Exploiting Structural Properties for Efficient Constraint-Aware HNSW Hyperparameter Tuning

GEON CHOI, Seoul National University, Republic of Korea
HOEUN LEE, Seoul National University, Republic of Korea
JAEYOUNG DO*, Seoul National University, Republic of Korea

Vector databases (VectorDBs) are a core component of modern retrieval systems, including Retrieval-Augmented Generation (RAG), where efficient Approximate Nearest Neighbor Search (ANNS) is critical. Among ANNS algorithms, Hierarchical Navigable Small World (HNSW) graphs are widely adopted for their strong recall-latency trade-off. However, configuring HNSW remains challenging: its hyperparameters jointly affect search quality, latency, build time, and index size in nonlinear ways, while production deployments impose strict resource and tuning-time constraints. We study HNSW hyperparameter tuning from a systems perspective and show that its configuration space exhibits strong structural regularities. Specifically, we identify monotonic, dominant unimodal, and separable relationships among search-time and construction-time parameters, which induce feasibility boundaries under performance and resource constraints. Building on this insight, we propose CHAT, a constraint-aware tuning framework for HNSW. Unlike generic black-box optimizers, CHAT exploits HNSW-specific structure to perform deterministic, sample-efficient search and prune resource-infeasible configurations before full index construction. Across multiple datasets and HNSW-based vector search engines, CHAT identifies configurations that maximize recall or throughput while satisfying constraints on accuracy, latency, build time, index size, and tuning budget. Compared to strong baselines, CHAT achieves up to 45% higher throughput or 11% higher recall, and converges up to 44× faster. These results show that principled, structure-aware tuning enables efficient and robust HNSW deployment beyond generic black-box optimization.

CCS Concepts: • **Information systems** → **Autonomous database administration**; **Database performance evaluation**; • **Theory of computation** → *Nearest neighbor algorithms.*

Additional Key Words and Phrases: Vector databases, Approximate nearest neighbor search, HNSW, Hyperparameter tuning, Constraint-aware optimization

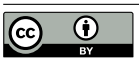
ACM Reference Format:
Geon Choi, Hoeun Lee, and Jaeyoung Do. 2026. Exploiting Structural Properties for Efficient Constraint-Aware HNSW Hyperparameter Tuning. *Proc. ACM Manag. Data* 4, 4 (SIGMOD), Article 273 (September 2026), 26 pages. https://doi.org/10.1145/3837111

1 Introduction

Recent advances in large language models (LLMs) have led to strong performance in text-based retrieval, generation, and summarization [32, 46]. However, their limited parameterization and training data can still lead to hallucinated or stale outputs [3, 15, 28, 43]. Retrieval-Augmented Generation (RAG) mitigates this limitation by retrieving external knowledge at inference time, avoiding costly fine-tuning while improving factual grounding [5, 10, 19, 22, 23, 35].

*Corresponding author.

Authors' Contact Information: Geon Choi, Seoul National University, Seoul, Republic of Korea, cds06126@snu.ac.kr; Hoeun Lee, Seoul National University, Seoul, Republic of Korea, hoeunlee@snu.ac.kr; Jaeyoung Do, Seoul National University, Seoul, Republic of Korea, jaeyoung.do@snu.ac.kr.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2026 Copyright held by the owner/author(s).
ACM 2836-6573/2026/9-ART273
https://doi.org/10.1145/3837111

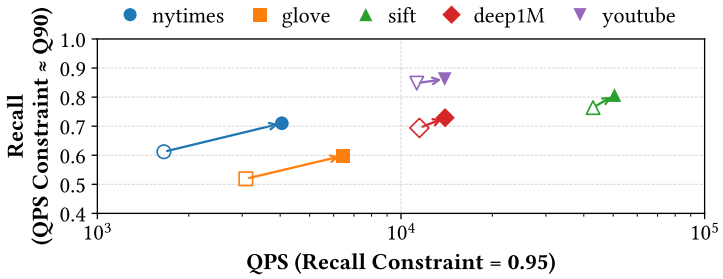


Fig. 1. Performance differences between default and optimal HNSW hyperparameter configurations across datasets using Faiss [20]. Filled markers indicate optimal configurations, while outlined markers indicate default configurations. See Section 6.1 for details.

At the core of RAG pipelines are vector databases (VectorDBs) [13, 39], which store large-scale embeddings and support Approximate Nearest Neighbor Search (ANNS) [12, 14, 21, 27, 30]. Among ANNS algorithms, Hierarchical Navigable Small World (HNSW) graphs [29] are widely adopted for their strong recall-latency trade-off and scalability [2].

Despite its popularity, deploying HNSW effectively remains difficult [29, 47]. HNSW performance is governed by three core hyperparameters: M, the maximum number of connections allowed per node; *efc*, the construction-time candidate-list size used during insertion; and *efs*, the query-time candidate-list size used during search. These parameters jointly affect recall, throughput, build time, and index size. As shown in Figure 1, fixed default HNSW configurations often do not coincide with the dataset-specific feasible frontier or constraint-aware optimum. Thus, even under identical recall or QPS targets, tuned configurations can avoid constraint violations or recover performance left on the table by defaults, achieving up to 144% higher throughput or 16% higher recall.

HNSW tuning is challenging because these parameters interact nonlinearly and often trade off competing objectives [45]. For example, increasing *efc* can improve graph quality and recall, but may also increase build cost, index size, and query traversal overhead. Existing tuning methods, including Grid/Random Search [4, 26], Optuna [1], NSGA-II [6], ECI [11], and VDTuner [44], largely treat HNSW as an opaque objective. More broadly, generic surrogate-based and Bayesian optimizers can be sample-inefficient in discrete, noisy, and narrow-feasibility search spaces [34, 36, 41], while Pareto-oriented methods [31] do not directly target the single constraint-satisfying optimum required here. These methods therefore spend substantial tuning budget exploring expensive configurations without directly exploiting the structural regularities induced by HNSW construction and search.

We propose CHAT, a constraint-aware, API-level black-box but HNSW-structure-aware tuning framework. CHAT does not inspect internal graph statistics; it uses only standard build/query APIs and measured validation signals. However, it exploits public HNSW mechanisms to decompose tuning into structured subproblems: guided search over construction parameters (M, *efc*), binary boundary search over *efs*, and resource-feasibility filtering before full index construction. Resource estimates are used only to avoid unnecessary builds; configurations that survive this filter are still built, measured, and validated before selection. This design leverages monotonic, dominant unimodal, and separable relationships in the HNSW configuration space to reduce unnecessary evaluations while selecting configurations validated against user-specified performance and resource constraints. We intentionally scope these structural claims to standard HNSW implementations;

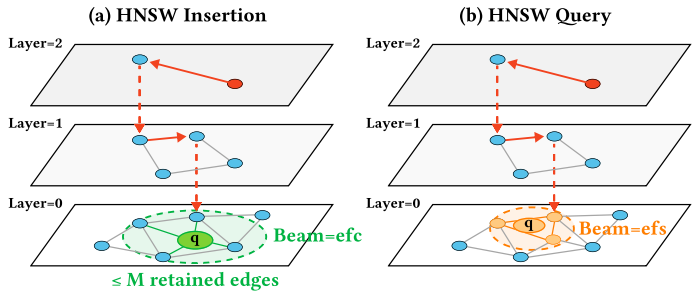


Fig. 2. HNSW insertion and query processing. (a) Insertion uses *efc* to search candidate neighbors and M to bound the close/diverse edges retained by SELECT-NEIGHBORS. (b) Query processing uses *efs* to control base-layer search after top-down greedy traversal.

extending the methodology to other ANN families requires identifying analogous search-time, construction-time, and resource regularities.

We evaluate CHAT on Faiss [20], Hnswlib [29], and Milvus [39] across five datasets. Compared with strong baselines, CHAT achieves up to 45% higher throughput or 11% higher recall, and converges up to 44× faster, while satisfying accuracy, throughput, build-time, index-size, and tuning-budget constraints.

Our key contributions are:

- We formulate constraint-aware HNSW hyperparameter tuning under performance, resource, and tuning-budget constraints.
- We characterize HNSW-specific structural regularities—monotone feasibility boundaries induced by *efs*, dominant unimodal construction-parameter trends, and separable resource dependencies.
- We introduce CHAT, an API-level black-box but HNSW-structure-aware tuner that uses hierarchical search and measured validation to identify constraint-satisfying configurations efficiently.
- We develop analytic resource models for build time and index size, enabling feasibility-aware pruning before full index construction.
- We validate CHAT across datasets and HNSW-based VectorDB backends, showing substantial performance and convergence gains over strong baselines.

2 Background

Hierarchical Navigable Small World (HNSW) [29] is a hierarchical graph-based index for Approximate Nearest Neighbor Search (ANNS). Each data point is represented as a node, and proximity relationships are represented as graph edges. HNSW exposes three core tuning parameters used throughout this paper: M, the degree budget that controls how many neighbors a node can retain; *efc*, the construction-time candidate-list size used when inserting nodes; and *efs*, the search-time candidate-list size used when answering queries. The construction parameters M and *efc* determine the graph structure, whereas *efs* controls query-time traversal over a fixed graph.

Figure 2 illustrates the layered structure and the two main HNSW procedures. Each node is assigned a maximum layer by randomized level generation and appears in every layer from Layer 0 up to that level. Since upper layers contain progressively fewer nodes, they provide sparse long-range routing links, while Layer 0 contains all data points and supports fine-grained search. Both insertion and query processing therefore follow a coarse-to-fine pattern: HNSW first performs greedy top-down traversal through upper layers and then conducts a broader search near the Layer 0.

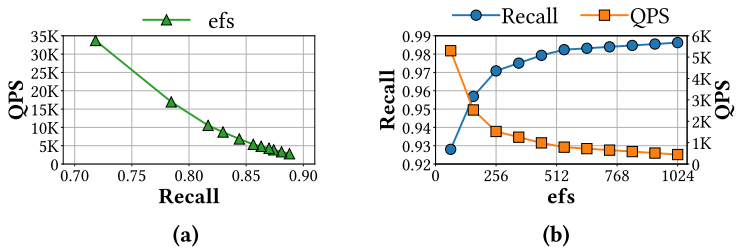


Fig. 3. Effect of *efs* on search performance. (a) Recall-QPS trade-off as *efs* increases. (b) Individual performance curves showing recall saturation and QPS decline with increasing *efs*.

During insertion (Figure 2(a)), a new point q first uses greedy descent with Beam = 1 to locate an entry point near q. From the assigned layer of q down to Layer 0, HNSW runs Search-Layer with candidate-list size *efc* to collect candidate neighbors. These candidates are then passed to Select-Neighbors, which retains close and diverse neighbors up to the layer-specific degree budget induced by M; fewer than M edges may be retained when candidates are redundant. During query processing (Figure 2(b)), the query vector q is not inserted into the graph. After the same greedy top-down traversal, HNSW runs Search-Layer at Layer 0 with candidate-list size *efs* and returns the top-k candidates from the final working set. Thus, increasing *efs* broadens query-time search on a fixed graph, while changing M or *efc* changes the graph and requires rebuilding the index.

3 Decoding HNSW Hyperparameter Trade-offs

This section presents an in-depth analysis of the three key HNSW hyperparameters (i.e., M, *efc*, and *efs*) and their non-trivial interactions in shaping search performance [29]. Because these parameters are tightly coupled, changing one often requires compensatory adjustments in others; understanding these trade-offs is essential for principled tuning and motivates our CHAT framework. We focus here on Recall and query throughput (QPS), and show that the strict monotonicity of *efs* with respect to recall and latency induces a total order over configurations, enabling binary feasibility search for constraint satisfaction rather than heuristic exploration. Resource-usage metrics (index build time and size) are analyzed in Section 4 to inform our constraint surrogate models. Unless otherwise specified, figures referenced in this section are generated using Faiss [20] on the nytimes dataset [2] (see Section 6.1 for details on the experimental setup). The qualitative patterns described below were consistent across query parallelism levels (1, 4, 16, and 64 threads); the figures in this section use 64 threads, and we focus on these structural trends rather than absolute QPS values.

3.1 Effect of *efs* on Search Accuracy and Speed

The hyperparameter *efs* controls the size of the candidate set maintained and expanded during query-time traversal at the base layer. Increasing *efs* broadens the search scope, thereby improving recall, but also incurs higher computational costs due to additional distance evaluations, resulting in lower query throughput (QPS). This monotonic accuracy–speed trade-off is illustrated in Figure 3(a).

Beyond this monotonic trade-off, *efs* exhibits diminishing returns in the high-*efs* regime. As shown in Figure 3(b), recall improvements rapidly saturate once most informative candidates near the query have already been explored, making additional candidates largely distant or redundant. QPS degradation also flattens in this regime because the number of candidates that can be meaningfully expanded is constrained by the graph structure and by the finite set of nearby nodes that can

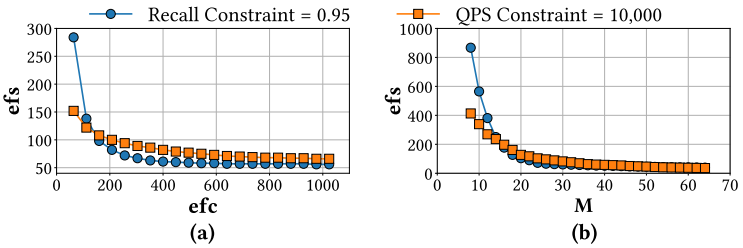


Fig. 4. Hyperparameter interactions under performance constraints. (a) Increasing *efc* shifts the Recall-feasibility boundary toward smaller *efs*. (b) Increasing M shifts the QPS-feasibility boundary toward smaller *efs*. Both exhibit steep initial movement followed by saturation.

contribute to the top-k results. Thus, while the nominal candidate budget increases, the marginal performance change from additional search effort decreases.

These saturation behaviors stem from the greedy search mechanism employed at layer 0 in HNSW. As the search converges toward a local optimum, discovering substantially better candidates becomes increasingly difficult, causing the marginal utility of additional search effort—measured as performance gain per unit cost—to drop sharply. Crucially, this structure induces a monotone feasibility ordering over *efs*: larger *efs* improves recall but increases query cost. This property enables binary feasibility search for constraint satisfaction, rather than costly heuristic exploration, and explains why blindly increasing *efs* is inefficient for balancing accuracy and latency.

3.2 Interaction between Construction Parameters and *efs*

The construction-time parameters *efc* and M shape the HNSW graph through complementary mechanisms. The parameter *efc* controls the breadth of candidate search during insertion: a larger *efc* exposes Select-Neighbors to a richer candidate pool, increasing the likelihood that high-quality, diverse edges survive pruning. Because Select-Neighbors often retains fewer than M edges due to its diversification criterion, larger *efc* can improve realized connectivity even when the degree cap is not the binding constraint. M, in contrast, directly caps per-node degree. Thus, *efc* shapes *which* candidate edges are available, while M bounds *how many* are retained.

To characterize the coupling with query-time search, we use the *constraint-boundary efs*: the minimum *efs* that reaches the target Recall under a Recall constraint, or the maximum *efs* that keeps throughput above the target under a QPS constraint. Figure 4(a) shows that increasing *efc* shifts this boundary downward under a Recall constraint, because a richer candidate pool improves graph quality and fewer query-time candidates suffice. Figure 4(b) shows the similar shift for M under a QPS constraint: larger degree caps increase work per expanded node, reducing the maximum feasible *efs*. The boundary moves in the same direction under the opposite constraint for both parameters, because stronger realized connectivity simultaneously improves Recall (lowering minimum-feasible *efs*) and raises traversal cost (lowering maximum-feasible *efs*).

The steep-to-gradual shape in both panels reflects a common saturation mechanism in construction-time graph improvement. For *efc*, the benefit saturates because Select-Neighbors increasingly rejects redundant candidates once the candidate pool is sufficiently rich. For M, realized connectivity can grow sublinearly and saturate below the nominal cap because the same diversification rule limits which additional neighbors are retained. Thus, *efc* and M can both reduce the boundary *efs*, but their marginal benefit diminishes once the graph becomes sufficiently navigable.

These observations have two implications for tuning. First, stronger realized connectivity typically shifts the constraint-boundary *efs* downward, allowing CHAT to narrow the *efs* search range

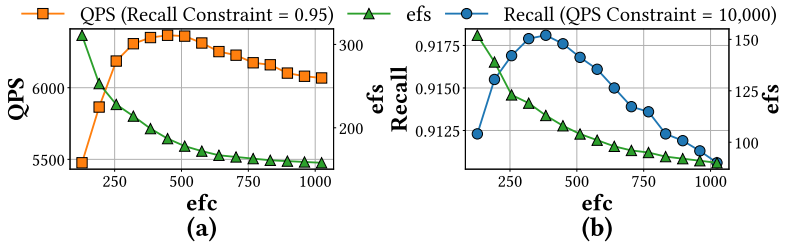


Fig. 5. Dominant unimodal performance patterns with varying *efc* under fixed M. (a) Under a Recall constraint, QPS first improves as increasing *efc* reduces the constraint-boundary *efs*, then declines as traversal overhead dominates. (b) Recall under a QPS constraint follows the same pattern.

using previous evaluations (Section 5.3). Second, when graph-quality gains saturate while traversal overhead continues to grow, the constrained objective moves from a quality-improving regime to an overhead-dominated regime. This crossover underlies the dominant unimodal trends analyzed in Sections 3.3 and 3.4.

3.3 Unimodality over *efc* under Fixed M

We fix M and study how *efc* affects the constrained objective after *efs* has been chosen at the feasibility boundary. For each *efc*, CHAT selects the boundary *efs* under the active performance constraint: the smallest *efs* satisfying the Recall target, or the largest *efs* satisfying the QPS target. Let $f_M(efc)$ denote the resulting objective value, namely QPS under a Recall constraint and Recall under a QPS constraint. We call f_M unimodal if it is non-decreasing up to some peak index and non-increasing thereafter.

Figure 5 shows this dominant unimodal behavior under both constraints. When *efc* is small, the graph is poorly connected, so the boundary *efs* is unfavorable: Recall constraints require large *efs*, while QPS constraints force small *efs* and sacrifice Recall. Increasing *efc* enlarges the construction-time candidate pool and improves graph navigability, allowing the constraint to be met more efficiently and raising the objective. Beyond the peak, additional candidates become increasingly redundant under Select-Neighbors and the fixed degree cap M, while traversal overhead continues to grow. The objective therefore shifts from a quality-improving regime to an overhead-dominated regime.

Lemma 3.1 (Sufficient condition for unimodality over *efc*). *Fix M and an active constraint $C \in \{\text{Recall}, \text{QPS}\}$. On the ordered grid $efc_1 < \cdots < efc_n$, write the marginal change as $\Delta f_M(efc_i) = \Delta B_M(efc_i) - \Delta T_M(efc_i)$, where ΔB_M is the marginal graph-quality benefit and ΔT_M is the residual traversal/search cost. Any reduction in expanded nodes from better graph navigability is counted in ΔB_M, not as negative cost. If*

(i) $\Delta T_M(efc_i) \geq 0$ for all i, and

(ii) once $\Delta B_M(efc_i) \leq \Delta T_M(efc_i)$, larger efc values do not reveal a delayed high-utility connectivity regime that restores $\Delta B_M > \Delta T_M$,

then $f_M(efc)$ is unimodal.

Proof sketch. We consider only ordered grid points for which a feasible boundary *efs* exists under the active performance constraint; resource feasibility is handled separately in Section 5.4. Let $d_i = f_M(efc_{i+1}) - f_M(efc_i)$. By the marginal decomposition in the lemma statement, $d_i = \Delta B_M(efc_i) - \Delta T_M(efc_i)$; hence $d_i \leq 0$ exactly when $\Delta B_M(efc_i) \leq \Delta T_M(efc_i)$, and condition (i) gives ΔT_M its intended interpretation as a non-negative residual traversal/search cost.

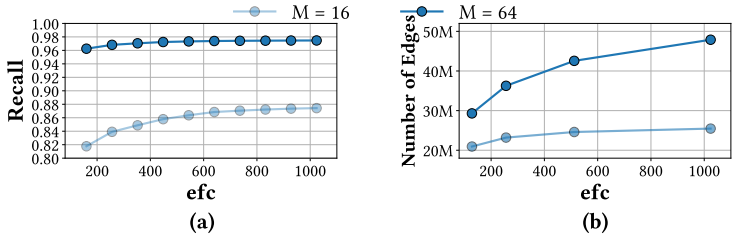


Fig. 6. Graph quality and realized edge growth across different M values. (a) Recall as a function of *efc*, showing that larger M values reach high graph quality with smaller *efc*. (b) Number of edges as a function of *efc*, showing saturation in realized connectivity under fixed degree caps.

If every d_i is positive, the sequence is increasing and efc_n is a peak. Otherwise, let $k = \min\{i : d_i \leq 0\}$; by minimality, $d_i > 0$ for every $i < k$. Condition (ii) states that, after this crossing, larger *efc* values cannot restore $\Delta B_M > \Delta T_M$; therefore $d_j \leq 0$ for every $j \geq k$. Consequently, $f_M(efc)$ is non-decreasing up to the peak efc_k and non-increasing thereafter, allowing ties, and is unimodal on the ordered grid. □

Lemma 3.1 is sufficient, not universal. It captures the standard HNSW regime targeted by CHAT: bounded degree, diversification-based pruning, and no delayed high-utility connectivity regime. When condition (ii) fails, the constrained objective may become bimodal or multi-modal; Section 6.4 presents such a counterexample. When the conditions hold, f_M can be searched efficiently using the ternary-style procedure in Section 5.

3.4 Outer-Objective Unimodality over M

We next analyze how the best achievable objective changes as M varies. For each M, CHAT chooses the best construction effort, denoted $efc^*_M \in \arg\max_{efc} F_C(M, efc)$, and the boundary *efs* satisfying the active performance constraint. Let $g(M) = \max_{efc} F_C(M, efc)$ denote the resulting outer objective: QPS under a Recall constraint and Recall under a QPS constraint. We call $g(M)$ unimodal if it is non-decreasing up to some peak index and non-increasing thereafter.

Figure 6 explains why efc^*_M generally decreases as M increases. When M is small, each node can retain only a limited number of neighbors, so larger *efc* is useful for exposing high-quality and diverse candidate edges. When M is larger, the graph can already retain more useful routing alternatives, so similar navigability can be achieved with a smaller construction-time candidate pool. Figure 6(b) confirms this substitution effect: realized edges saturate quickly under tight degree caps, while larger M allows the graph to keep growing as larger *efc* exposes more useful candidates. Thus, M and *efc* act as partial substitutes for graph quality, explaining the dominant decreasing trend of efc^*_M.

Figure 7 shows the resulting outer objective. Under a Recall constraint, small M produces poor connectivity, so the Recall target requires a large boundary *efs* and suppresses QPS. Increasing M initially improves graph navigability and reduces the required *efs*, so QPS improves. At larger M, however, additional degree capacity provides diminishing graph-quality benefit while increasing neighbor processing per expanded node, shifting the objective to an overhead-dominated regime. Under a QPS constraint, the same trade-off appears in the opposite objective: increasing M initially improves Recall, but excessive degree forces a smaller boundary *efs* to preserve throughput, eventually reducing Recall. Both cases show a dominant unimodal trend in $g(M)$.

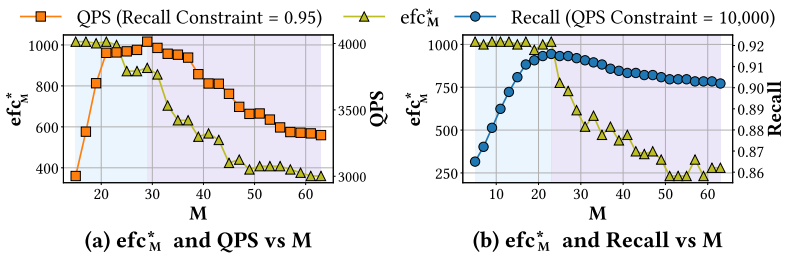


Fig. 7. Optimized *efc* and objective performance across varying M values. (a) efc^*_M and resulting QPS versus M under a Recall constraint. (b) efc^*_M and resulting Recall versus M under a QPS constraint.

Lemma 3.2 (Sufficient condition for outer-objective unimodality over M). *Fix an active constraint $C \in \{\text{Recall}, \text{QPS}\}$. On the ordered grid $M_1 < \cdots < M_n$, write the marginal change as $\Delta g(M_i) = \Delta B_{\text{out}}(M_i) - \Delta T_{\text{out}}(M_i)$, where ΔB_{out} is the marginal graph-quality benefit from increasing M with efc re-optimized at each M, and ΔT_{out} is the residual traversal/search cost. Any reduction in expanded nodes from better graph navigability is counted in ΔB_{out}, not as negative cost. If*

(i) $\Delta T_{\text{out}}(M_i) \geq 0$ for all i,

(ii) once $\Delta B_{\text{out}}(M_i) \leq \Delta T_{\text{out}}(M_i)$, larger M values do not reveal a delayed high-utility connectivity regime that restores $\Delta B_{\text{out}} > \Delta T_{\text{out}}$, and

(iii) re-optimizing efc at larger M does not create a new high-performing branch, i.e., M and efc remain partial substitutes for graph quality,

then $g(M)$ is unimodal over the ordered M grid.

Proof sketch. We consider only grid points for which the outer objective $g(M_i)$ is defined, and let $d_i = g(M_{i+1}) - g(M_i)$. By the marginal decomposition in the lemma statement, $d_i = \Delta B_{\text{out}}(M_i) - \Delta T_{\text{out}}(M_i)$, and condition (i) gives ΔT_{out} its intended interpretation as a non-negative residual traversal/search cost.

To distinguish the two possible sources of an outer-envelope recovery, choose $efc^*_i \in \arg\max_{efc} F_C(M_i, efc)$, so that $g(M_i) = F_C(M_i, efc^*_i)$ with F_C evaluated at the corresponding boundary *efs*, and write $a_i = F_C(M_{i+1}, efc^*_i) - F_C(M_i, efc^*_i)$ and $r_i = F_C(M_{i+1}, efc^*_{i+1}) - F_C(M_{i+1}, efc^*_i)$. Then $d_i = a_i + r_i$, and $r_i \geq 0$ by the optimality of efc^*_{i+1}; thus, even if the change along the previously optimal branch becomes non-positive, re-optimizing *efc* could in principle create a positive outer marginal. When efc^*_i is infeasible at M_{i+1}, this additive decomposition does not apply; any recovery at that step then arises entirely from a different *efc* branch and is therefore itself a cross-branch recovery. This decomposition only identifies the two recovery mechanisms; the outer marginal remains the benefit–cost marginal defined in the lemma statement.

If every d_i is positive, $g(M)$ is increasing and M_n is a peak. Otherwise, let $k = \min\{i : d_i \leq 0\}$, so $d_i > 0$ for every $i < k$. After this first crossing, condition (ii) rules out a delayed recovery from further increases in M once the graph-quality benefit no longer offsets the residual traversal cost, and condition (iii) rules out the remaining cross-branch recovery by excluding a newly dominant optimizing *efc* branch at larger M. The observed partial-substitution trend between M and *efc* motivates condition (iii), but its formal role is the exclusion of such branch switching. Together with condition (i)'s residual-cost interpretation, conditions (ii) and (iii) imply that once an outer marginal becomes non-positive, all later outer marginals remain non-positive: $d_j \leq 0$ for every $j \geq k$. Consequently, $g(M)$ is non-decreasing up to the peak M_k and non-increasing thereafter, allowing ties, and is unimodal over the ordered M grid. □

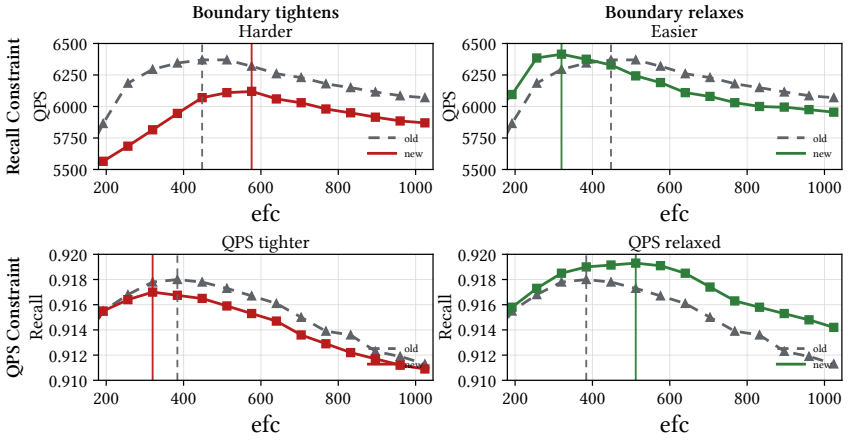


Fig. 8. Drift implications of *efs*-boundary shifts. Under a Recall constraint, efc^*_M moves right when the required *efs* increases and left when it decreases. Under a QPS constraint, efc^*_M moves left when the maximum allowed *efs* decreases and right when it increases.

Lemma 3.2 is sufficient, not universal. In particular, Lemma 3.1 alone does not imply unimodality of the outer envelope $g(M)$; condition (iii) rules out recoveries caused by re-optimizing *efc* into a delayed high-utility regime at larger M. If these conditions fail, $g(M)$ may become bimodal or multi-modal, as discussed in Section 6.4. When they hold, the outer search over M can be performed efficiently using the ternary-style procedure in Section 5.

3.5 Structural Implications

Taken together, these observations reveal that the HNSW hyperparameter space is not an unstructured three-dimensional landscape. Instead, it forms a partially ordered search space with distinct structural properties:

- ***efs*** induces a monotonic trade-off between recall and latency, enabling binary feasibility search.
- **M** exhibits unimodal objective behavior under fixed constraints, enabling efficient ternary optimization.
- ***efc*** acts as a feasibility gate that constrains construction-time resources.

These properties fundamentally distinguish HNSW tuning from generic black-box optimization problems. They enable deterministic, low-sample optimization strategies that exploit algorithmic structure rather than relying on random exploration. While the precise curvature of these trade-offs varies across datasets and backends, the monotonic and unimodal structures consistently appear in standard HNSW implementations with bounded degree and greedy layer-wise search. These structural properties also provide a lens for reasoning about how the optimal construction effort responds to corpus and workload updates, as discussed next.

3.6 Drift Implications of Feasibility Boundaries

The preceding analysis assumes a fixed corpus and query set. We now consider how the same trade-offs change when the corpus or query distribution changes. Under a Recall constraint, efc^*_M marks the transition between a quality-limited region—where increasing *efc* still substantially reduces the required *efs*—and a cost-limited region, where graph-quality gains have saturated. A change that *tightens* the Recall boundary $efs^*_{\text{Recall}}(M, efc)$ (the minimum *efs* satisfying the Recall target increases) expands the quality-limited region and moves efc^*_M toward larger *efc*; *relaxing* the

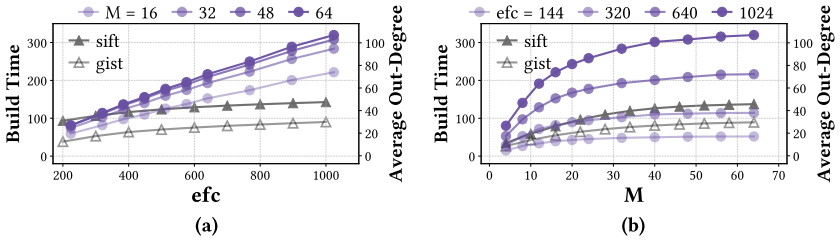


Fig. 9. Build time and realized connectivity of HNSW as functions of the construction hyperparameters M and *efc*. Index construction time (left y-axis) and average out-degree (AOD; right y-axis) are shown. (a) For fixed M, both build time and AOD increase monotonically with *efc*. (b) For fixed *efc*, build time grows sublinearly in M, while AOD saturates as M increases.

boundary shifts efc^*_M in the opposite direction. Under a QPS constraint, tightening the boundary $efs^*_{\mathrm{QPS}}(M, efc)$ (the maximum feasible *efs* decreases) makes high-*efc* configurations cost-dominated earlier, so efc^*_M moves leftward; relaxing it shifts efc^*_M rightward.

Figure 8 summarizes these directional effects. Insertions, deletions, and query-distribution shifts can each tighten or relax the active boundary depending on their net effect; the change type alone therefore does not determine the direction of movement. These directions should be understood as typical structural behaviors rather than universal guarantees, especially under near-duplicate insertions, targeted deletions, or strong query distribution shifts.

4 Relationship between Hyperparameters and HNSW Resources

This section characterizes how the HNSW construction hyperparameters (M, *efc*) affect build time and index size. We separate the algorithmic dependence induced by Insert, Search-Layer, and Select-Neighbors from backend-specific constants, treating N and d as fixed workload parameters. The goal is not to introduce a new objective, but to identify structural resource dependencies that enable feasibility-aware pruning. Unless otherwise specified, measurements use the same experimental setup as Section 3.

4.1 Index Construction Time

HNSW constructs the index by inserting points sequentially. For a dataset $D = \{x_i\}_{i=1}^N$, the total build time decomposes over insertions:

$$T_{\mathrm{build}}(N, M, efc, d) = \sum_{i=1}^{N} T_{\mathrm{ins}}(i; M, efc, d). \tag{1}$$

Each insertion descends from the top layer to layer 0, performs Search-Layer to identify candidate neighbors at each visited layer, and then applies Select-Neighbors to finalize outgoing edges. Let L_i denote the maximum layer of the i-th inserted point. The insertion cost can be decomposed as

$$T_{\mathrm{ins}}(i; M, efc, d) = \sum_{\ell=0}^{L_i} T_{\mathrm{search}}(i, \ell; M, efc, d) + \sum_{\ell=0}^{L_i} T_{\mathrm{select}}(i, \ell; M, d). \tag{2}$$

Search-cost accounting. The dominant cost in Search-Layer comes from graph exploration: the algorithm expands candidate nodes, inspects their adjacency lists, and evaluates previously unseen candidates. Let $X_{i,\ell}(M, efc)$ denote the number of expanded nodes when inserting point i at layer ℓ, and let $\bar{\delta}_{i,\ell}(M, efc)$ denote the average realized out-degree inspected per expansion. The

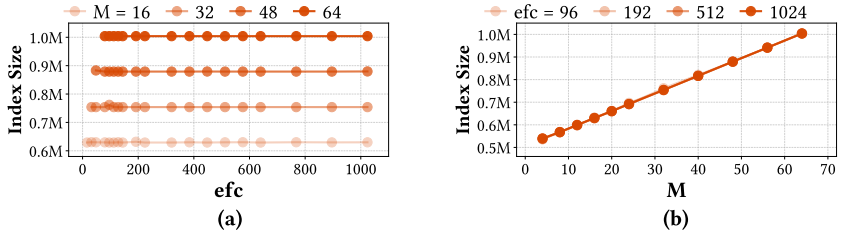


Fig. 10. Index size dependence on the construction hyperparameters M and *efc*. (a) For a fixed M, index size is largely insensitive to *efc*. (b) For a fixed *efc*, index size increases approximately linearly with M, and curves for different *efc* largely overlap.

dominant search cost can then be summarized as

$$T_{\text{search}}(i, \ell; M, efc, d) \approx c_d \, X_{i,\ell}(M, efc) \bar{\delta}_{i,\ell}(M, efc) + C_{\text{impl}}, \tag{3}$$

where c_d captures the cost of candidate inspection and distance evaluation for a fixed d-dimensional workload, and C_{impl} absorbs backend-specific bookkeeping overheads. This accounting abstracts away implementation constants. In particular, implementations that maintain visited sets may avoid repeated distance computations for the same node, so adjacency inspection and distance evaluation need not be in one-to-one correspondence. The structural dependence is nevertheless unchanged: *efc* primarily controls the expansion term, whereas M controls the degree cap that bounds the realized-degree term.

Role of *efc*. The construction parameter *efc* bounds the candidate set maintained by SEARCH-LAYER and therefore controls the breadth of graph exploration during insertion. Increasing *efc* allows the search to retain more candidates and expand more nodes before termination. In Equation (3), this primarily increases the expansion term $X_{i,\ell}(M, efc)$, explaining the build-time increase observed in Figure 9(a). A broader construction search can also expose SELECT-NEIGHBORS to more candidate neighbors, which explains the accompanying increase in realized connectivity.

Saturation effects with increasing M. The parameter M sets the per-layer degree cap up to a constant-factor difference at layer 0. To quantify realized connectivity, we define the average out-degree across the HNSW layer hierarchy as

$$\text{AOD}(M, efc) = \frac{\sum_\ell \sum_{v \in V^{(\ell)}} \deg^{(\ell)}(v)}{\sum_\ell |V^{(\ell)}|}. \tag{4}$$

AOD is the realized counterpart of the degree cap: it measures how many neighbors are retained after diversification and re-pruning. Although the degree cap grows with M, realized AOD need not grow linearly. HNSW's neighbor selection procedure favors diverse neighbors and may reject redundant candidates even when the cap increases; when existing neighborhoods exceed their degree limits, the same pruning logic is applied again. As a result, increasing M yields diminishing marginal increases in realized connectivity.

In Equation (3), this means that $\bar{\delta}_{i,\ell}(M, efc)$ grows sublinearly with the cap, which is consistent with the build-time saturation observed in Figure 9(b).

Overall, HNSW construction induces a separable resource dependence: *efc* primarily governs the breadth of construction search, while M controls per-expansion connectivity through the realized graph degree. Section 5.4 instantiates this structure as a compact build-time feasibility estimator.

4.2 Index Size

For fixed N and d, vector storage and metadata are constant over (M, efc); the hyperparameter-dependent term is adjacency storage. Since HNSW stores $O(M)$ outgoing edges for an expected $O(N)$ node copies across layers, the dominant index-size dependence is

$$S_{\text{index}}(N, M) = C_{\text{fixed}}(N, d) + \Theta(NM), \tag{5}$$

where $C_{\text{fixed}}(N, d)$ includes vector storage and fixed metadata. Backend-specific choices such as memory layout, alignment, and auxiliary bookkeeping change the constants, but not the qualitative dependence on (M, efc).

The construction parameter *efc* does not appear in Equation (5) because it changes the search budget used during construction, not the degree cap that determines adjacency capacity. A larger *efc* may affect which neighbors are selected, and can therefore slightly change realized connectivity, but it does not change the dominant storage budget induced by M. Thus, unlike build time, index size is governed primarily by the degree-cap-induced adjacency storage rather than by the amount of construction search. This explains why index size remains largely invariant to *efc* in Figure 10(a), while increasing approximately linearly with M in Figure 10(b).

4.3 Feasibility-Aware Tuning Implications

The analysis above yields three structural properties of HNSW resource usage:

- construction time increases with the search budget *efc*.
- construction time grows sublinearly with M because realized connectivity saturates under neighbor selection and pruning.
- index size scales approximately linearly with M and is largely independent of *efc*.

These separable dependencies allow infeasible regions of the hyperparameter space to be identified without constructing every candidate index. Build-time constraints can be evaluated using the dependence of search breadth on *efc* and realized connectivity on M, while index-size constraints can be evaluated primarily from M. Section 5.4 uses these structural properties to construct compact feasibility estimators, and Section 5 applies them to prune resource-violating configurations before expensive index construction.

5 Efficient Constraint-Aware Hyperparameter Optimization

This section presents CHAT, a constraint-aware framework for tuning HNSW hyperparameters. Given user constraints on search quality, search efficiency, resource usage, and tuning time, CHAT seeks the configuration (M, efc, efs) that maximizes the target objective while satisfying these constraints.

CHAT combines three ideas: it decomposes HNSW tuning into structured subproblems, exploits the monotonicity and dominant unimodality analyzed in Section 3, and filters resource-infeasible configurations using the models in Section 4.

CHAT is API-level black-box but HNSW-structure-aware. It uses only standard build/query APIs and validation-workload measurements, including Recall, QPS, build time, and index size, without inspecting adjacency lists, layer populations, or degree distributions. However, it is not ANN-agnostic: its search decomposition relies on public HNSW mechanisms, namely bounded-degree construction controlled by M, greedy layer-wise Search-Layer, and diversification-based Select-Neighbors. We therefore scope its structural claims to standard CPU HNSW implementations that preserve these semantics, including Faiss [20], Hnswlib [29], and Milvus [39]. These assumptions may weaken for variants with different neighbor selection, disk- or GPU-dominated bottlenecks, or dynamic graph maintenance. They also do not automatically transfer to non-HNSW families such as IVF+PQ [21], ScaNN [14], or NSG [9].

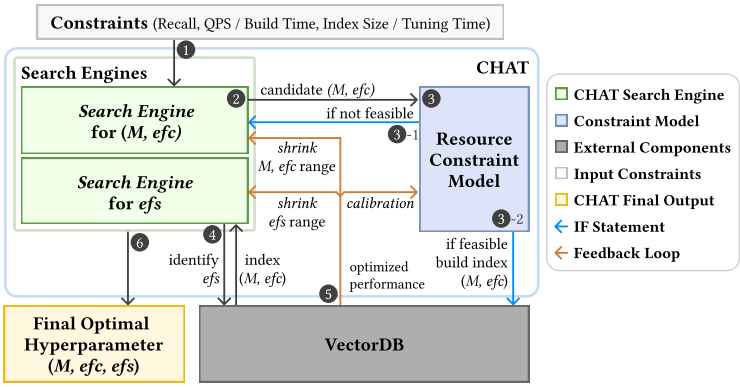


Fig. 11. The overall architecture and iterative tuning workflow of CHAT.

5.1 System Overview

Figure 11 summarizes the end-to-end workflow of CHAT. Given a dataset, an HNSW backend, performance and resource constraints, and a tuning budget (❶), the *Search Engine for (M, efc)* proposes a candidate construction configuration (❷). The *Resource Constraint Models* screen the candidate (❸): predicted budget violations are pruned without index construction (❸-1), while the remaining candidates are built and measured on the target backend (❸-2). For each measured resource-feasible index, the *Search Engine for efs* identifies the boundary *efs* that maximizes the objective under the active performance constraint (❹). The resulting performance and resource measurements calibrate the resource models and update the search history, tightening subsequent search ranges (❺). At convergence or budget exhaustion, CHAT returns the best validated feasible configuration (M, efc, efs) found within the budget (❻). When the corpus or query workload changes, the same history warm-starts retuning, as detailed in Section 5.5.

5.2 Search for Optimal Construction Configuration

CHAT searches the construction parameters using a nested but simple strategy. For each probed M, it searches for a promising *efc*, skips candidates that are model-infeasible under the resource budgets, builds only the remaining candidates, and then selects *efs* by boundary search on the built index. Thus, the original three-dimensional tuning problem is reduced to three one-dimensional decisions: localizing a good M region, selecting the best *efc* for that M, and choosing the boundary *efs* that satisfies the active performance constraint. We write $g(M)$ for the best validated objective value returned after optimizing *efc* and *efs* under a fixed M; formally, $g(M) = \max_{efc} perf(M, efc, efs^*(M, efc))$ over configurations feasible under C_{perf} and C_{res}.

Algorithm 1 implements the outer search over M. Each probed M is evaluated by Optimize-EFC, which returns the best measured feasible triplet found for that M. The outer loop then shrinks the M interval toward the probe with higher validated performance, following the dominant unimodal trend of $g(M)$ analyzed in Section 3.4. The best measured feasible configuration is tracked throughout the search, so CHAT can return a validated configuration even if the tuning budget expires before the intervals fully converge.

Algorithm 2 performs the inner search over *efc* for a fixed M. Before ternary search, CHAT uses prior optima in $\mathcal{H}$ to narrow the candidate range: larger M values typically need smaller construction effort because additional degree capacity already improves graph connectivity. This

Algorithm 1 Search Engine for (M, efc)

Input: Dataset $\mathcal{D}$; HNSW backend $\mathcal{V}$; performance constraint C_{perf}; resource constraints C_{res}; tuning budget $\mathcal{T}$
Output: Best validated configuration $bestCfg = (M, efc, efs)$

Initialize resource models $\hat{h}_{bt}, \hat{h}_{is}$ and history $\mathcal{H} \leftarrow \emptyset$
Initialize domains $\mathcal{M} \leftarrow [4, 64]$, $\mathcal{E}_c \leftarrow [8, 1024]$
$bestCfg \leftarrow \perp$, $bestPerf \leftarrow -\infty$
while remaining tuning time > 0 and $\mathcal{M}$ not converged **do**
  Choose ternary probes $M_1, M_2 \in \mathcal{M}$
  for $M \in \{M_1, M_2\}$ **do**
    $(perf_M, cfg_M) \leftarrow$ OPTIMIZE-EFC$(M, \mathcal{E}_c, \mathcal{D}, \mathcal{V}, C_{\text{perf}}, C_{\text{res}}, \mathcal{T}, \hat{h}_{bt}, \hat{h}_{is}, \mathcal{H})$
    if $perf_M > bestPerf$ **then**
      $bestPerf \leftarrow perf_M$; $bestCfg \leftarrow cfg_M$
    end if
  end for
  Update $\mathcal{M}$ toward the probe with higher validated performance
end while
return $bestCfg$

Algorithm 2 OPTIMIZE-EFC

Input: Fixed M; efc domain $\mathcal{E}_c$; dataset $\mathcal{D}$; backend $\mathcal{V}$; performance constraint C_{perf}; resource constraints C_{res}; tuning budget $\mathcal{T}$; models $\hat{h}_{bt}, \hat{h}_{is}$; history $\mathcal{H}$
Output: Best validated score $g(M)$ and configuration cfg_M

Shrink $\mathcal{E}_c$ using history $\mathcal{H}$
$g(M) \leftarrow -\infty$, $cfg_M \leftarrow \perp$
while remaining tuning time > 0 and $\mathcal{E}_c$ not converged **do**
  Choose ternary probes $efc_1, efc_2 \in \mathcal{E}_c$
  for $efc \in \{efc_1, efc_2\}$ **do**
    if $\hat{h}_{bt}(M, efc)$ or $\hat{h}_{is}(M)$ violates C_{res} **then**
      Record model-infeasibility in $\mathcal{H}$; **continue**
    end if
    $I \leftarrow$ BUILDINDEX$(\mathcal{V}, \mathcal{D}, M, efc)$
    Measure build time and index size; update $\hat{h}_{bt}, \hat{h}_{is}$
    if measured resource usage violates C_{res} **then**
      Record measured infeasibility in $\mathcal{H}$; **continue**
    end if
    $(efs^*, perf) \leftarrow$ OPTIMIZE-EFS$(I, C_{\text{perf}}, \mathcal{H})$
    if $efs^* = \perp$ **then**
      Record performance infeasibility in $\mathcal{H}$; **continue**
    end if
    Record $(M, efc, efs^*, perf)$ in $\mathcal{H}$
    if $perf > g(M)$ **then**
      $g(M) \leftarrow perf$; $cfg_M \leftarrow (M, efc, efs^*)$
    end if
  end for
  Update $\mathcal{E}_c$ toward the probe with higher validated performance
end while
return $(g(M), cfg_M)$

rule only reduces the search range; the final choice is still made using measured resource feasibility and validation performance.

The resource models and validation measurements play different roles. The models are used only to avoid unnecessary index builds under the conservative resource test. Candidates that pass this

Algorithm 3 OPTIMIZE-EFS

Input: Pre-built index $I(M, efc)$; performance constraint C_{perf}; history $\mathcal{H}$
Output: Boundary value efs^* and measured objective $perf$

$[efs_{\min}, efs_{\max}] \leftarrow$ shrink interval using $\mathcal{H}$ // history-based interval
if C_{perf} is $Recall \geq Recall_{\min}$ **then**
  $efs^*, perf \leftarrow$ BSEARCHMINFEASIBLEEFS$(I, [efs_{\min}, efs_{\max}], Recall_{\min})$
else if C_{perf} is $QPS \geq QPS_{\min}$ **then**
  $efs^*, perf \leftarrow$ BSEARCHMAXFEASIBLEEFS$(I, [efs_{\min}, efs_{\max}], QPS_{\min})$
end if
if no feasible *efs* exists **then**
  return $(\perp, -\infty)$
end if
return $(efs^*, perf)$

filter are still built, measured for build time and index size, and then evaluated by OPTIMIZE-EFS. Thus, early pruning reduces tuning cost, while the reported configuration is selected only from configurations explicitly validated against the user constraints.

5.3 Search for Optimal *efs*

Given a fixed constructed index $I(M, efc)$, CHAT only needs to tune the search-time parameter *efs*. Unlike M and *efc*, changing *efs* does not alter the graph, so all probes reuse the same built index. As analyzed in Section 3.1, *efs* induces an ordered feasibility boundary: larger *efs* broadens base-layer traversal, improving Recall but increasing query cost. CHAT therefore finds the boundary *efs* by binary search rather than by rebuilding or re-exploring construction parameters.

Algorithm 3 handles the two main performance constraints symmetrically. Under a Recall constraint, CHAT returns the smallest feasible *efs*, which maximizes QPS while meeting the Recall target. Under a QPS constraint, it returns the largest feasible *efs*, which maximizes Recall while meeting the throughput target. The resulting pair $(efs^*, perf)$ is returned to OPTIMIZE-EFC and used to compare candidate construction settings.

The initial search interval is tightened using previous boundary values stored in $\mathcal{H}$. Intuitively, stronger construction settings often require no larger *efs* to satisfy the same performance constraint, because better graph connectivity reduces the query-time exploration needed for Recall and can also tighten the QPS-feasible upper bound. Formally, for a prior configuration (M', efc') with boundary value $efs^*_{M',efc'}$, if $M' \geq M$ and $efc' \geq efc$, then $efs^*_{M,efc} \geq efs^*_{M',efc'}$; reversing both inequalities gives the corresponding upper bound. CHAT uses this history only to reduce the number of probes; the returned *efs* is still selected by measured feasibility on the current index. Thus, history accelerates binary search but does not replace validation.

The same boundary-search view supports query-subset constraints. For example, a deployment may require both average Recall and hard-query Recall: $\text{Recall}(Q_{\text{all}}) \geq \rho_{\text{all}}$ and $\text{Recall}(Q_{\text{hard}}) \geq \rho_{\text{hard}}$. Because Recall on any fixed query subset is monotone in *efs*, the conjunction remains a monotone predicate and can be handled by the same binary search. We identify Q_{hard} using Steiner-hardness [42].

Tail-latency constraints can be handled analogously. For a fixed index and validation workload, CHAT measures the requested tail metric, such as P95 or P99 latency, at each probed *efs*. Because increasing *efs* only enlarges the SEARCH-LAYER candidate budget, the permitted traversal work is monotone non-decreasing in *efs*; consequently, a fixed-workload latency quantile defines an upper-bound feasibility predicate up to measurement noise. A service-level constraint such as $\text{P}_\alpha(\text{Latency}) \leq \tau_\alpha$ is therefore handled using the same max-feasible binary search as $QPS \geq QPS_{\min}$,

returning the largest *efs* whose measured tail latency satisfies the constraint. If Recall lower bounds and tail-latency upper bounds are specified together, the feasible *efs* values form an interval, provided that the minimum Recall-feasible *efs* does not exceed the maximum latency-feasible efs. The selected point in this interval depends on the active objective.

5.4 Verifying Resource Constraints Satisfaction

Efficient hyperparameter tuning requires an early mechanism to rule out configurations that violate user-specified resource budgets. Performance feasibility (QPS/Recall) is evaluated during search on a fixed index (Section 3 and Algorithm 3). This subsection addresses *resource* feasibility. Build time and index size budgets cannot be verified for every candidate (M, efc) by fully constructing an index, as doing so would dominate tuning cost. We therefore employ a lightweight *resource-feasibility filter* that instantiates the algorithm-driven dependencies derived in Section 4 into closed-form surrogates and continuously calibrates their remaining constants using measurements obtained during the search process.

5.4.1 Closed-form Instantiation. Section 4 establishes two structural properties. First, build time is governed by a search-breadth component controlled by *efc* and a per-expansion scan component governed by realized out-degree, whose marginal growth in M saturates due to neighbor diversification and pruning. Second, index size is dominated by the cap-induced adjacency capacity and therefore scales linearly with M while remaining invariant to *efc*. We instantiate these dependencies using compact parametric surrogates that preserve monotonicity and saturation while leaving scale and offset to be determined by calibration.

We use two resource surrogates: $h_{\text{bt}}(M, efc)$ for build time and $h_{\text{is}}(M)$ for index size. For build time, we adopt a log-like surrogate in M to represent saturating realized connectivity, while retaining an explicit dependence on *efc* to capture increased exploration:

$$h_{\text{bt}}(M, efc) = \alpha_0 + \alpha_1\, efc + (\beta_0 + \beta_1\, efc) \cdot \log_2(1 + M) + \gamma\, efc \cdot \log_2^2(1 + M). \tag{6}$$

The linear term in *efc* captures the monotone increase in construction search budget (Figure 9(a)). The $\log_2(1 + M)$ terms represent a representative monotone concave dependence on M, consistent with the saturation of realized out-degree observed empirically (Figure 9(b)). Logarithmic growth is a convenient member of a broader class of concave monotone functions consistent with the algorithmic upper bounds derived in Section 4.1; it preserves sublinear growth while remaining analytically simple and stable under incremental calibration from a limited number of observations. The interaction term $efc \log_2^2(1 + M)$ captures that higher connectivity increases the number of candidate evaluations performed per unit of exploration during construction.

For index size, Section 4.2 implies a linear dependence on M and no dependence on *efc*:

$$h_{\text{is}}(M) = \eta_0 + \eta_1\, M. \tag{7}$$

Although *efc* may influence realized connectivity (Figure 9(a)), it does not affect the cap-induced adjacency budget that dominates index size (Figure 10); accordingly, *efc* is excluded from h_{is}.

5.4.2 Online Calibration from Search-time Measurements. The functional forms in (6) and (7) are fixed by the algorithmic structure; only their scale and offset parameters are backend-dependent. These parameters are calibrated using all configurations that are actually constructed during the search process. Whenever the search engine builds an index to evaluate performance feasibility, the resulting build time and index size are recorded and incorporated into calibration.

Let C denote the set of configurations (M_j, efc_j) that have been constructed so far during search, with measured build times T_j and index sizes S_j. Define the feature maps

$$\phi_{\text{bt}}(M, efc) = \left[1,\ efc,\ \log_2(1 + M),\ efc \log_2(1 + M),\ efc \log_2^2(1 + M)\right], \tag{8}$$

$$\phi_{\text{is}}(M) = [1,\ M]. \tag{9}$$

The surrogate parameters are obtained by solving linear least-squares problems over C:

$$h_{\text{bt}} = \langle \theta_{\text{bt}}, \phi_{\text{bt}} \rangle, \qquad h_{\text{is}} = \langle \theta_{\text{is}}, \phi_{\text{is}} \rangle, \tag{10}$$

where $\theta_{\text{bt}} = (\alpha_0, \alpha_1, \beta_0, \beta_1, \gamma)$ and $\theta_{\text{is}} = (\eta_0, \eta_1)$. Because calibration is linear in the number of accumulated measurements, its overhead is negligible relative to index construction.

As search progresses, the calibration set C naturally expands to cover a wider range of (M, efc) values, including regions where saturation in M becomes apparent. This incremental calibration improves alignment between the surrogates and backend-specific constants without altering the surrogate structure.

5.4.3 Conservative Feasibility Test. Let $T_{\max}$ and $S_{\max}$ denote user-specified budgets for build time and index size. To avoid accepting configurations whose resource usage is underestimated by the calibrated surrogates, CHAT applies one-sided safety margins to the build-time and index-size estimates. Each margin is computed from the positive relative underestimation residuals observed on constructed configurations in C and is recomputed whenever C grows. A candidate is pruned before construction only when the margin-adjusted build time or index size estimate exceeds the corresponding budget. This is a conservative false-feasible guard, not a deterministic no-pruning guarantee; larger margins reduce false-feasible risk at the cost of pruning aggressiveness.

5.5 Drift-Aware Retuning

CHAT supports retuning when the indexed corpus or query workload changes after an initial tuning run. Rather than restarting from scratch, it reuses the previous tuning history $\mathcal{H}$ as a cross-run warm start. The system refreshes the validation workload using recent queries and the current corpus, monitors Recall, QPS, and the *efs* feasibility boundary, and first reruns only Optimize-EFS at the previous (M, efc). Construction retuning is invoked only when the deployed configuration becomes infeasible or when changing *efs* alone is insufficient.

When construction retuning is needed, CHAT applies *Constraint-Directed Hard Pruning with Sentinel Safety* (CDHP). CDHP uses the feasibility-boundary shift in Section 3.6 to predict the movement of the best construction effort efc^*_M. Under a Recall constraint, a tighter boundary, i.e., a larger minimum *efs* to reach the Recall target, indicates that graph quality has become more important and suggests moving toward larger *efc*. Under a QPS constraint, a tighter boundary, i.e., a smaller maximum feasible *efs*, makes high-*efc* configurations cost-dominated earlier and suggests moving toward smaller *efc*. Relaxed boundaries induce the opposite directions.

Guided by this signal, CDHP keeps only the corresponding one-sided construction region: $efc \geq efc^{*,old}_M$ for a rightward move and $efc \leq efc^{*,old}_M$ for a leftward move. For unseen M values, the cutoff is inferred from the historical envelope of efc^*_M. CHAT then runs the standard Optimize-EFC and Optimize-EFS procedures within the retained domain, with the same resource filtering as in the initial run. Since the boundary-shift signal is structural rather than universal, CDHP places sentinel probes on the pruned side and expands the cached search domain if a sentinel is competitive within measurement tolerance; in the worst case, it completes the remaining search domain using cached measurements rather than restarting from scratch. For deletions, CHAT first cleanly rebuilds the updated corpus using the previous (M, efc) to distinguish in-place deletion artifacts from true hyperparameter drift.

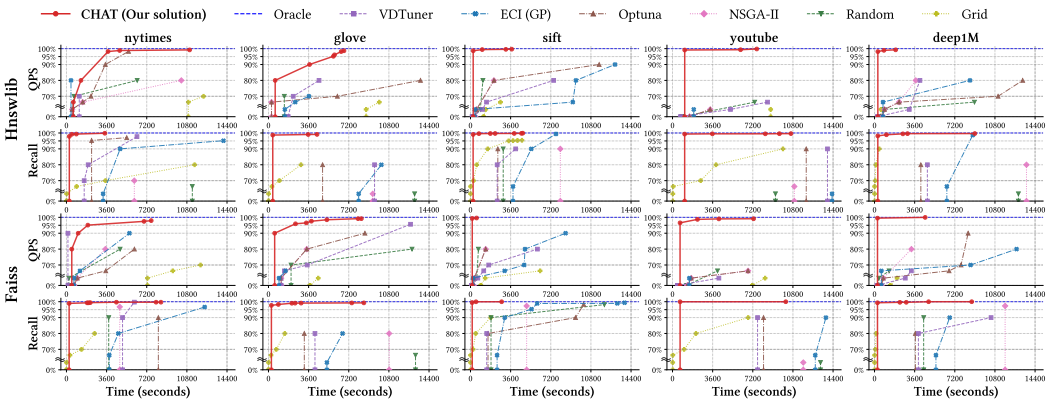


Fig. 12. Tuning performance over time, showing the best-so-far objective value normalized to the Oracle (exhaustive search). Each column corresponds to a dataset, and rows represent QPS (under Recall constraint) and Recall (under QPS constraint) for Hnswlib and Faiss backends, respectively. CHAT consistently achieves faster convergence and higher final performance than baseline methods across all datasets and constraint settings.

6 Experiments

6.1 Experimental Setup

We evaluated CHAT across multiple vector database backends, datasets, baselines, and metrics. We used Hnswlib [29] and Faiss [20] as reference HNSW implementations, as they expose the core behavior of HNSW with minimal system-level optimizations, allowing us to isolate the intrinsic performance characteristics analyzed in Section 3. To assess real-world applicability, we additionally evaluated CHAT on Milvus [39], a production-grade vector database that uses HNSW internally. Across all platforms, we varied three hyperparameters: the maximum node degree $M \in [4, 64]$, the construction candidate size $efc \in [8, 1024]$, and the search candidate size $efs \in [10, 1024]$.

We benchmarked CHAT on five representative datasets. Four of them—`nytimes`, `glove`, `sift`, and `deep1M`—were standard public datasets provided by ANN-Benchmarks [2], widely adopted for evaluating approximate nearest neighbor search (ANNS) algorithms. In addition, we constructed the `youtube` dataset [7], a realistic video retrieval workload built by encoding extracted I-frames from public high-view-count YouTube videos with OpenCLIP H-14 [17], providing a diverse and complementary evaluation point.

As comparison baselines, we considered a diverse set of black-box hyperparameter tuning methods, covering model-free search, sequential model-based optimization, evolutionary multi-objective search, constrained Bayesian optimization, and vector-database tuning. We included Random Search [4] and Grid Search [26], which respectively sample configurations uniformly and enumerate a predefined grid; both enforce constraints by discarding infeasible configurations. We also evaluated Optuna [1], a TPE-based sequential optimizer that proposes trials from past observations. In our implementation, Optuna handles constraints through a large-penalty scalar objective, assigning infeasible trials a penalty proportional to the degree of violation. NSGA-II [6] is a population-based evolutionary optimizer that uses Pareto dominance to balance objective value and constraint violation; we implement it as a bi-objective search that minimizes violation while maximizing the target metric. We additionally included Expected Constrained Improvement (ECI) [11], a Gaussian-process constrained Bayesian optimization method that models both objective

and feasibility and selects configurations by feasibility-weighted expected improvement. We also compared against VDTuner [44], a Bayesian-optimization framework designed for vector database tuning, restricted to the same HNSW parameter space for fairness. All baselines are treated as black-box tuners that do not inspect internal HNSW graph statistics.

For evaluation, we measured Recall@10 and Queries Per Second (QPS) to assess search accuracy and throughput, respectively, and recorded build time and index size to capture index construction cost and memory footprint. All experiments were conducted on a single Ubuntu server with an AMD Ryzen Threadripper 3970X CPU (32 cores, 64 threads, 3.7 GHz), 188 GB RAM, and 14 GB swap, using CPU-only execution to reflect typical vector database deployments. To reduce runtime noise, each index configuration was built once and evaluated with the same query workload over 10 runs, reporting the average Recall and QPS.

6.2 Constraint-Aware Tuning Results

We evaluated the effectiveness of CHAT in identifying optimal HNSW hyperparameter configurations under user-specified performance constraints, using a common 4-hour wall-clock cap for all methods. We considered two performance constraints: (1) Recall ≥ 0.95, where QPS is maximized, and (2) QPS $\geq \tau$, where Recall is maximized. CHAT always takes τ as a raw throughput target, e.g., QPS ≥ 5000; the 75th percentile is used only in our evaluation to choose comparably difficult targets across dataset-backend pairs. In practice, τ can come from a service SLO or a small pilot probe—a quick QPS measurement at minimal M, efc, and efs values that estimates the high-throughput scale of the workload.

To assess the effectiveness of our search strategy, we first compared CHAT against an Oracle (Exhaustive Search) that evaluates all valid hyperparameter configurations within the predefined search ranges under the same performance constraint, providing an upper bound on achievable performance. Despite this strong reference, CHAT achieved 98.03%–99.97% of the Oracle's QPS (under the Recall constraint) and 99.11%–99.99% of the Oracle's Recall (under the QPS constraint). These results show that CHAT identifies near-optimal configurations with only a fraction of the exhaustive search cost, quantifying a small optimality gap under strict tuning budgets.

Figure 12 summarizes the tuning performance of all methods by plotting the best objective values attained over time, normalized to the Oracle, across all datasets and constraint settings. In all cases, CHAT (shown in red) either converged to the best-performing configuration substantially faster than baseline methods or maintained superior objective performance throughout the tuning process. Under the Recall constraint (i.e., when maximizing QPS), CHAT outperformed all baseline methods by margins ranging from 1% to 45% (1% on the `nytimes` dataset with Hnswlib, and 45% on the `youtube` dataset with Hnswlib) in QPS. Under the QPS constraint (i.e., when maximizing Recall), CHAT again demonstrated strong performance, achieving Recall improvements of 0.6% to 11% (0.6% on the `nytimes` dataset with Faiss, and 11% on the `glove` dataset with Faiss) over the best baselines.

CHAT converges quickly enough for practical deployment. Using the time to reach 95% of the Oracle objective as a convergence metric, CHAT is the only method to reach this threshold within the 4-hour cap in 13 out of 20 settings; in the remaining cases, it reaches the target 1.5×–44× faster than the best competing baseline. The 4-hour cap is an evaluation budget rather than the runtime of CHAT: across the 10 backend–dataset combinations (two HNSW backends, Faiss and Hnswlib, evaluated on five datasets) under the Recall=0.95 constraint, CHAT reaches 99% of the Oracle objective in 48 minutes on average (range 13–123 minutes), consuming only 20% of the cap. Under a conservative no-serving-during-tuning break-even model, the mean break-even time is 3 hours and every backend–dataset combination recovers the tuning cost within 7 hours; over one week, the tuned configuration serves 0.25–2 billion additional queries per combination.

Table 1. Impact of analytic resource-constraint models on the tuning process. All values are reported as *with constraint model / without constraint model*. Resource-feasibility filtering substantially reduces unnecessary index constructions and total tuning time while fully preserving the final selected configuration.

Metric	**Index Size Constraint**		**Build Time Constraint**	
Engine	**Faiss**	**Hnswlib**	**Faiss**	**Hnswlib**
(a) Search Process				
Index Builds / Visited Configs	29.8 / 38.3	25.5 / 33.1	34.8 / 38.3	29.4 / 33.1
Pruned Before Build	22.44%	25.95%	4.20%	11.60%
Constraint-Violating Builds	4.0 / 10.5	4.0 / 10.4	2.9 / 9.7	2.4 / 9.3
Feasibility Coverage	0.9800	0.9817	0.9956	0.9873
(b) Search Results				
Total Search Time (s)	4791 / 6531	5042 / 7557	4787 / 6531	5327 / 7557
Average Performance Gain	1.36×	1.50×	1.36×	1.42×
Best Performance Gain	6.26×	5.21×	2.57×	2.32×
Performance Preservation	100.0%	100.0%	100.0%	100.0%

Beyond average Recall and QPS, we also evaluate query-side SLOs supported by Optimize-EFS in Section 5.3. For hard-query Recall, we rank validation queries by Steiner-hardness and define Hard-20 as the hardest 20% of queries. Average-only tuning satisfies the all-query Recall target but leaves Hard-20 Recall below 0.95 in nine of ten Faiss/Hnswlib settings. When Hard-20 Recall is added as an explicit constraint, CHAT satisfies both the all-query and Hard-20 Recall targets in all ten settings while achieving 98.21%–99.69% of the corresponding Oracle objective. For tail latency, we evaluate raw P95 and P99 latency upper-bound constraints. Across the same ten Faiss/Hnswlib settings, the structural properties exploited by CHAT continue to hold empirically under these query-side SLOs: the *efs* feasibility predicate remains monotone, and the fixed-M objective over *efc* and the outer objective over M continue to show dominant unimodal trends. Under the P95/P99 latency constraints, CHAT satisfies the requested tail-latency bounds while achieving 99.07%–99.99% of the corresponding Oracle objective.

Why CHAT Outperforms. CHAT matches each HNSW parameter to an observed structural property: binary search over monotone *efs* feasibility boundaries, ternary-style localization over dominant unimodal trends in M and *efc* (Section 3), and pre-build pruning using the resource models in Section 5.4. This design concentrates expensive index constructions near promising feasible regions. The baselines, though powerful general-purpose optimizers, are mismatched to this structure in complementary ways: Grid and Random Search explore without structural guidance; Optuna, VDTuner, and ECI spend evaluations learning generic objective and feasibility landscapes in a discrete, noisy space rather than exploiting these orderings; and NSGA-II allocates trials to maintaining Pareto diversity although the target here is a single constrained optimum.

6.3 Tuning under Resource Constraints

This section evaluates the effect of analytic resource constraint models on the tuning process. In contrast to Section 6.2, which compares tuning strategies under performance constraints (recall and QPS), this section isolates the contribution of the resource-feasibility filter itself. The resource constraint model is treated as a black-box module that can be attached to any search strategy; evaluating it separately avoids biasing cross-baseline comparisons and allows its standalone impact to be assessed under identical tuning budgets. Table 1 summarizes the results under two resource constraints (index size and build time) and two HNSW backends (Faiss and Hnswlib). All values are

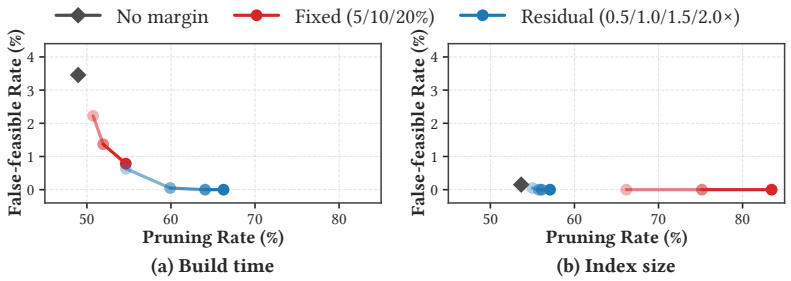


Fig. 13. Safety-margin sensitivity under resource constraints. The figure compares no-margin, fixed-margin, and residual-based variants in terms of pruning rate and false-feasible rate for build-time and index-size constraints. Marker opacity increases with the numeric margin value, so darker markers indicate larger margins.

reported as *with constraint model / without constraint model* and are averaged over the five datasets described in Section 6.1.

Search Process Efficiency. Table 1(a) summarizes the tuning process. *Index Builds / Visited Configs* shows that resource-feasibility filtering substantially reduces the number of index constructions required to explore the same configuration space by pruning infeasible candidates prior to construction. This behavior is reflected in the *Pruned Before Build* rate (4.2%–26.0%) and the consistently lower number of *Constraint-Violating Builds* when constraint models are applied. The safety of early pruning is quantified by *Feasibility Coverage*, computed over all configurations visited during the search. Coverage exceeds 0.98 across all settings, indicating that the constraint models remain conservative while effectively filtering infeasible regions.

Impact on Search Outcomes. As shown in Table 1(b), *Total Search Time*, which is dominated by index construction, is consistently lower when resource-feasibility filtering is enabled. This reduction is consistent with the decrease in index builds observed in Table 1(a), indicating that pruning infeasible configurations prior to construction is the primary source of the time savings.

Despite the reduced exploration cost, the quality of the final tuning result is fully preserved. As shown in Table 1(b), *Performance Preservation* is 100% across all settings, indicating that the configuration selected with resource constraints achieves identical performance to that selected without constraints. Across all five datasets, *Average Performance Gain* remains stable in the range of 1.36×–1.50×, showing that feasibility filtering does not restrict access to competitive configurations. Figure 13 isolates the safety-margin mechanism by comparing no-margin, fixed-margin, and residual-based variants. No-margin pruning is more exposed to false-feasible decisions, while fixed margins require a hand-tuned conservatism level. Residual-based margins adapt to observed underestimation residuals, reducing false-feasible risk while maintaining high pruning rates. Together with the performance preservation and best-case gains in Table 1, this shows that resource-feasibility filtering lowers tuning cost without blocking high-performing configurations.

6.4 Robustness of Structural Trends and Measurements

CHAT uses ternary-style localization over measured objective values, but it does not assume convexity or universal unimodality. Section 3 already verifies that changing query parallelism shifts absolute QPS but preserves the *efs* feasibility boundary and the dominant trends over *efc* and M. Here we audit two further risks: whether measurement noise perturbs the pairwise comparisons used during search, and whether the dominant structural trends—formalized as sufficient conditions in Lemmas 3.1 and 3.2—persist across different data geometries.

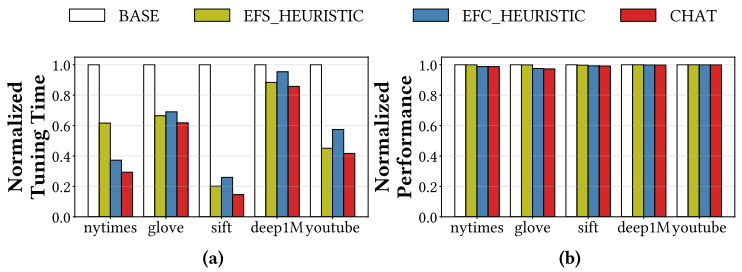


Fig. 14. Impact of heuristic search-space reduction. (a) Tuning time. (b) Average performance (Recall and QPS). All values are normalized to BASE.

We first evaluate robustness to measurement noise using the 10-run average as a reference. We recompute configuration scores using 1, 3, and 5 runs, and measure QPS P_{95} relative error, pairwise comparison flip rate, same-or-adjacent optimum stability, and retained objective value. Pairwise flips exclude candidate pairs whose reference objective gap is below 0.5%, so near-ties are not counted as unstable decisions. Even with a single run, the QPS P_{95} error is 1.86%, the pairwise flip rate is only 1.1%, the selected optimum remains the same or adjacent grid point in 98.8% of cases, and the selected configuration retains 99.21% of the 10-run reference objective. With five runs, these improve to 0.62%, 0.4%, 99.7%, and 99.79%, respectively. Thus, runtime noise does not materially perturb the pairwise comparisons used by CHAT.

We further stress-test the structural trends on six synthetic workloads that vary cluster separation/overlap, cluster-size imbalance, anisotropy, dimensionality, and cluster structure. For the well-separated, overlapping, imbalanced, and anisotropic workloads, the trends exploited by CHAT continue to hold empirically under both Recall and QPS constraints: the *efs* feasibility predicate remains monotone, fixed-M slices over *efc* remain dominantly unimodal, and the outer envelope $g(M)$ remains dominantly unimodal. In these cases, HNSW construction exposes useful routing alternatives early, after which additional construction effort becomes increasingly redundant.

The high-dimensional and uniform workloads expose the main failure mode. Distance concentration or the absence of cluster structure can delay useful routing edges until larger construction budgets, producing a delayed graph-quality gain and a second objective peak. This effect is most visible under a Recall constraint, where the delayed gain can reduce the minimum feasible *efs* and recover QPS; under a QPS constraint, the same branch must reduce *efs* to preserve throughput, so the gain is dampened. This behavior is consistent with the sufficient conditions in Lemmas 3.1 and 3.2; it illustrates cases where the diminishing-utility condition required by these lemmas fails. CHAT is therefore not a global optimizer for arbitrary multi-modal objectives, but a sample-efficient search procedure for dominant discrete trends induced by standard HNSW mechanisms. In the high-dimensional counterexample, CHAT selects $(M, efc, efs) = (16, 400, 160)$, while the full-grid Oracle selects $(32, 660, 320)$; both satisfy the Recall constraint. CHAT achieves QPS 4,970 versus the Oracle's 5,280, retaining 94.1% of the Oracle objective. Thus, when the sufficient conditions fail, ternary-style localization may converge to a local peak, but best-so-far tracking over validated configurations substantially limits the loss.

6.5 Impact of Heuristic Search Space Reduction

To assess the effectiveness of the two heuristics used in CHAT for narrowing the search ranges of *efs* and *efc*, we performed an ablation study comparing four configurations: (1) CHAT, which incorporates both heuristics described in Sections 5.2 and 5.3; (2) EFS_HEURISTIC, which applies

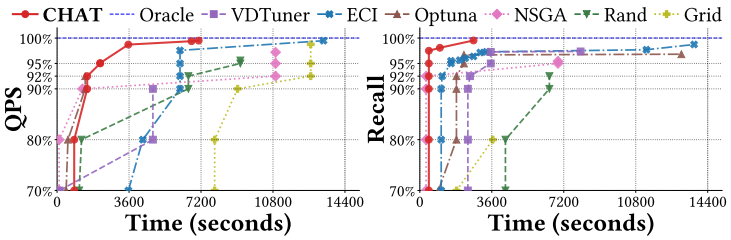


Fig. 15. Performance of CHAT and baselines on the nytimes dataset using Milvus.

only the *efs* range narrowing heuristic; (3) EFC_HEURISTIC, which applies only the *efc* heuristic; and (4) BASE, which disables both heuristics and performs exhaustive search over the entire range using binary and ternary search procedures. The performance metric was defined as the aggregated objective value under a Recall threshold of 0.95 and a QPS threshold set at the 75th percentile.

Figure 14(a) shows that each heuristic individually reduces tuning time relative to BASE, with the *efs* heuristic achieving speedups of 1.13×–4.96× and the *efc* heuristic yielding 1.05×–3.85× improvements. When combined, CHAT attains speedups of 1.17×–6.85×, indicating a complementary effect between the two heuristics. Figure 14(b) further shows that these gains do not compromise solution quality: CHAT achieves normalized performance of 98.03%–99.9%, while EFS_HEURISTIC and EFC_HEURISTIC attain 99.4%–99.9% and 98.7%–99.9%, respectively, demonstrating that heuristic search space reduction substantially accelerates tuning with only negligible loss in objective performance.

6.6 Evaluation on a Commercial VectorDB

To validate the generality and practical effectiveness of our approach, we conducted additional experiments on Milvus, a widely adopted commercial vector database, beyond the open-source HNSW implementations evaluated earlier. We followed the same experimental setup described in Section 6.1. Specifically, we applied a minimum Recall constraint of 0.95, or a minimum QPS constraint defined as the 75th percentile of all configurations for each dataset, with a fixed tuning time budget of four hours.

As shown in Figure 15, which presents results on the nytimes dataset, our method consistently outperformed all baseline methods under both types of constraints[1]. Under the Recall constraint, it achieved up to 99.2% and at least 96.8% of the Oracle's optimal QPS. Under the QPS constraint, it successfully matched the Oracle's optimal Recall. These results demonstrate that our method remains effective and stable even in production-grade vector database environments, maintaining high performance and constraint satisfaction under limited tuning budgets.

7 Related Work

VectorDB and ANN Indexing. Vector databases and approximate nearest neighbor (ANN) search methods are central to high-dimensional data retrieval [25, 40]. Early methods such as locality-sensitive hashing (LSH) [16, 18] and product quantization (PQ) [21, 33] introduced efficient hashing and compression techniques. Graph-based ANN indices, including Hierarchical Navigable Small World (HNSW) [29] and Navigating Spreading-out Graph (NSG) [9], improve recall–speed trade-offs by organizing points into navigable proximity graphs [8]. DiskANN [37] scales graph-based search to billion-scale datasets through memory–disk optimized storage and retrieval. Libraries such as

[1]We observed similar trends on the other datasets used in this experiment.

Faiss [20] and ScaNN [14] combine quantization and GPU acceleration for large-scale search, with adoption in systems such as Milvus [39].

Database Auto-Tuning. Automated database tuning has progressed from heuristic methods to machine-learning-based approaches. OtterTune [38] introduced supervised learning for knob tuning, while QTune [24] extended this direction with reinforcement learning (RL) for query-level precision. Recent works emphasize constraint-awareness and multi-objective optimization.

8 Conclusion

We presented CHAT, an API-level black-box but HNSW-structure-aware tuning framework that exploits monotone search-time feasibility boundaries, dominant unimodal trends over construction parameters, and separable resource dependencies to guide a hierarchical search with resource surrogates. Across datasets and HNSW backends, CHAT efficiently identifies and validates configurations that satisfy user-defined performance and resource constraints, offering a practical path toward predictable tuning for production HNSW deployments.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF; RS-2025-00560762 and RS-2024-00414981), the Institute of Information & Communications Technology Planning & Evaluation (IITP; RS-2024-00454666, RS-2025-25442338, RS-2021-II211343, RS-2026-25551943, RS-2026-25518317, and RS-2024-00397085), the Ministry of Health and Welfare under the ARPA-H BAYS Project (RS-2025-25455095), the Ministry of Science and ICT under the Advanced GPU Utilization Support Program, and Samsung Electronics Co., Ltd. (IO251215-14535-01). The authors thank Samsung MAX Lab for providing research infrastructure. J. Do is affiliated with the Automation and Systems Research Institute (ASRI), Seoul National University (SNU).

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 2623–2631. doi:10.1145/3292500.3330701

[2] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374. doi:10.1016/j.is.2019.02.006

[3] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. 2023. A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity. arXiv:2302.04023 [cs.CL] https://arxiv.org/abs/2302.04023

[4] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13, null (Feb. 2012), 281–305.

[5] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*. PMLR, 2206–2240.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. doi:10.1109/4235.996017

[7] dnotitia. [n. d.]. SeahorseDB-dataset. https://huggingface.co/datasets/dnotitia/SeahorseDB-dataset/tree/main. Accessed: 2025-07-16.

[8] Cong Fu, Changxu Wang, and Deng Cai. 2022. High Dimensional Similarity Search With Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2022), 4139–4150. doi:10.1109/TPAMI.2021.3067706

[9] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2025. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. arXiv:1707.00143 [cs.LG] https://arxiv.org/abs/1707.00143

[10] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] https://arxiv.org/abs/2312.10997

[11] Jacob R Gardner, Matt J Kusner, Zhixiang Eddie Xu, Kilian Q Weinberger, and John P Cunningham. 2014. Bayesian optimization with inequality constraints.. In *ICML*, Vol. 2014. 937–945.

[12] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization for Approximate Nearest Neighbor Search. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2946–2953. doi:10.1109/CVPR.2013.379

[13] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, Zhenshan Cao, Yanliang Qiao, Ting Wang, Bo Tang, and Charles Xie. 2022. Manu: A Cloud Native Vector Database Management System. arXiv:2206.13843 [cs.DB] https://arxiv.org/abs/2206.13843

[14] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 3887–3896. https://proceedings.mlr.press/v119/guo20h.html

[15] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2025. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ACM Transactions on Information Systems* 43, 2 (Jan. 2025), 1–55. doi:10.1145/3703155

[16] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proc. VLDB Endow.* 9, 1 (Sept. 2015), 1–12. doi:10.14778/2850469.2850470

[17] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. 2021. *OpenCLIP*. doi:10.5281/zenodo.5143773 If you use this software, please cite it as below..

[18] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (Dallas, Texas, USA) *(STOC '98)*. Association for Computing Machinery, New York, NY, USA, 604–613. doi:10.1145/276698.276876

[19] Gautier Izacard and Edouard Grave. 2022. Distilling Knowledge from Reader to Retriever for Question Answering. arXiv:2012.04584 [cs.CL] https://arxiv.org/abs/2012.04584

[20] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547. doi:10.1109/TBDATA.2019.2921572

[21] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128. doi:10.1109/TPAMI.2010.57

[22] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through Memorization: Nearest Neighbor Language Models. arXiv:1911.00172 [cs.CL] https://arxiv.org/abs/1911.00172

[23] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[24] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2118–2130.

[25] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2020), 1475–1488. doi:10.1109/TKDE.2019.2909204

[26] Petro Liashchynskyi and Pavlo Liashchynskyi. 2019. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. arXiv:1912.06059 [cs.LG] https://arxiv.org/abs/1912.06059

[27] Ting Liu, Andrew W. Moore, Alexander Gray, and Ke Yang. 2004. An investigation of practical approximate nearest neighbor algorithms. In *Proceedings of the 18th International Conference on Neural Information Processing Systems* (Vancouver, British Columbia, Canada) *(NIPS'04)*. MIT Press, Cambridge, MA, USA, 825–832.

[28] Le Ma, Ran Zhang, Yikun Han, Shirui Yu, Zaitian Wang, Zhiyuan Ning, Jinghan Zhang, Ping Xu, Pengjiang Li, Wei Ju, Chong Chen, Dongjie Wang, Kunpeng Liu, Pengyang Wang, Pengfei Wang, Yanjie Fu, Chunjiang Liu, Yuanchun Zhou, and Chang-Tien Lu. 2025. A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge. arXiv:2310.11703 [cs.DB] https://arxiv.org/abs/2310.11703

[29] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[30] Marius Muja and David G. Lowe. 2014. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 11 (2014), 2227–2240. doi:10.1109/TPAMI.2014.2321376

[31] P. Ngatchou, A. Zarei, and A. El-Sharkawi. 2005. Pareto Multi Objective Optimization. In *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*. 84–91. doi:10.1109/ISAP.2005.1599245

[32] OpenAI. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[33] Zhibin Pan, Liangzhuang Wang, Yang Wang, and Yuchen Liu. 2020. Product Quantization with Dual Codebooks for approximate Nearest Neighbor Search. *Neurocomputing* 401 (03 2020). doi:10.1016/j.neucom.2020.03.016

[34] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. 2016. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* 104, 1 (2016), 148–175. doi:10.1109/JPROC.2015.2494218

[35] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2024. Replug: Retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 8371–8384.

[36] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (Lake Tahoe, Nevada) *(NIPS'12)*. Curran Associates Inc., Red Hook, NY, USA, 2951–2959.

[37] Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. 2019. *DiskANN: fast accurate billion-point nearest neighbor search on a single node.* Curran Associates Inc., Red Hook, NY, USA.

[38] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1009–1024. doi:10.1145/3035918.3064029

[39] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2614–2627. doi:10.1145/3448016.3457550

[40] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.* 14, 11 (July 2021), 1964–1978. doi:10.14778/3476249.3476255

[41] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando De Freitas. 2016. Bayesian optimization in a billion dimensions via random embeddings. *J. Artif. Int. Res.* 55, 1 (Jan. 2016), 361–387.

[42] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. 2024. Steiner-hardness: A query hardness measure for graph-based ann indexes. *Proceedings of the VLDB Endowment* 17, 13 (2024), 4668–4682.

[43] Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. 2025. Hallucination is Inevitable: An Innate Limitation of Large Language Models. arXiv:2401.11817 [cs.CL] https://arxiv.org/abs/2401.11817

[44] Tiannuo Yang, Wen Hu, Wangqi Peng, Yusen Li, Jianguo Li, Gang Wang, and Xiaoguang Liu. 2024. Vdtuner: Automated performance tuning for vector data management systems. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4357–4369.

[45] Tong Yu and Hong Zhu. 2020. Hyper-Parameter Optimization: A Review of Algorithms and Applications. arXiv:2003.05689 [cs.LG] https://arxiv.org/abs/2003.05689

[46] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2025. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL] https://arxiv.org/abs/2303.18223

[47] Wenyang Zhou, Yuzhi Jiang, Yingfan Liu, Xiaotian Qiao, Hui Zhang, Hui Li, and Jiangtao Cui. 2024. Auto-Tuning the Construction Parameters of Hierarchical Navigable Small World Graphs. doi:10.2139/ssrn.4734062

Received 17 January 2026; revised 19 May 2026; accepted 12 June 2026