0.4735	0.4413
KEAG\Φ	0.5401	0.4758	0.3994	0.4187	0.4025
KEAG	**0.4832**	**0.4324**	**0.3497**	**0.3771**	**0.3626**

Referring to Table 1, KEAG outperforms its two variants across all scenarios, reducing I/O costs by up to 20.4% and 12.4% compared to KEAG\S and KEAG\Φ, respectively. Among all three, KEAG\S performs worst, since it relies solely on geometric properties and omits the information in representative points. While KEAG\Φ achieves better I/O efficiency than KEAG\S, it still falls short of KEAG's performance. Given that the only difference between KEAG and KEAG\Φ is the inclusion of the point embedder Φ, we conclude that the learned embeddings generated by Φ provide a more meaningful representation of spatial relationships than raw points. This enhanced understanding of context-candidate spatial patterns ultimately leads to superior selection decisions in KEAG.

A.3 Effect of Data Dimensionality

To investigate the impact of data dimensionality, we generated multiple GAU datasets with varying dimensions d. For each dataset, we constructed R-tree(L), R-tree(Q), R*-tree, RR*-tree, RLR-tree, and KEAG, and then evaluated their performance on the Cold Start workload. Table 2 presents the relative I/O costs of these five R-tree variants for the 25M case of the GAU dataset of varied dimensionality d. We omit the results on the other datasets for brevity.

Table 2: Relative I/O cost comparisons across different d

d	R-tree(L)	R-tree(Q)	R*-tree	RR*-tree	RLR-tree	KEAG
2	2.7454	1	0.5025	0.4987	0.4976	**0.3497**
3	2.0136	1	0.4833	0.5123	0.4592	**0.3378**
5	2.3105	1	0.3680	0.3833	0.3968	**0.2901**

As shown in Table 2, KEAG consistently maintains the lowest I/O costs across all datasets. R-tree(L) and R-tree(Q) delivers up to 7.96× and 3.45× higher I/O cost than KEAG. Compared to R*-tree, RR*-tree and RLR-tree, KEAG reduces up to 30.4%, 34.1 and 29.7% lower I/O costs, respectively. These results verify the effectiveness of KEAG on candidate selection for higher-dimensional datasets.

A.4 Effect of Distribution Shifting

To investigate if KEAG still works well under significant data distribution shifts, we conduct an experiment with 20M randomly generated records. The first and last 10M records follow different MBR distributions, which we incrementally insert into different R-tree variants. We then measure I/O costs by executing 100K randomly sampled range queries. Notably, KEAG uses the same training data on GAU, Zipf, and UNI datasets as in Section **??** instead of the data from the distribution-shifting insertions. Table 3 shows the relative I/O costs for R-tree(L), R-tree(Q), R*-tree, RR*-tree, RLR-tree, and KEAG across three representative cases. The results for other cases show similar trends and are omitted.

Table 3: Relative I/O cost comparisons *w.r.t.* distribution-shifting insertions

Distribution	R-tree(L)	R-tree(Q)	R*-tree	RR*-tree	RLR-tree	KEAG
GAU-Zipf	2.4245	1	0.4830	0.4765	0.4976	**0.3439**
GAU-UNI	3.1129	1	0.4207	0.4755	0.4503	**0.3378**
Zipf-GAU	2.6213	1	0.3952	0.4092	0.3784	**0.3025**

Referring to Table 3, despite minor performance degradation under distribution-shifting insertions, KEAG still achieves the lowest relative I/O costs among all R-tree variants. Compared to R-tree(L), R-tree(Q), R*-tree, RR*-tree and RLR-tree, KEAG reduces I/O costs by up to 89.1%, 69.8%, 28.8%, 29.0% and 30.9%, respectively. These results indicate KEAG exhibits low sensitivity to the distribution-shifting insertions and remains capable of making appropriate candidate selections for both ChooseSubtree and Split operations, even when the underlying data distribution changes significantly.

A.5 Performance Comparisons on KNN Queries

To evaluate KEAG's effectiveness on another crucial query type—K-Nearest-Neighbor (KNN) queries, we conduct extensive experiments with varying K values ($K \in \{1, 5, 25, 100\}$). For each K setting, we randomly sample 100,000 query points and execute KNN searches across all R-tree variants. In this experiment, the distance between a point $\boldsymbol{p}$ and an MBR $\boldsymbol{B}$ is the minimum Euclidean distance from $\boldsymbol{p}$ to any point within $\boldsymbol{B}$. Table 4 presents the relative

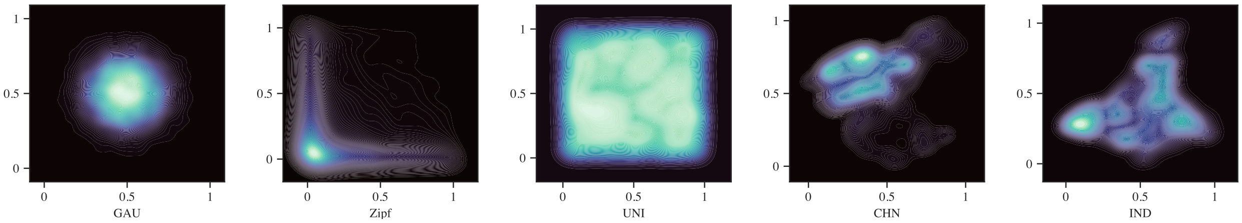

Fig. 1: Spatial distributions of the five datasets

I/O costs for these variants on the 25M GAU dataset case, while results from other datasets are similar and omitted.

Table 4: Relative I/O cost comparisons on KNN queries

K	R-tree(L)	R-tree(Q)	R*-tree	RR*-tree	RLR-tree	KEAG
1	2.8232	1	0.5220	0.5053	0.4810	**0.4195**
5	2.6598	1	0.5182	0.5059	0.4605	**0.3893**
25	2.4016	1	0.5193	0.5933	0.5804	**0.3953**
100	2.1039	1	0.5312	0.5527	0.6392	**0.4035**

Table 4 demonstrates KEAG's effectiveness for KNN queries, despite not being explicitly trained for this query type. Notably, KEAG achieves up to 85.4%, 61.1%, 24.9%, 33.4% and 36.9% lower I/O costs for KNN queries, compared to R-tree(L), R-tree(Q), R*-tree, RR*-tree and RLR-tree. These results suggest that KEAG's ChooseSubtree and Split strategies produce a better R-tree structure, leading to superior performance for both range and KNN queries.

A.6 Performance Comparisons on Spatial Joins

We further evaluate KEAG's efficacy on spatial (self-)joins across different dataset pairs. For a given pair (*e.g.*, GAU and Zipf), we execute the joins using all R-tree variants over a sample of 100,000 query hyper-rectangles. The resulting relative I/O costs for all variants are summarized in Table 5.

Table 5: Relative I/O cost comparisons on spatial joins

Datasets	R-tree(L)	R-tree(Q)	R*-tree	RR*-tree	RLR-tree	KEAG
GAU-Zipf	8.8588	1	0.2635	0.3398	0.2523	**0.2000**
GAU-UNI	6.8084	1	0.2383	0.2727	0.2495	**0.1730**
Zipf-UNI	9.8091	1	0.2606	0.3912	0.2509	**0.1830**
GAU-GAU	7.3417	1	0.2645	0.2517	0.2487	**0.1795**
Zipf-Zipf	11.5583	1	0.2897	0.3433	0.2759	**0.2174**
UNI-UNI	9.7478	1	0.2439	0.3257	0.2430	**0.1702**

As shown in Table 5, KEAG consistently outperforms other R-tree variants in I/O cost for spatial joins. It reduces I/O by up to 98.3%, 83.0%, 32.1%, 53.2% and 30.7% compared to R-tree(L), R-tree(Q), R*-tree, RR*-tree, and RLR-tree, respectively. This consistent advantage underscores how KEAG's refined ChooseSubtree and Split strategies produce a more query-efficient tree structure, directly translating to superior spatial join performance.

A.7 Effect of M and m

To assess the robustness of KEAG under different configurations, we varied the node fanout parameters M and m, which were previously fixed at 50 and 20. We then evaluate the range query performance of different R-tree variants with varying values of M and m. Table 6 presents the relative I/O costs of six R-tree variants for the 25M case of the GAU datasets under different settings for M and m. Results on other datasets are similar and omitted for brevity.

Table 6: Relative I/O cost *w.r.t.* varying M and m

M & m	R-tree(L)	R-tree(Q)	R*-tree	RR*-tree	RLR-tree	KEAG
32, 15	3.1595	1	0.5898	0.5550	0.5710	**0.4320**
50, 16	2.1587	1	0.6804	0.6747	0.5386	**0.4255**
50, 20	2.5744	1	0.5025	0.4977	0.4476	**0.3497**
50, 24	2.9714	1	0.4517	0.4014	0.4053	**0.3300**
64, 30	4.6930	1	0.5017	0.4286	0.4395	**0.3601**
100, 40	2.5989	1	0.4601	0.4299	0.4141	**0.3235**

As shown in Table 6, KEAG consistently achieves the lowest I/O cost across all tested values of M and m. Specifically, it reduces I/O cost by up to 92.3%, 67.0%, 37.5%, 36.9%, 24.3% compared to R-tree(L), R-tree(Q), R*-tree, RR*-tree, and RLR-tree, respectively. This demonstrates KEAG's robustness to different node fanout configurations. The underlying cause is that its ChooseSubtree and Split strategies are capable of making appropriate candidate selections for both large and small nodes.

A.8 Trade-off between Insertion Cost and Tree Quality

The model-based ChooseSubtree and Split operations in KEAG introduce a slight extra insertion overhead compared to conventional R-tree variants. However, this cost is offset by the substantially better tree structure it creates, which leads to considerably lower query I/O costs. To systematically evaluate this trade-off, we conduct an experiment with mixed workloads. We generate five workloads from the GAU dataset, each containing 10M insertions and a different number of range queries, with all operations randomly shuffled. After inserting 10M records into each empty tree, we execute these workloads, measuring the total response time. Table 7 compares the response times of six R-tree variants across five workloads with different insertion-to-query ratios. KEAG incurs the best response time on all workloads. Its performance advantage over other variants amplifies as the query proportion increases, demonstrating that KEAG is particularly well-suited for query-intensive applications.

Table 7: Relative response time *w.r.t.* different insertion-to-query ratios

Insertion-to -query ratio	R-tree(L)	R-tree(Q)	R*-tree	RR*-tree	RLR-tree	KEAG
5 : 1	2.2940	1	0.5934	0.6395	0.6301	**0.5376**
2 : 1	2.4495	1	0.5430	0.5614	0.5566	**0.4334**
1 : 1	2.5095	1	0.5335	0.5303	0.5283	**0.3932**
1 : 2	2.5413	1	0.5232	0.5153	0.5132	**0.3719**
1 : 5	2.5610	1	0.5168	0.5054	0.4939	**0.3587**