EFFICIENT APPROXIMATE VECTOR SEARCH VIA ADAPTIVE RESULT FEEDBACK ON GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph-based approximate nearest neighbor search is a fundamental component of vector retrieval in modern AI systems, such as retrieval-augmented generation for large language models. Efficient search requires adapting exploration to the needs of individual queries, whose results may improve at different rates throughout execution. Conventional search-width and frontier-based controls do not explicitly track this temporal progress, limiting their ability to identify when additional exploration remains productive. We propose FEEDSEARCH, a query-state-aware search strategy that accelerates search on proximity graphs through online result feedback. FEEDSEARCH characterizes a query's evolving search state using the magnitude and temporal history of improvements to its current top-k results. A lightweight, training-free controller inspired by Adam maintains decaying statistics of these improvements and normalizes recent progress against a query-specific scale. This feedback determines how long exploration continues, allowing the strategy to adjust search effort as the query progresses. A lazy implementation skips repeated statistic updates during zero-gain intervals. FEEDSEARCH works with existing graph indexes without modifying or rebuilding them. Experiments across diverse datasets, graph indexes, and similarity measures demonstrate improved recall–throughput trade-offs on a range of workloads.

1 INTRODUCTION

Vector search has become a foundational inference-time primitive for modern AI systems, supporting semantic retrieval (Reimers & Gurevych, 2019; Karpukhin et al., 2020; Khattab & Zaharia, 2020), multimodal retrieval (Radford et al., 2021), recommendation (Covington et al., 2016), and retrieval-augmented generation (Lewis et al., 2020; Borgeaud et al., 2022). As embedding collections grow to millions or billions of high-dimensional vectors, exhaustive nearest-neighbor search becomes prohibitively expensive, motivating approximate nearest neighbor search (ANNS). Graph-based ANNS methods, including HNSW, NSG, and Vamana, are particularly effective in the high-recall regime. They organize database vectors into proximity graphs, enabling queries to be answered efficiently by exploring only a small fraction of the dataset (Malkov & Yashunin, 2020; Fu et al., 2019; Subramanya et al., 2019).

However, these methods typically regulate query-time computation through a search width, such as `efSearch` in HNSW or L in NSG and Vamana. A fundamental limitation of this design is that search difficulty is highly query-dependent: different queries can require substantially different amounts of computation to achieve comparable query quality. Some queries reach a stable, high-quality result after relatively little exploration, whereas others continue to benefit from substantially more work. A fixed width therefore tends to over-explore easy queries while under-exploring difficult ones (Zhang & Miller, 2026). Figure 1 illustrates this mismatch for six DEEP1M queries on the same HNSW index. Each row shows when the top-10 result set changes and when native search stops. For Simple q1 in the top row, all ten nearest neighbors have been found after 141 distance computations, yet search continues to 4,961.

Existing adaptive methods select query-specific search widths (Zhang & Miller, 2026), predict search effort or recall from intermediate states (Li et al., 2020; Chatzakis et al., 2025), or use a distance-based termination condition (Al-Jazzazi et al., 2025). This motivates a search strategy that regulates exploration using the temporal evolution of the retrieved results.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

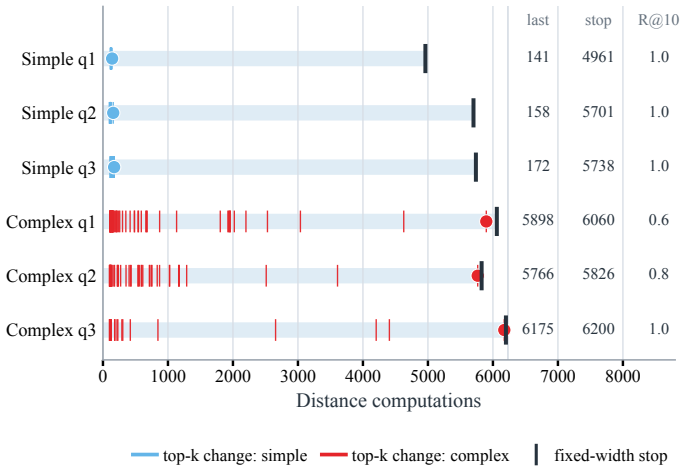


Figure 1: Top-10 result changes for six DEEP1M queries on one HNSW index with `efSearch` $= 400$. Colored ticks mark result changes, circles mark the last change, and black ticks mark native termination. R@10 is recall at termination.

A simple starting point is to count how many distance computations have occurred since the current top-k result set last changed. We call this number the *stagnation length*. Prior work such as PiP also uses result-collection saturation to decide when to stop (Teofili & Lin, 2025). A *stagnation counter* terminates search when this number reaches a fixed threshold. Although all queries share the threshold, they reach it after different amounts of search work. For Simple q1 in Figure 1, replaying a counter with a threshold of 100 on the recorded search stops at 241 distance computations with the same exact answer, saving 95.1% of the computations.

However, every result change resets the counter, whether the improvement is large or small. The reset also discards earlier progress and is therefore myopic. A small, isolated improvement therefore permits as much further search as a large improvement following many others. We need a measure that accounts for both the size and history of improvements.

To address this issue, we propose FEEDSEARCH, a query-state-aware search strategy based on online result feedback. A lightweight, training-free controller inspired by Adam (Kingma & Ba, 2015) averages recent improvements with exponential decay and divides this average by a decaying reference value for the query. Search terminates when the resulting score falls below a threshold. The score reflects the size and frequency of recent improvements and decreases during computations without improvement. Previous improvements therefore determine how long the search continues.

FEEDSEARCH can be applied to existing HNSW, NSG, and Vamana indexes without modifying or rebuilding them. It changes query processing while reusing the stored vectors and graph edges. Our analysis gives a sufficient condition for FEEDSEARCH to continue while the results improve, and bounds the additional distance computations once improvements cease (Section 4.1). We also construct graphs, including one where every point except the nearest neighbor has a neighbor closer to the query, on which FEEDSEARCH returns the exact nearest neighbors with fewer distance computations than the original HNSW and Vamana searches (Sections 4.2–4.3). Section 5.4 validates these cost comparisons on the constructed graphs using the HNSW and Vamana search implementations. Our experiments compare search methods on the same indexes and show improved recall–throughput trade-offs across a range of workloads (Section 5).

Our contributions are threefold:

- We introduce FEEDSEARCH, a query-state-aware strategy that uses the magnitude and history of top-k improvements to reduce query cost on existing graph indexes, without index reconstruction.
- We develop a training-free, Adamax-inspired feedback controller with lazy updates for efficient execution.

- We analyze the termination condition and give graph constructions where FEEDSEARCH uses fewer distance computations at the same recall.

2 BACKGROUND AND RELATED WORK

2.1 GRAPH SEARCH AND SEARCH WIDTH

Approximate nearest neighbor search (ANNS) retrieves vectors close to a query under certain distance functions. Quantization compresses distance computation (Jégou et al., 2011; Ge et al., 2013); proximity graphs reduce the number of vectors visited (Malkov & Yashunin, 2020; Fu et al., 2019; Subramanya et al., 2019). We study search on an already built graph.

The search loop. Graph search keeps a candidate queue C and the best k results found so far, R. Starting from an entry vertex, it repeatedly takes the closest eligible candidate, evaluates its unseen neighbors, updates R, and retains candidates for later expansion. It returns R when the stopping rule fires or no eligible candidate remains. Algorithm 1 makes these steps explicit.

Algorithm 1: Graph Search with a Termination Policy

Input: Graph G, query q, entry s, result size k, candidate-retention rule A, stopping rule P

Evaluate $d(q, s)$; $C \leftarrow \{s\}$; $R \leftarrow \{s\}$; $D \leftarrow \{s\}$;
while *C has an eligible unexpanded candidate under A* **do**
  Remove the closest eligible vertex v from C;
  foreach *neighbor u of v with $u \notin D$* **do**
    Evaluate $d(q, u)$; $D \leftarrow D \cup \{u\}$;
    $R \leftarrow \mathrm{Top}_k(R \cup \{u\})$;
    Update candidate state with u according to A;
    if *P signals stop after observing the updated state* **then return** R;
return R;

The role of search width. Native HNSW retains an internal pool of at most `efSearch` vertices; NSG and Vamana expose a corresponding list size L. These pools retain search alternatives, including already expanded vertices, and are distinct from the returned top-k answer. HNSW stops when the closest queued candidate lies beyond its retained boundary; Vamana stops when its retained list has no unchecked member. Appendix C gives both procedures and proves the common retained-width interpretation under matched search conditions. Width is not a fixed number of distance computations. It controls which alternatives survive, without explicitly measuring the temporal progress of the returned answer.

2.2 ADAPTIVE SEARCH AND ONLINE FEEDBACK

Adaptive learned search algorithms. Adaptive search estimates the effort a query needs or observes its progress online. Ada-ef calibrates query-specific widths (Zhang & Miller, 2026); LAET and DARTH predict search effort and recall, respectively (Li et al., 2020; Chatzakis et al., 2025). Training-free methods use the search state directly: DABS compares frontier and result distances (Al-Jazzazi et al., 2025), PiP detects result-set saturation (Teofili & Lin, 2025), and Terminus measures rank-weighted result gains per I/O (Lu et al., 2026). Adamax provides a complementary idea: normalizing a moving average by a decayed maximum (Kingma & Ba, 2015). Table 1 compares the timed baselines. These choices leave two issues: adaptation cost and the information captured by the stopping signal. Calibration and learned predictors require offline work; prediction also adds query-time overhead. Current-distance tests omit improvement history, while membership and rank changes can treat small and large distance improvements alike. FEEDSEARCH combines improvement magnitude and history without predictor training. It adapts Adamax's average and scale to result gains, using their ratio to decide when to stop. This lets recent improvements determine how much further exploration is allowed.

Table 1: Adaptive control mechanisms, excluding inherited native stopping rules: ✓ denotes presence and ✗ absence. LAET predicts search effort and DARTH predicts recall (HNSW variants). Ada-ef requires statistics and calibration despite having no trained termination predictor. Temporal gains refer to the explicit history of top-k potential improvements, rather than intermediate-result features in general.

Method	Adaptive search width	Learned stopping	Geometric stopping	Temporal gain feedback	No trained predictor
Ada-ef (Zhang & Miller, 2026)	✓	✗	✗	✗	✓
LAET (Li et al., 2020)	✗	✓	✗	✗	✗
DARTH (Chatzakis et al., 2025)	✗	✓	✗	✗	✗
DABS (Al-Jazzazi et al., 2025)	✗	✗	✓	✗	✓
FEEDSEARCH	✗	✗	✗	✓	✓

3 FEEDSEARCH: ADAPTIVE SEARCH VIA RESULT FEEDBACK

FEEDSEARCH uses the search loop in Algorithm 1 with unrestricted candidate retention and a stopping rule driven by top-k progress. We first define result gains and then use adaptive moments to obtain a normalized feedback score.

3.1 RESULT FEEDBACK

Search state. Let $G = (V, E)$ be the fixed graph, q the query, C the candidate min-heap, and R_t the best k discovered vertices. Every unseen neighbor enters C, including those that do not improve R_t. The clock t starts at zero when R first fills and advances once per subsequent new distance evaluation. We use a nonnegative ranking value $d(q, v)$: squared L2, $1 - \cos(q, v)$, or $B(q) - q^\top v$ for MIPS, where a query-fixed norm bound $B(q)$ with a numerical margin makes the values nonnegative.

From identity changes to improvement magnitude. If ℓ_t is the most recent evaluation that changed the top-k identities (initially $\ell_0 = 0$), the *stagnation length* $a_t = t - \ell_t$ measures work since that change. A stagnation counter stops at $a_t \geq \tau$ and resets to zero after every identity change (Figure 2). To distinguish large and small updates, we instead measure the decrease in the mean log-distance:

$$\theta_t = \frac{1}{k} \sum_{v \in R_t} \log \max\{d(q, v), \delta\}, \qquad \delta = 10^{-30}, \tag{1}$$

$$g_t = \max\{0, \theta_{t-1} - \theta_t\}. \tag{2}$$

Each evaluation replaces at most one result, so its gain is obtained in constant time:

$$g_t = \frac{\log \max\{d_{\text{out}}, \delta\} - \log \max\{d_{\text{in}}, \delta\}}{k}, \tag{3}$$

where d_{out} is the distance from q to the result being replaced, and d_{in} is the distance from q to its replacement. If no replacement occurs, $g_t = 0$. Equal-distance and floor-clipped replacements also have zero gain. Thus a zero-gain interval need not coincide with an interval of unchanged result identities. The logarithm measures relative improvement: halving a distance gives the same gain at different scales when the floor is inactive. The MIPS offset is part of this feedback definition, even though it does not affect the ranking.

3.2 NORMALIZED RESULT FEEDBACK VIA ADAPTIVE MOMENTS

The gains in Section 3.1 quantify individual result improvements. We now use their history to decide when to stop. A single zero gain can occur between useful updates, so we average recent gains to measure sustained progress. Since gain magnitudes differ across queries, we normalize this average by a reference scale from the same query. The resulting score measures how strong recent progress is relative to earlier improvements. Following Adamax (Kingma & Ba, 2015), we maintain

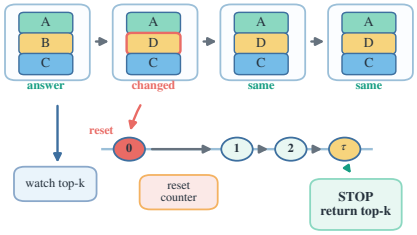


Figure 2: A stagnation counter resets after each top-k identity change, regardless of improvement magnitude.

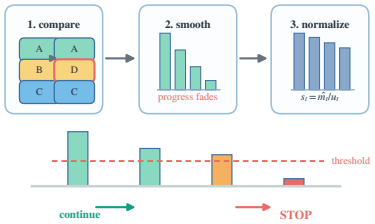


Figure 3: Adaptive result feedback smooths gains and normalizes them by a decaying query-local scale to decide when to stop.

an exponential moving average m_t of the gains and a decayed maximum u_t as their reference scale. The average smooths intermittent improvements, while the reference gradually discounts old large gains. Each query starts with $m_0 = u_0 = 0$, before any gain history has been collected, and updates these quantities as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{4}$$

$$u_t = \max\{\beta_2 u_{t-1}, g_t\}. \tag{5}$$

The decay factors β_1 and β_2 control how quickly past gains lose weight. The factor $1 - \beta_1^t$ corrects the average's zero initialization, giving the *feedback score*:

$$s_t = \frac{m_t}{(1 - \beta_1^t) u_t}, \qquad u_t > 0. \tag{6}$$

We use $\beta_1 = 0.9$ and $\beta_2 = 0.99$ by default. Without new improvements, m_t decays faster than u_t, so the score falls. New gains contribute to the average without discarding earlier improvements. The gain history therefore determines how long search continues through subsequent computations without improvement (Figure 3).

Stopping rule. After each feedback observation, we stop when the score falls below a threshold:

$$s_t < \epsilon, \qquad \epsilon = 10^{-\Lambda}. \tag{7}$$

One Λ is shared by all queries at an operating point; their different gain histories yield different stopping times. Larger Λ permits longer exploration. Stopping takes effect immediately, possibly within a neighbor list. The score is relative progress, not a recall probability, and can exceed one. Before the first positive gain, $u_t = 0$; a separate safeguard stops after more than H_0 consecutive zero-gain evaluations, with $H_0 = 10{,}000$ in the implementation. Only a positive gain resets this counter. Frontier exhaustion also ends search.

Computational cost. The controller uses $O(1)$ state; its direct recurrence takes constant work per observation. A lazy implementation computes the next zero-gain stopping deadline, then advances counters until that deadline or the next positive gain; each deadline search takes $O(\log(1 + J_\epsilon))$ steps, where J_ϵ is bounded in Eq. 10. Appendix A gives the equivalence argument. For D discovered vertices and P top-k replacements, heap work is $O(D \log D + (k+P) \log k)$, in addition to distance computation and adjacency scanning. Unrestricted retention can use $O(D)$ frontier space; the result heap uses $O(k)$, and visited-state storage depends on the index implementation.

4 Theoretical Analysis

We analyze FeedSearch's termination condition and query cost. We measure cost by the number of query-to-point distance computations. We assume exact arithmetic, distinct query distances for all points, and $d(q, v) \geq \delta$, so the lower bound in Eq. 1 does not change the distances. Let $\rho = \beta_1/\beta_2$, $0 < \beta_1 < \beta_2 < 1$, and $\epsilon \in (0, 1]$. Proofs are in Appendix A.

4.1 TERMINATION CONDITION

An improvement means that the mean log-distance of the current results decreases, giving $g_t > 0$.

Proposition 1 (Continuing while results improve)**.** *Consider the first T feedback steps of* FEEDSEARCH *with early termination disabled, ending in an improvement. Suppose the first improvement occurs by step $H_0 + 1$, where H_0 is the limit before the first improvement in Section 3.2. In these steps, every positive gain lies in $[g_{\min}, g_{\max}]$, with $g_{\min} > 0$, and at most h distance computations occur without improvement between two consecutive improvements. If*

$$\epsilon \leq (1 - \beta_1) \frac{g_{\min}}{g_{\max}} \rho^h, \tag{8}$$

FEEDSEARCH *completes these T steps.*

Proposition 2 (Stopping after results stop improving)**.** *Let $M = \beta_2(1 - \beta_1)/(\beta_2 - \beta_1)$. After at least one improvement, $u_t > 0$ and $0 < s_t \leq M$. If the next r distance computations give no improvement, then*

$$s_{t+r} = s_t \rho^r \frac{1 - \beta_1^t}{1 - \beta_1^{t+r}}. \tag{9}$$

The score strictly decreases, so FEEDSEARCH *terminates after at most*

$$J_\epsilon = 1 + \left\lfloor \frac{\log(M/\epsilon)}{\log(\beta_2/\beta_1)} \right\rfloor \tag{10}$$

further computations without improvement. It may terminate earlier if no candidates remain.

If the search in Proposition 1 finds the exact k nearest neighbors after $C^\star$ distance computations, Proposition 2 bounds its total cost by $C^\star + J_\epsilon$. A long period without improvement alone does not imply that all nearest neighbors have been found. Equation 9 also lets us skip repeated score updates without changing when the search terminates (Appendix A.2).

4.2 QUERY COST AFTER FINDING THE NEAREST NEIGHBORS

Monotonic search networks (MSNETs) provide a path that gets closer to the destination at each step for every pair of graph vertices Fu et al. (2019). Here we require this property only for the current query: every point except its nearest neighbor has a neighbor closer to the query. Even on such a graph, HNSW and Vamana can continue checking points after finding the exact results. We compare HNSW's base-layer search and Vamana's search with FEEDSEARCH on the same graph and from the same entry point. Let B denote HNSW's `efSearch` or Vamana's L. Both algorithms run until their own termination conditions are met (Appendix C). In the example below, the entry point connects to all k nearest neighbors, and each of them connects to h additional, farther points.

Proposition 3 (Fewer distance computations at the same recall)**.** *For integers $k, h \geq 1$ with $kh > J_\epsilon$, there is a connected, undirected graph satisfying the above condition, with at most $\max\{k, h + 3\}$ neighbors per point, such that:*

1. *HNSW and Vamana, for every $B \geq k$, return Recall@$k = 1$ with $1 + k + kh$ distance computations;*
2. FEEDSEARCH *returns Recall@$k = 1$ with at most $1 + k + J_\epsilon$ distance computations.*

The exact results are found first. HNSW and Vamana then check all kh additional points, whereas FEEDSEARCH checks at most J_ϵ. Appendix A.3 gives the complete graph. Section 5.4 verifies this cost comparison as h increases (Figure 6(a)).

4.3 COMPARISON WITH SEARCH WIDTH

A small search width can discard a point needed to reach a nearest neighbor. A larger width can keep that point, but also cause more work after the nearest neighbor has been found.

Proposition 4 (Lower query cost than any successful search width)**.** *For $\epsilon \leq (1 - \beta_1)\rho$ and any integer $n > J_\epsilon$, there exists a connected graph, with at most two neighbors per point, connecting*

two points exactly when their Euclidean distance is at most a fixed threshold, such that, for the same entry point and $k = 1$: FEEDSEARCH has recall one using at most $4 + J_\epsilon$ distance computations; HNSW and Vamana have recall zero and cost 3 for $B \in \{1, 2\}$, and recall one and cost $n + 4$ for every $B \geq 3$. Thus FEEDSEARCH saves at least $n - J_\epsilon$ distance computations at recall one.

Here n counts the farther points on a path beyond the nearest neighbor; Appendix A.3 gives the graph. Section 5.4 verifies these costs over the tested values of n (Figure 6(b)). These comparisons use the original termination conditions without a separate computation limit. They establish savings from stopping according to result improvements on the constructed graphs.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate FEEDSEARCH against native graph search implementations across various datasets and graph indexes. Experiments use four Intel Xeon Gold 6248 CPUs (80 physical cores, 160 hardware threads) with Ubuntu 22.04.5 LTS. Eight datasets cover L2, cosine, and maximum-inner-product search on HNSW, NSG, and Vamana: 24 graph–dataset workloads (Table 2). FEEDSEARCH and native search share each built index (Table 3); we vary only their query-time controls.

Queries and predictor training. All methods use the same fixed queries. LAET and DARTH use five-fold cross-fitting: each query is evaluated by a predictor trained on the other four folds, so no evaluation query enters its corresponding model's training data. Ada-ef samples 200 database vectors with a fixed seed, disjoint from the evaluation queries; FEEDSEARCH trains no predictor.

Table 2: Evaluation datasets. M denotes one million vectors.

Metric	Dataset	Dimensions	Vectors	Queries
L2	DEEP1M	256	1M	1,000
L2	SIFT10M	128	10M	1,000
L2	GIST1M	960	1M	1,000
L2	Tiny5M	384	5M	1,000
MIPS	YahooMusic	300	0.625M	200
MIPS	Yandex10M	200	10M	1,000
Cosine	DBpedia1M	1,536	1M	1,000
Cosine	OpenAI / BEIR NQ	1,536	2.68M	3,452

Table 3: Index construction parameters.

Dataset	HNSW $(M, \texttt{efConstruction})$	NSG $(R, k_G, L_{\text{build}}, C)$	Vamana $(R, L_{\text{build}}, \alpha_{\text{V}})$
DEEP1M	$(12, 120)$	$(50, 200, 60, 500)$	$(64, 100, 1.2)$
GIST1M	$(16, 200)$	$(70, 400, 60, 500)$	$(96, 200, 1.2)$
Other datasets	$(16, 200)$	$(64, 200, 60, 500)$	$(100, 150, 1.2)$

Accuracy–efficiency trade-off. The most important evaluation criteria for vector search algorithms are accuracy and efficiency. We use recall–QPS and recall–COMPS curves with $k = 10$ to evaluate the accuracy–efficiency trade-off of different vector search algorithms. Recall@10 is the mean fraction of exact top-10 neighbors retrieved. QPS measures wall-clock queries per second; COMPS counts mean distance or similarity evaluations per query, including HNSW upper layers. At equal recall, higher QPS and lower COMPS are better; at equal cost, higher recall is better. QPS includes implementation overhead, while COMPS isolates search computation (Aumüller et al., 2020; Simhadri et al., 2022).

Baselines. We additionally compare with other adaptive learned search algorithms, including Ada-ef, DARTH, LAET, and DABS implementations on HNSW under L2 distance with their default setting. Ada-ef uses an L2 adaptation with per-target calibration; LAET retains its original nearest-neighbor work-prediction target. The six-method comparison uses 160 persistent CPU-pinned workers, ten repetitions per query in each of three rounds, and median wall-clock QPS.

5.2 Performance Across Graph Indexes

Figure 4 compares FeedSearch with native HNSW, NSG, and Vamana on all 24 fixed-index workloads using smoothed recall–QPS and recall–COMPS curves. It shows that higher QPS often accompanies fewer distance computations. For example, when first meeting Recall@10 ≥ 0.95, FeedSearch increases QPS from 43.3K to 53.8K on NSG/SIFT10M while reducing distance computations by 19.5%. On HNSW/OpenAI, it increases QPS from 18.5K to 22.4K with 17.9% fewer distance computations. These results show that FeedSearch can reduce the search work needed to meet the recall requirement. This reduction is consistent with the purpose of result feedback. When recent result improvements become small or infrequent, FeedSearch can stop without waiting for the native termination condition. Continued improvements allow further exploration. It therefore adjusts search effort according to the progress of each query. Since both methods use the same graph index, these improvements require no index modification or reconstruction.

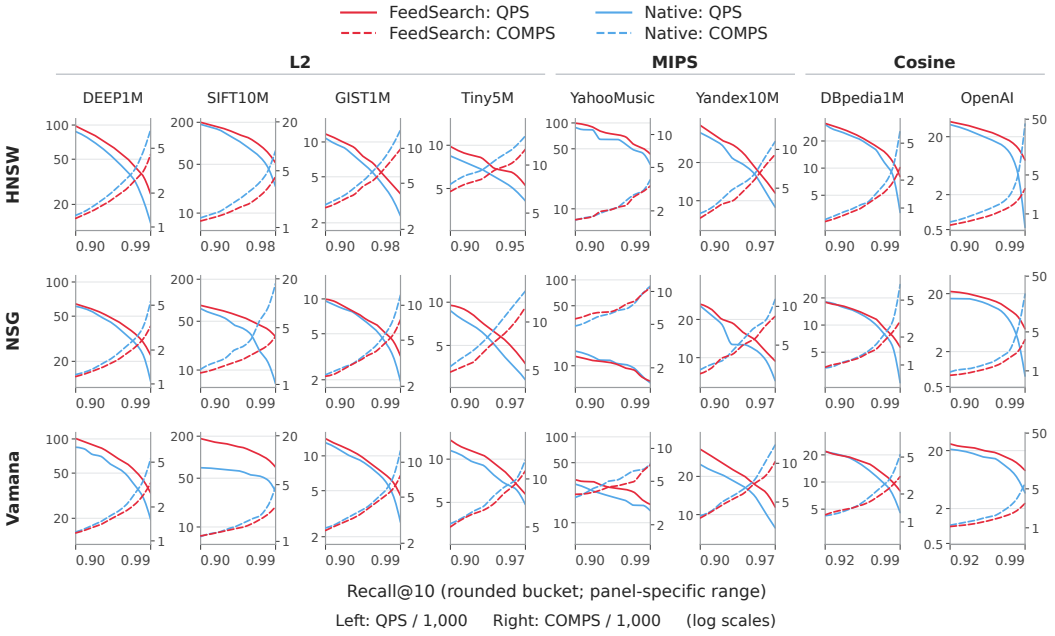


Figure 4: Recall–efficiency trade-offs on 24 fixed indexes. Rows: HNSW, NSG, Vamana; columns: datasets grouped by similarity. Solid: QPS; dashed: COMPS (both logarithmic).

5.3 Comparison with Adaptive Learned Search Algorithms

We next compare FeedSearch with Native HNSW, Ada-ef, DARTH, LAET, and DABS on the four L2 workloads. Figure 5 shows the four highest fully covered recall targets between 0.90 and 0.99 for each dataset. FeedSearch achieves the highest plotted QPS in all four recall targets on SIFT10M. In the 0.97 target, it processes approximately 27.3K queries per second, compared with 20.9K for DABS and 17.5K for Ada-ef. The corresponding distance counts help explain the performance difference from LAET and DARTH. At the 0.97 target on SIFT10M, FeedSearch performs approximately 2,788 distance computations per query, compared with 5,068 for LAET and 8,609 for DARTH. Its higher throughput is therefore accompanied by less search work. FeedSearch uses the magnitude and frequency of observed result improvements to decide when to stop, allowing it to respond to the progress of the current query without a trained predictor.

5.4 Validation of the Query-Cost Analysis

We test the constructions in Propositions 3 and 4 using FeedSearch, hnswlib's base-layer search, and DiskANN's Vamana search. Each constructed graph has one query; all methods share its vectors, edges, entry point, and neighbor order. These graphs are prescribed by the analysis rather

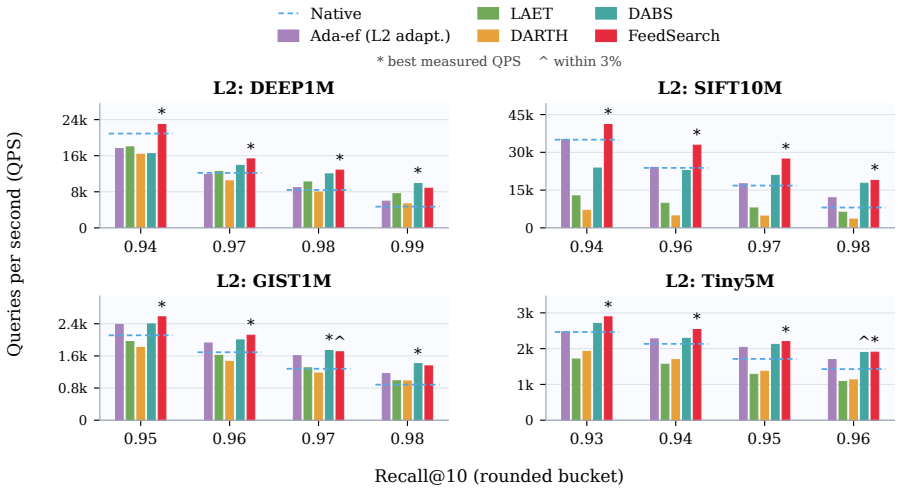


Figure 5: Six-method HNSW/L2 comparison: three-round median QPS.

than built by the indexing algorithms. We use squared Euclidean distances, the default betas, and $\epsilon = 0.01$, and sweep every integer native width $B = k, \ldots, |V|$.

Figure 6 shows the measured query cost. On the first construction with $k = 10$, all methods discover the exact results after 11 distance computations. At $h = 64$, HNSW and Vamana continue for 640 more computations, whereas FEEDSEARCH stops after 25 more, giving totals of 651 and 36. Its cost stays at 36 over the tested h. On the two-dimensional radius graph, widths $B = 1, 2$ miss the nearest neighbor; every $B \geq 3$ finds it but uses $n + 4$ computations. FEEDSEARCH returns the exact neighbor in 35 computations at every tested n. These measurements agree with the predicted savings from stopping after result improvements cease.

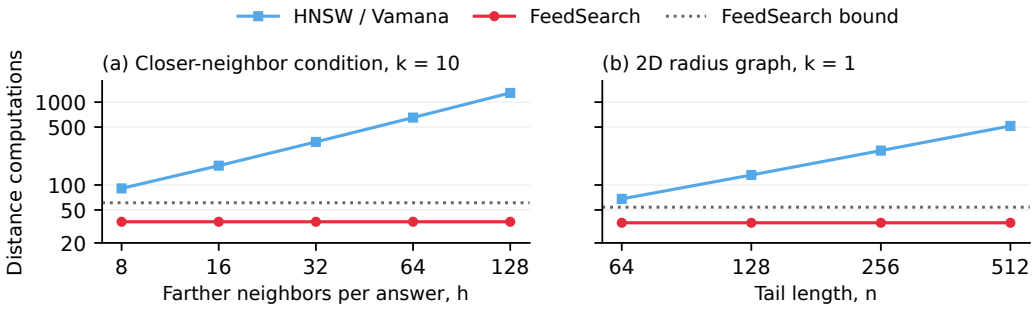


Figure 6: Query cost on the two graph constructions (logarithmic axes). All plotted searches attain Recall@$k = 1$; both native methods have identical costs at every successful width. Dotted lines show FEEDSEARCH's theoretical upper bounds.

6 CONCLUSION

We proposed FEEDSEARCH, a query-state-aware strategy that accelerates nearest-neighbor graph search through training-free, adaptive result feedback. Its Adam-inspired controller uses the magnitude and history of top-k improvements to regulate search effort, improving recall–efficiency trade-offs across diverse graph indexes and similarity settings. At a common minimum Recall@10 of 0.95, throughput gains over native search range from 5% to 40%. It works with existing indexes without modifying or rebuilding them, so these gains require no additional overhead.

AI USE STATEMENT

The authors developed the core idea and design. Generative AI assisted with figures, literature and citation checking, polish, and experimental settings and running. Results come from recorded measurements; the authors are responsible for verifying all claims, references, and results.

REPRODUCIBILITY STATEMENT

The FEEDSEARCH core and HNSW, NSG, and Vamana adapters are available at `https://github.com/FeedSearch2027ICLR/FeedSearch`.

REFERENCES

Yousef Al-Jazzazi, Haya Diwan, Jinrui Gou, Cameron Musco, Christopher Musco, and Torsten Suel. Distance adaptive beam search for provably accurate graph-based nearest neighbor search. In *Advances in Neural Information Processing Systems*, volume 38, pp. 100119–100148. Curran Associates, Inc., 2025. doi: 10.52202/085713-3346. URL `https://proceedings.neurips.cc/paper_files/paper/2025/file/904fe070f484231aa26dbdb37816cd40-Paper-Conference.pdf`.

Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020. doi: 10.1016/j.is.2019.02.006. URL `https://doi.org/10.1016/j.is.2019.02.006`.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 2206–2240, 2022. URL `https://proceedings.mlr.press/v162/borgeaud22a.html`.

Manos Chatzakis, Yannis Papakonstantinou, and Themis Palpanas. DARTH: Declarative recall through early termination for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data*, 3(4), September 2025. doi: 10.1145/3749160. URL `https://helios2.mi.parisdescartes.fr/~themisp/publications/sigmod26-darth.pdf`.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for YouTube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 2016. URL `https://research.google/pubs/deep-neural-networks-for-youtube-recommendations/`.

Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment*, 12(5):461–474, 2019. doi: 10.14778/3303753.3303754. URL `https://www.vldb.org/pvldb/vol12/p461-fu.pdf`.

Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2946–2953, 2013. URL `https://openaccess.thecvf.com/content_cvpr_2013/html/Ge_Optimized_Product_Quantization_2013_CVPR_paper.html`.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. doi: 10.1109/tpami.2010.57. URL `https://doi.org/10.1109/tpami.2010.57`.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*

(EMNLP), pp. 6769–6781. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 39–48, 2020. doi: 10.1145/3397271.3401075. URL https://doi.org/10.1145/3397271.3401075.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. Improving approximate nearest neighbor search through learned adaptive early termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 2539–2554, 2020. doi: 10.1145/3318464.3380600. URL https://www.pdl.cmu.edu/PDL-FTP/BigLearning/mod0246-liA.pdf.

Jianan Lu, Asaf Cidon, and Michael J. Freedman. When enough is enough: Rank-aware early termination for vector search. In *Proceedings of Machine Learning and Systems*, volume 8, 2026. URL https://www.cs.princeton.edu/~mfreed/docs/terminus-mlsys26.pdf.

Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020. doi: 10.1109/TPAMI.2018.2889473. URL https://arxiv.org/abs/1603.09320.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. Results of the NeurIPS'21 challenge on billion-scale approximate nearest neighbor search. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, volume 176, pp. 177–189, 2022. URL https://proceedings.mlr.press/v176/simhadri22a.html.

Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, and Rohan Kadekodi. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Abstract.html.

Tommaso Teofili and Jimmy Lin. Patience in proximity: A simple early termination strategy for HNSW graph traversal in approximate k-nearest neighbor search. In *Advances in Information Retrieval (ECIR)*, volume 15574 of *Lecture Notes in Computer Science*, pp. 401–407. Springer, 2025. doi: 10.1007/978-3-031-88714-7_39. URL https://cs.uwaterloo.ca/~jimmylin/publications/978-3-031-88714-7_39.pdf.

Chao Zhang and Renée J. Miller. Distribution-aware exploration for adaptive HNSW search. *Proceedings of the ACM on Management of Data*, 4(1), 2026. doi: 10.1145/3786639. URL https://arxiv.org/abs/2512.06636.

A Proofs and Scope of the Feedback Analysis

Throughout this appendix, the graph, query, initial entry state, adjacency order, and tie-breaking rule are fixed. We assume exact arithmetic, $m_0 = u_0 = 0$, $g_t \geq 0$, and $0 < \beta_1 < \beta_2 < 1$. Finite-precision underflow, logarithmic clipping, and differences between distance kernels require implementation checks beyond these identities.

A.1 Further Properties of Result Feedback

The following statements explain how the signal responds to scale, small improvements, and the stopping threshold. They hold under the exact-arithmetic controller assumptions in Section 4; the graph, entry, adjacency order, and tie-breaking remain fixed.

Proposition 5 (Scale invariance)**.** *Multiplying all gains by any $c > 0$ preserves every feedback score and stopping decision. In particular, if $d' = ad^p$ with $a, p > 0$ and the logarithmic floor is inactive in both representations, the traversal and stopping decisions are unchanged.*

This includes Euclidean versus squared Euclidean ranking distances when the logarithmic floor is inactive. Additive shifts, such as changing a MIPS offset, are not covered, and the result does not imply threshold transfer across datasets.

Proposition 6 (A small improvement produces a partial response)**.** *Fix a state with $u_t > 0$. If the next gain satisfies $0 \leq g \leq \beta_2 u_t$, so that it does not replace the decayed reference scale, then*

$$s_{t+1}(g) = \rho \frac{1-\beta_1^t}{1-\beta_1^{t+1}} s_t + \frac{(1-\beta_1)g}{\beta_2 u_t (1-\beta_1^{t+1})}. \tag{11}$$

For gains below the decaying reference scale, the extra response is proportional to g/u_t and tends to zero with the improvement magnitude. Together with Proposition 2, this characterizes the controller's response to improvement magnitude and zero-gain intervals.

Threshold monotonicity. Candidate admission does not depend on Λ. With all other search and controller settings fixed,

$$\Lambda_1 < \Lambda_2 \quad \Longrightarrow \quad \mathrm{Trace}_{\Lambda_1} \preceq \mathrm{Trace}_{\Lambda_2}, \tag{12}$$

Here $\preceq$ means that the shorter search checks the same points in the same order as the beginning of the longer search. Continuing the search cannot increase any of the sorted top-k distances. Recall is also nondecreasing under a ground-truth tie convention consistent with the search; this does not provide a numerical recall bound. Proofs of the statements in this subsection are in Appendix A.

A.2 Proofs of the Controller Properties

Proof of Proposition 5. Under $g'_t = cg_t$, induction in Eqs. 4–5 gives $m'_t = cm_t$ and $u'_t = cu_t$. Thus $s'_t = s_t$ whenever $u_t > 0$. The locations of zero and positive gains also coincide, preserving the zero-scale safeguard. If $d' = ad^p$ and neither representation activates the floor, then $\log d' = \log a + p \log d$. Consequently, $\theta'_t = \log a + p\theta_t$ and $g'_t = pg_t$. This transformation is strictly increasing, so it preserves distance order and ties. Coupled with the same entry state and adjacency order, it preserves both traversal and termination. The statement concerns representations of a fixed graph; rebuilding an index using a transformed distance is outside its scope. □

Proof of Proposition 2. Unrolling the recurrences yields

$$m_t = (1-\beta_1) \sum_{j=0}^{t-1} \beta_1^j g_{t-j}, \qquad u_t = \max_{0 \leq j < t} \beta_2^j g_{t-j}.$$

Since $g_{t-j} \leq u_t \beta_2^{-j}$ and $u_t > 0$,

$$\begin{aligned} s_t &\leq \frac{1-\beta_1}{1-\beta_1^t} \sum_{j=0}^{t-1} (\beta_1/\beta_2)^j \\ &= M \frac{1-\rho^t}{1-\beta_1^t} \leq M. \end{aligned}$$

The last inequality uses $\beta_1 < \rho < 1$. A positive gain gives both $u_t > 0$ and $m_t > 0$ at every subsequent finite step in exact arithmetic, so $s_t > 0$.

If the next r gains are zero, repeated substitution gives $m_{t+r} = \beta_1^r m_t$ and $u_{t+r} = \beta_2^r u_t$. Substitution into Eq. 6 proves Eq. 9. For consecutive steps in this interval,

$$\frac{s_{t+r+1}}{s_{t+r}} = \rho \frac{1 - \beta_1^{t+r}}{1 - \beta_1^{t+r+1}} < \rho < 1.$$

Moreover, $s_{t+r} \leq M\rho^r$. The choice of J_ϵ in Eq. 10 ensures $M\rho^{J_\epsilon} < \epsilon$, including the case in which the logarithmic ratio is an integer. The stopping test therefore succeeds by this step if no improvement occurs before then. An improvement interrupts this interval. □

Proof of lazy-update equivalence. For $u_t > 0$ at a state that has not stopped, define the next zero-gain stopping deadline by

$$r_* = \min \left\{ r \geq 1 : s_t \rho^r \frac{1 - \beta_1^t}{1 - \beta_1^{t+r}} < \epsilon \right\}. \tag{13}$$

Proposition 2 ensures $r_* \leq J_\epsilon$ and allows a binary search for the first integer that satisfies the condition. During a zero-gain interval, an implementation can advance a counter and stop exactly at r_*, without recomputing floating-point statistics at every evaluation. If a positive gain arrives first, it applies the decay for the preceding steps and processes that gain with the ordinary recurrence. This preserves the stopping step by induction over gain events. The zero-scale safeguard is handled separately. The equivalence is mathematical; floating-point implementations must preserve the strict inequality at the deadline.

Proof of Proposition 6. For $0 \leq g \leq \beta_2 u_t$, the next scale is $u_{t+1} = \beta_2 u_t$. Substituting $m_{t+1} = \beta_1 m_t + (1 - \beta_1) g$ and $m_t = s_t(1 - \beta_1^t) u_t$ into Eq. 6 gives Eq. 11. Its second term is the difference from the zero-gain score. The claim is conditional on the gain not exceeding the decayed scale; it does not assert that arbitrarily large gains always increase the normalized score. □

Proof of threshold monotonicity. Consider two runs with $\Lambda_1 < \Lambda_2$ and every other setting identical. Their thresholds satisfy $\epsilon_1 > \epsilon_2$. Until either stops, both runs have identical frontier and result heaps, visited sets, evaluation order, and controller states, by induction over distance evaluations. If the more permissive run stops because $s_t < \epsilon_2$, the other run must also have stopped by that step. Frontier exhaustion and the zero-scale safeguard do not depend on Λ. This proves Eq. 12. The top-k heap retains the best k discovered ranking values, so extending a prefix cannot increase any order statistic of those values. Translating this statement into Recall@k monotonicity additionally requires a ground truth tie convention consistent with the search; it does not supply a numerical recall bound. □

Why no recent improvement does not guarantee recall. Consider a controller whose observations contain only the evaluated vertices and their result feedback, with no bounds on unchecked points and no condition on paths through the graph. A connected graph can contain a long path of vertices that do not improve the initial answer, followed by k vertices closer to the query than every returned result. Two instances can agree on every coordinate and adjacency list inspected before stopping, while assigning those unseen terminal vertices either very large or very small query distances. They produce the same observed feedback and stopping decision, but their returned Recall@k can be one or zero. Such an example can be embedded in a Euclidean space by assigning the unseen vectors distinct positive distances arbitrarily near the query; the graph is unrestricted. Thus the bound on additional computations alone does not guarantee recall. A recall theorem would need additional graph or data assumptions that relate the checked points to those not yet checked, as geometric guarantees do in other settings.

A.3 Graphs Used to Compare Query Cost

Graph for Proposition 3. Proposition 3 uses the fixed-query monotonic-search condition defined in Section 4.2, without requiring the graph to be an MSNET for all vertex pairs.

The entry point s is adjacent to the exact k nearest neighbors $Y = \{y_1, \dots, y_k\}$, the answers are linked by edges $y_i y_{i+1}$, and each y_i has h points connected only to it, denoted by $z_{i,j}$. Choose distinct query distances satisfying

$$1 < d(y_1) < \cdots < d(y_k) < 2 < d(s) = 3 < d(z_{i,j}),$$

and scan s's neighbors in the order $y_1, \dots, y_k$. The search therefore finds all k nearest neighbors before checking the additional points.

The proof is in Appendix A.4.

The cost saving comes from avoiding unnecessary checks after finding the exact results. The closer-neighbor condition alone does not imply this saving. For example, $k = 10$, $h = 64$, the default betas, and $\epsilon = 0.01$ give $J_\epsilon = 50$: HNSW and Vamana use 651 distance computations at every width $B \geq k$, while the bound for FEEDSEARCH is 61. This is a constructed case with maximum degree 67, not a benchmark measurement.

Graph for Proposition 4. Consider the path

$$a \;-\; s \;-\; z \;-\; y \;-\; c_1 \;-\; \cdots \;-\; c_n,$$

with entry s, $k = 1$, and squared query distances

$$\begin{gathered} d(y) = 1, \quad d(a) = e^g, \quad d(s) = e^{2g}, \quad d(z) = e^{3g}, \\ d(c_i) = e^{g(n+1-i)/(n+1)}, \qquad 1 \leq i \leq n. \end{gathered} \tag{14}$$

Here $g > 0$, and s scans a before z. To reach the nearest neighbor y, the search must pass through z, which is farther from the query than s, while $c_1, \dots, c_n$ become progressively closer without improving the answer y. For every finite n, sufficiently small g realizes this path in $\mathbb{R}^2$, with edges connecting precisely the point pairs within a fixed radius. Its maximum degree is two. The coordinates and comparison proof are in Appendix A.4.

A.4 PROOFS OF THE TERMINATION AND QUERY COST RESULTS

Proof of Proposition 1. Before the first positive gain, the normalized stopping test is inactive. The first positive gain occurs by step $H_0 + 1$, so there are at most H_0 preceding zero-gain steps and the zero-scale safeguard cannot have fired. At any subsequent step $t \leq T$, let t_i be the most recent positive-gain step and $r = t - t_i \leq h$. From the recurrences,

$$m_t \geq (1 - \beta_1)\beta_1^r g_{\min}, \qquad u_t = \beta_2^r u_{t_i} \leq \beta_2^r g_{\max}.$$

The second inequality follows because every earlier gain is at most $g_{\max}$ and u is a decayed maximum. Consequently,

$$s_t \geq \frac{(1 - \beta_1)(g_{\min}/g_{\max})\rho^r}{1 - \beta_1^t} \geq (1 - \beta_1)(g_{\min}/g_{\max})\rho^h \geq \epsilon.$$

Suppose FEEDSEARCH stopped before completing step T. Before that step the two traversals, result heaps, and controller states coincide, but the bound at that step contradicts the strict stopping test $s_t < \epsilon$. Thus FEEDSEARCH completes the first T steps. If these steps find the true top-k, every subsequent potential gain is zero, and Proposition 2 gives the additional-work bound J_ϵ. An empty candidate queue can only end the search sooner. □

Proof of Proposition 3. Let s be the entry, $Y = \{y_1, \dots, y_k\}$ the exact answer, and connect each y_i to h additional points $z_{i,1}, \dots, z_{i,h}$. Each additional point has only y_i as a neighbor. Add edges from s to every y_i, and the chain edges $y_i y_{i+1}$ for $1 \leq i < k$. Choose distinct query distances satisfying

$$1 < d(y_1) < \cdots < d(y_k) < 2 < d(s) = 3 < d(z_{i,j}).$$

These values can be realized by Euclidean vectors, and all distances exceed δ. The closer-neighbor condition for the query holds: s reaches y_1, each y_i with $i > 1$ reaches y_{i-1}, and each $z_{i,j}$ reaches y_i. Order s's adjacency list as $y_1, \dots, y_k$.

After expanding s, both original algorithms retain all of Y for any width $B \geq k$. Since these are the globally closest k vertices, none can be evicted. Both algorithms must therefore expand every y_i

and compute distances to all kh additional points. Checking the neighbors of these additional points finds nothing new, giving exactly $1 + k + kh$ computations and the exact answer.

For FEEDSEARCH, the result heap first fills with $\{s, y_1, \ldots, y_{k-1}\}$, including $\{s\}$ when $k = 1$. Evaluating y_k gives the first positive gain $(\log d(s) - \log d(y_k))/k$, whose feedback score is $s_1 = 1$. Thus $\epsilon \leq 1$ permits discovery of the entire exact answer. Every subsequent distance computation concerns one of these farther points and gives no improvement. The zero-gain interval bound stops the search within J_ϵ more computations, proving both the cost bound and strict cost improvement. □

Coordinates for the graph in Eq. 14. Fix $n \geq 1$ and let $v_0, \ldots, v_{n+3}$ denote vertices in path order $a, s, z, y, c_1, \ldots, c_n$. Set

$$h_0 = \frac{\pi}{n+3}, \quad A = 2\sin(h_0/2), \quad B = 2\sin(h_0), \quad \eta = \frac{B - A}{8}, \quad g = \frac{2}{3}\log(1 + \eta) > 0.$$

Use query $q = (0, 0)$ and vectors

$$x_{v_i} = \sqrt{d(v_i)}\big(\cos(ih_0), \sin(ih_0)\big), \qquad 0 \leq i \leq n + 3,$$

with d from Eq. 14. These vectors have exactly the stated distinct squared query distances. All radii lie between 1 and $e^{3g/2} = 1+\eta$. Start instead with unit-radius points at these same angles. Adjacent unit-radius points are separated by A, and every nonadjacent pair is separated by at least B, since angular differences range from $2h_0$ to π. Moving each point radially changes its position by at most η, so each pair distance changes by at most 2η. For the fixed graph radius $r_0 = (A + B)/2$,

$$\|x_{v_i} - x_{v_{i+1}}\| \leq A + 2\eta < r_0, \qquad \|x_{v_i} - x_{v_j}\| \geq B - 2\eta > r_0 \quad (|i - j| \geq 2).$$

Thus connecting points within distance r_0 gives exactly the stated path, with at most two neighbors per point. As n grows, the radii become closer. The comparison uses this constructed graph rather than the output of a particular indexing algorithm.

Proof of Proposition 4. The count includes the initial evaluation of s. Every subsequent vertex distance is computed on first discovery and then cached.

FeedSearch. The result heap initially contains s. Expanding s evaluates a then z, producing gains g and 0. Expanding a adds no unseen vertex. FEEDSEARCH keeps z as a candidate and then checks its neighbors, finding y with gain g. Thus the first three feedback steps have gains $(g, 0, g)$, $g_{\min} = g_{\max} = g$, and $h = 1$. The assumed threshold satisfies Eq. 8, so FEEDSEARCH reaches y after four distance computations. Since $d(c_i) > d(y)$ for every i, every remaining gain is zero. Its total cost is at most $4 + J_\epsilon$, and it returns the exact nearest neighbor y.

Widths one and two. After expanding s, the discovered vertices have strict order a, s, z. Point z has rank three and is excluded from the retained pool for both widths. The remaining eligible vertex a reveals no unseen neighbor. Each original algorithm therefore returns a with recall zero after exactly three computations.

Every width at least three. Point z is initially retained. Both original algorithms expand a, then z, discover y, and expand y to discover c_1. Thereafter, c_{i+1} has smaller distance than all earlier c_j, while y remains the unique closest vertex. Thus each newly discovered chain vertex ranks second and belongs to every retained pool of size at least three. It is expanded next, exposing its successor. Termination consequently requires visiting all n chain vertices, for exactly $n + 4$ unique distance computations and recall one. Proposition 7 shows that this argument applies to both HNSW and Vamana when each point's distance is computed once.

Since $n > J_\epsilon$, the difference at recall one is at least $n - J_\epsilon > 0$. At budget $4 + J_\epsilon$, the only original searches that finish have width below three and recall zero, whereas FEEDSEARCH has recall one. Upper-layer routing, if shared before the fixed entry, adds the same cost to both sides and preserves these differences. □

Exact number of computations before termination. After the first three gains $(g, 0, g)$ and r computations without improvement,

$$s_{3+r} = \frac{(1 - \beta_1)(1 + \beta_1^2)\rho^r}{1 - \beta_1^{3+r}}.$$

Let T_ϵ be the smallest positive integer r at which this is below ϵ. Then $T_\epsilon \leq J_\epsilon$ and, provided $n > T_\epsilon$, FEEDSEARCH uses exactly $4 + T_\epsilon$ computations. This expression also shows that shrinking the positive parameter g to construct the graph does not change its stopping step. The separation concerns the number of distance computations; it is not a wall-clock or memory bound.

Recall at the same expected query cost. The cost comparison also holds if HNSW or Vamana chooses the width at random. Let p be the probability of choosing $B \geq 3$ on the graph in Eq. 14. Widths below three have cost 3 and recall zero; all larger widths have cost $n + 4$ and recall one. Consequently,

$$\mathbb{E}[C_{\text{native}}] = 3 + (n+1)p, \qquad \mathbb{E}[\text{Recall}_{\text{native}}@1] = p.$$

At the same expected cost as FEEDSEARCH's actual cost $C_F = 4 + T_\epsilon$, any such mixture therefore has

$$\mathbb{E}[\text{Recall}_{\text{native}}@1] = \frac{C_F - 3}{n+1} = \frac{T_\epsilon + 1}{n+1} < 1,$$

whereas FEEDSEARCH has recall one. Here each original search runs until its own termination condition is met, without a separate limit on distance computations.

B Additional Experimental Details

B.1 Six-Method Bucket Comparisons

Table 4 counts the highest measured QPS in all fully covered recall buckets between 0.90 and 0.99, using the selection rule in Figure 5. FeedSearch wins 22 of 27 buckets; these comparisons need not have identical achieved recall.

Table 4: Highest measured QPS among selected operating points in fully covered Recall@10 buckets (0.90–0.99). Counts depend on the bucket-matching and within-bucket selection protocol; each retained bucket covers all six methods.

Method	DEEP1M	SIFT10M	GIST1M	Tiny5M	Total
Native	0	0	0	0	0
Ada-ef	0	0	1	0	1
DARTH	0	0	0	0	0
LAET	0	0	0	0	0
DABS	1	0	2	1	4
FeedSearch	6	5	6	5	22
Fully covered buckets	7	5	9	6	27

B.2 Pairwise Comparison at Identical Recall

Table 5 compares each baseline with FeedSearch at identical measured Recall@10. Each pair uses its shared recall closest to 0.95 (ties favor lower recall) and each method's highest three-round median QPS at that recall. Recall is aggregate, not matched per query. FeedSearch has higher median QPS in 19 of 20 pairs; DABS is slightly faster on GIST1M. Tiny5M/DABS matches at 0.8846, outside the range of Figure 5.

Table 5: Pairwise QPS comparisons at identical achieved Recall@10. Both methods in a row attain the listed recall; recalls may differ across rows. Speedup is FeedSearch QPS divided by baseline QPS, with values above one favoring FeedSearch.

Dataset	Baseline	Shared recall	Baseline QPS	FeedSearch QPS	Speedup
DEEP1M	Native	0.9728	12,515.9	13,567.8	1.084×
DEEP1M	Ada-ef	0.9878	5,918.0	7,771.3	1.313×
DEEP1M	DARTH	0.9463	14,526.5	20,427.3	1.406×
DEEP1M	LAET	0.9709	11,421.4	14,113.0	1.236×
DEEP1M	DABS	0.9392	16,550.2	22,141.6	1.338×
SIFT10M	Native	0.9689	20,022.3	23,458.8	1.172×
SIFT10M	Ada-ef	0.9689	17,945.6	23,458.8	1.307×
SIFT10M	DARTH	0.9623	4,592.3	29,890.8	6.509×
SIFT10M	LAET	0.9614	8,760.7	31,151.7	3.556×
SIFT10M	DABS	0.9738	18,109.3	20,555.2	1.135×
GIST1M	Native	0.9537	2,159.8	2,195.0	1.016×
GIST1M	Ada-ef	0.9537	2,114.1	2,195.0	1.038×
GIST1M	DARTH	0.9598	1,369.7	1,889.8	1.380×
GIST1M	LAET	0.9607	1,459.2	1,860.5	1.275×
GIST1M	DABS	0.9548	2,139.6	2,132.7	0.997×
Tiny5M	Native	0.9547	1,712.8	1,925.0	1.124×
Tiny5M	Ada-ef	0.9477	1,960.5	2,103.6	1.073×
Tiny5M	DARTH	0.9489	1,302.7	2,097.8	1.610×
Tiny5M	LAET	0.9298	1,660.2	2,784.5	1.677×
Tiny5M	DABS	0.8846	4,211.4	4,271.6	1.014×

C NATIVE SEARCH PROCEDURES AND THE ROLE OF SEARCH WIDTH

This appendix relates HNSW's query-time efS (`efSearch`) to Vamana's query-time list size L. We distinguish retained search capacity, physical queue contents, and total search work. The procedures below follow HNSW's SEARCH-LAYER/K-NN-SEARCH (Malkov & Yashunin, 2020) and Vamana's GREEDYSEARCH (Subramanya et al., 2019). Our correspondence concerns sequential, unfiltered search on a fixed graph with a fixed distance function. Hierarchical routing, graph construction, disk I/O scheduling, and compressed-distance evaluation are separate considerations.

C.1 TWO IMPLEMENTATIONS OF WIDTH-CONTROLLED EXPLORATION

Write $d(v) = d(q, v)$ and let $\mathrm{Top}_b(A)$ contain the $\min\{b, |A|\}$ closest distinct vertices in A. Initially assume distinct query distances. Both widths satisfy $b \geq k$. A vertex is *discovered* when its query distance is first evaluated, and *expanded* when its adjacency list is examined. These are different events: HNSW's visited marks usually record discovery, whereas the visited set in Vamana's original pseudocode records expansion.

HNSW. Upper layers route the query to a base-layer entry using width one. The base layer maintains a min-priority candidate heap C and a max-priority retained set W, with $|W| \leq \mathrm{efS}$. A newly discovered vertex enters both only when W has space or the vertex improves its worst distance. Eviction from W need not remove the same vertex from C. Search stops when C is empty or its closest candidate is farther than the worst member of W.

Algorithm 2: HNSW query search: upper-layer routing and base-layer search

Input: Query q, result size k, width $B = \mathrm{efS} \geq k$

$s \leftarrow$ the index entry point;
for $\ell = \ell_{\max}, \ldots, 1$ **do**
  $s \leftarrow$ width-one greedy routing from s in layer ℓ;
Evaluate $d(s)$; $D \leftarrow \{s\}$; $C \leftarrow \{s\}$; $W \leftarrow \{s\}$;
while $C \neq \varnothing$ **do**
  $c \leftarrow \arg\min_{v \in C} d(v)$; $f \leftarrow \arg\max_{v \in W} d(v)$;
  if $d(c) > d(f)$ **then break**;
  $C \leftarrow C \setminus \{c\}$;
  foreach *neighbor* v *of* c *in layer zero* **do**
    if $v \notin D$ **then**
      $D \leftarrow D \cup \{v\}$; evaluate $d(v)$;
      if $|W| < B$ *or* $d(v) < \max_{u \in W} d(u)$ **then**
        $C \leftarrow C \cup \{v\}$; $W \leftarrow \mathrm{Top}_B(W \cup \{v\})$;
        `// Evictions affect` W`, not` C
return $\mathrm{Top}_k(W)$;

Vamana. The in-memory search starts from a designated vertex s. An ordered list S, capped at L, retains both expanded and unexpanded vertices. The next expansion is the closest member of $S \setminus X$, where X records expanded vertices. New neighbors join the list before retaining its closest L members. The stopping condition is $S \setminus X = \varnothing$. A discovered-ID cache can avoid repeated distance evaluations without changing this set-based procedure.

Algorithm 3: Vamana query search on a fixed graph

Input: Query q, graph G, entry s, result size k, width $L \geq k$

$S \leftarrow \{s\}$; $X \leftarrow \varnothing$;
while $S \setminus X \neq \varnothing$ **do**
  $c \leftarrow \arg\min_{v \in S \setminus X} d(v)$;
  $X \leftarrow X \cup \{c\}$;
  $S \leftarrow \mathrm{Top}_L(S \cup N_G(c))$;
  `// Expanded vertices can remain in` S
return $\mathrm{Top}_k(S)$;

C.2 A Common Retained-Set Invariant

For analysis, let D_t contain all vertices discovered through t completed expansions, and let X_t contain the expanded vertices. Define a retained pool and its eligible frontier by

$$P_t = \mathrm{Top}_B(D_t), \qquad F_t = P_t \setminus X_t. \tag{15}$$

The width-limited pool includes already expanded vertices. Thus B does not specify a queue of B *unexpanded* vertices.

Proposition 7 (Correspondence of efS and L)**.** *Consider the base-layer kernel of Algorithm 2 and Algorithm 3 on the same finite graph, query, entry vertex, and distinct query distances, with* $\mathrm{efS} = L = B$*. At every completed expansion, both retain P_t in Eq. 15: HNSW has $W_t = P_t$ and Vamana has $S_t = P_t$. Their eligible next vertex is* $\arg\min_{v \in F_t} d(v)$*, and both terminate exactly when $F_t = \varnothing$. Consequently they expand the same vertex sequence and return the same top-k result, although their physical candidate queues can differ.*

Proof. Retention. For any sets A, Z and fixed distance ordering,

$$\mathrm{Top}_B(\mathrm{Top}_B(A) \cup Z) = \mathrm{Top}_B(A \cup Z). \tag{16}$$

Every element discarded from A already has at least B closer elements; adding vertices cannot restore its rank to the first B. HNSW's per-neighbor admission test and eviction implement this operator incrementally. Vamana applies it to the union with an adjacency list. Both therefore maintain the closest B discovered vertices, irrespective of whether a temporarily competitive neighbor remains physically queued.

Eligibility and termination. For HNSW, every unexpanded member of W_t is in C_t: it was queued on admission and is removed from C_t only for expansion. No expanded vertex is requeued because discovery is marked. Hence

$$C_t \cap W_t = W_t \setminus X_t = F_t. \tag{17}$$

If $|D_t| < B$, nothing has been evicted, so $C_t = F_t$. Otherwise each stale candidate in $C_t \setminus W_t$ is strictly farther than every member of W_t. Such a vertex cannot become retained again as D_t grows. If F_t is nonempty, the closest member of C_t belongs to F_t and passes the HNSW boundary check. If F_t is empty, either C_t is empty or its closest member exceeds the boundary, causing HNSW to stop. This is precisely Vamana's test $S_t \setminus X_t = \varnothing$.

Coupling the searches. Initially both pools contain only s. If their states agree after t expansions, the eligibility argument gives the same next vertex. Both examine its neighbors and update their retained pools by Eq. 16. Induction gives identical expansion sequences and final retained pools, proving the result. □

What the common parameter controls. Both parameters instantiate the same rank cutoff B on discovered vertices and thus the same eligible frontier F_t. For any fixed discovered set D and $B_1 \leq B_2$,

$$\mathrm{Top}_{B_1}(D) \subseteq \mathrm{Top}_{B_2}(D). \tag{18}$$

A larger width admits additional, currently less competitive vertices whose neighbors may improve the answer. HNSW represents exhaustion of this retained frontier by a distance-boundary test; Vamana represents it by the absence of an unexpanded list entry. This is the precise sense in which efS and L play the same role. Equation 18 is conditional on a common D; complete searches with different widths can discover different sets. It does not provide a universal width-to-recall or width-to-runtime mapping.