Scaling Laws for Grid-Based Approximate Nearest Neighbor Search in High Dimensions

author names withheld

Under Review for the Workshop on High-dimensional Learning Dynamics, 2026

Abstract

Grid-based approaches to approximate nearest neighbor (ANN) search have been absent from modern scaling analyses. We present a systematic characterization of a multiprobe grid algorithm with respect to dataset size N and dimensionality d. Our experiments reveal a previously unreported d-scaling crossover on the GloVe embedding family, in which multiprobe grid search maintains an approximately constant dimensional scaling exponent while other graph-, tree-, and partitioning-based methods exhibit degrading throughput. The advantage comes with near-linear query scaling in N, but also with lower indexing cost than competing ANN methods. Our results suggest that grid-based methods such as multiprobe grid may be competitive in rebuild-heavy or high-dimensional settings where indexing cost and dimensional robustness dictate performance. More broadly, recent work has formalized self-attention as an ANN operation. Thus, the N- and d-scaling properties of ANN algorithms may guide cost analysis of efficient transformer architectures.

1. Introduction

Approximate nearest neighbor (ANN) search is central to modern machine learning systems operating on large, high-dimensional datasets [10]. Beyond its classical role in retrieval, ANN has emerged as a computational primitive in transformer architectures. Recent studies formalize self-attention as an ANN operation over token embeddings, enabling sub-quadratic approximations to full attention [13, 19, 21]. As empirical scaling laws guide transformer design [20], the scaling behavior of ANN algorithms is increasingly relevant to the development of more efficient transformer architectures. In particular, scaling dataset size N stresses candidate filtering and indexing efficiency, while scaling dimensionality d exacerbates the curse of dimensionality and degrades the effectiveness of geometric pruning. As a result, ANN methods including graph-, tree-, and partitioning-based approaches exhibit regime-dependent trade-offs in which no method dominates universally [14, 27].

Grid-based methods (a variant of partitioning approaches) played a foundational role in early ANN theory [8, 15], but remain undercharacterized relative to graph- and tree-based methods in modern scaling analyses (see Appendix A for Related Work). In this work, we characterize N and d scaling relationships for a multiprobe grid algorithm (overview in Section 2) that decouples cell selection from d, where cell selection denotes the query-time procedure that identifies which grid cells supply candidate vectors. This decoupling is achieved by performing cell selection in a PCA-reduced subspace $\mathbb{R}^m$ while re-ranking candidates in $\mathbb{R}^d$. We show that while multiprobe grid exhibits near-linear scaling with N, its d-scaling behavior remains favorable as dimensionality increases. This work situates grid-based methods within the broader ANN design space and highlights regimes in which they offer competitive trade-offs.

2. Theoretical scaling model for multiprobe grid search

We provide a theoretical overview for the multiprobe grid algorithm, deriving a closed-form relationship between query cost and recall. Appendix B contains the full discussion and Figure D1 provides an overview schematic for the algorithm.

Cost model. After PCA projection to $\mathbb{R}^m$, the space is partitioned into G^m cells with N/G^m points each. A query q probes a total of $1 \le n_{\text{probe}} \le 2^m$ cells, consisting of the home cell c_h (defined in Appendix B.1) and up to $2^m - 1$ neighboring cells ordered ascending by wall distance w_i^2 (Appendix B.2). Each probed cell contributes, on average, N/G^m candidates to the cost:

$$\text{cost} \;=\; 1/\text{QPS} \;=\; K \cdot n_{\text{probe}} \cdot \frac{N}{G^m}. \tag{1}$$

Recall model. Let $P_i = \Pr[x^* \in c_i]$ be the probability that the true nearest neighbor lies in the i-th probed cell. Under the uniform distribution assumption, P_i decreases monotonically in w_i^2, and we adopt the exponential approximation (Appendices B.3 and B.6).

$$P_i \;\approx\; P_h\, e^{-\mu w_i^2}, \qquad P_h := P_0,\ \mu > 0 \tag{2}$$

Closed form. Using a linear approximation $\mathbb{E}[w_i^2] \approx \theta i$ (Appendix B.4) on the mean gap yields

$$R(n_{\text{probe}}) \;=\; P_h \sum_{i=0}^{n_{\text{probe}}-1} \Delta^i \;=\; P_h \cdot \frac{1-\Delta^{n_{\text{probe}}}}{1-\Delta}, \Delta := e^{-\mu\theta} \in (0,1) \tag{3}$$

Log-linearity. Solving equation 3 for n_{probe}, substituting into equation 1, and inverting gives

$$\text{QPS}(R) \;=\; \frac{G^m}{KN} \cdot \frac{|\ln \Delta|}{-\ln(1 - R/R_{\max})}. \tag{4}$$

Taking logs, we arrive at $\log \text{QPS}(R) \approx \log(\text{const}') - \log R$. Thus, our framework predicts a log-linear relationship between QPS and recall, arising from the exponential decay of nearest-neighbor membership probability across probed cells and the linear growth of candidate set size with n_{probe}.

3. Empirical scaling laws

We evaluate our multiprobe grid algorithm against four baselines representing major ANN families: Voyager (graph-based) [3], PyNNDescent (graph-based) [23], Annoy (tree-based) [2], and FAISS-IVF (quantization-based partitioning) [1]. Algorithms are evaluated using the `ann-benchmarks` framework [5, 6], executing each method in a dedicated Docker container to enforce single-CPU isolation. The baseline implementations use highly optimized C++ implementations, while our multiprobe grid implementation is a Python-based proof-of-concept. Consequently, the measured QPS for multiprobe grid likely understates the throughput attainable with a comparably optimized implementation. Nonetheless, we expect that the relative scaling trends reported here remain informative, given that multiprobe grid's per-query time is primarily spent in NumPy/BLAS (see profiling study in Appendix C). Baseline algorithms use their established `ann-benchmarks` parameter sweeps, which represent well-explored, community-validated search spaces; the multiprobe grid algorithm required bespoke NSGA-II tuning to identify competitive configurations (Appendix C for implementation details).

3.1. Pareto fronts reveal a log-linear throughput-recall relationship for multiprobe grid

Figure 1 shows Pareto fronts on GloVe-200-angular ($N = 1.18 \times 10^6$ points) [5, 6, 24] for all five algorithms. Consistent with Section 2, multiprobe grid shows that $\log(\text{QPS})$ decreases linearly with increasing recall, indicating that performance is determined by grid geometry. At recall@k=10 > 0.9, multiprobe's throughput converges toward brute-force: achieving high recall requires either coarse PCA projections (m=2) that produce densely-populated cells, or exhaustive multiprobing at higher m, resulting in ranking a large fraction of the dataset. We show individual Pareto fronts at each subsampled N for GloVe-200 and d (GloVe-25, 50, 100, 200) in Figures D2–D3.

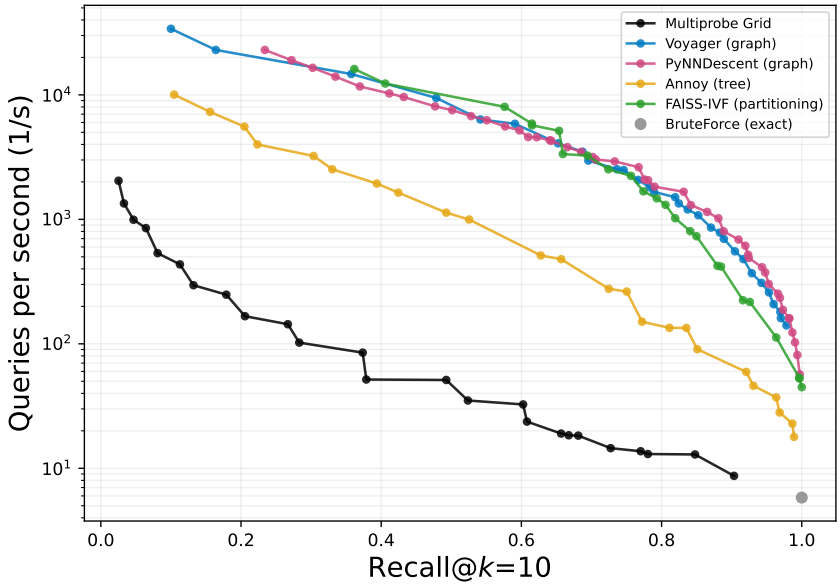

Figure 1: Pareto fronts for GloVe-200-angular ($d = 200$, $N = 1.18 \times 10^6$).

3.2. N-scaling: multiprobe grid scales near-linearly with dataset size

3.2.1. GLOVE DATASET (ANGULAR)

Figure 2a shows how the N-scaling exponent α_N varies with recall across all five ANN algorithms. At recall@k=10 $= 0.80$, multiprobe grid exhibits $\alpha_N = -0.94$ ($R^2 = 1.00$), indicating near-linear degradation of QPS with dataset size. In contrast, we observe sublinear scaling for the baselines, ranging from $\alpha_N = -0.44$ to -0.59. Figure D4 shows the $\log_{10}(\text{QPS})$ vs. $\log(N)$ plots at each target recall@k=10 used to derive the exponents in Figure 2a.

The near-linear N-scaling of multiprobe is mechanistically expected: for fixed grid parameters (m, G), the candidate count per cell grows as N/G^m, and re-ranking cost is linear in candidate count. As recall targets increase, all algorithms' exponents trend toward $\alpha_N = -1$, reflecting how perfect recall requires exhaustive search. Multiprobe's exponent is already near its asymptotic value across the full recall range, while baseline methods degrade more rapidly at higher recalls.

3.2.2. SIFT-128 DATASET (EUCLIDEAN)

To investigate whether the observed N-scaling relationships generalize beyond word embeddings, we repeat the analysis on SIFT-128-euclidean (image descriptors, $d = 128$) [18]. The log-linear Pareto behavior of multiprobe grid (Figure D5) and relative trends in α_N are preserved (Figures D6–D7). At recall@k=10 $= 0.80$, multiprobe grid exhibits a scaling exponent $\alpha_N = -0.83$ ($R^2 = 1.00$), while baseline algorithms fall between -0.27 and -0.39. The consistency across data modality (images vs. words) and similarity metric (Euclidean vs. angular) suggests that N-scaling in multiprobe grid is intrinsic to the algorithm rather than dataset-specific.

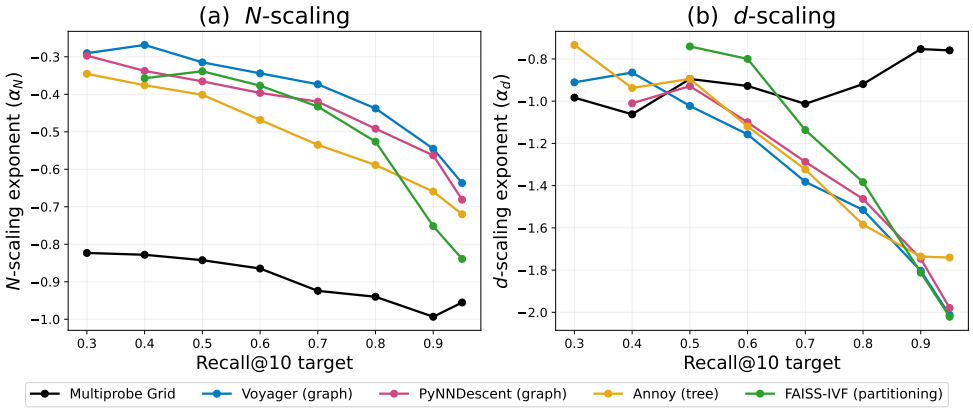

Figure 2: **Scaling exponents vs. recall@k =10 for all five algorithms**. **(a)** N-scaling exponent α_N. Datasets: GloVe-200-angular ($d = 200$) subsampled at varying N. **(b)** d-scaling exponent α_d. Datasets: GloVe-25-, 50-, 100-, and 200-angular ($N = 1.18 \times 10^6$).

3.3. d-scaling: multiprobe grid is increasingly competitive for higher dimensional data

Figure 2b shows how the d-scaling exponent α_d varies with recall, revealing a non-monotonic crossover between multiprobe grid and the other four algorithms. As recall increases beyond 0.7, α_d of all algorithms besides multiprobe grid steepen dramatically, reflecting lower throughput with increasing dimensionality. Meanwhile, α_d for multiprobe grid remains relatively flat. The contrast is notable given that prior empirical studies, including our own in Figure 2a, typically find N-scaling exponents to be remarkably consistent across algorithm families [26]. The d-scaling crossover subverts this universality - the same algorithm that exhibits the least favorable N-scaling simultaneously exhibits the most favorable d-scaling at high recall. Figure D8 shows the $\log_{10}(\text{QPS})$ vs. $\log(d)$ plots at each target recall@k=10 used to derive the exponents in Figure 2b.

The crossover in α_d arises from how each algorithm interacts with d. Graph-based methods (Voyager, PyNNDescent) construct and traverse proximity graphs in the full d-dimensional space. At high recall, accurate retrieval requires exploring larger neighborhoods and more backtracking. Similarly, tree-based (Annoy) and quantization-based partitioning (FAISS-IVF) methods operate on the raw d-dimensional vectors, where pruning effectiveness degrades as d grows. In contrast, multiprobe grid performs cell selection in a PCA subspace of dimensionality $m \ll d$. Although PCA retains a smaller fraction of total variance as d increases (Figure D9), the Pareto-optimal (m, G) adapts with both d and the target recall rather than remaining fixed (Appendix Table E1).

At higher d, fewer total cells G^m are favored, concentrating more candidates per probed cell to preserve recall. Because re-ranking grows only linearly in candidate count, the query cost scales less aggressively with d than for methods that traverse the full d-dimensional space. We note that our d-scaling characterization is bounded by $d = 200$, the maximum dimensionality available in the GloVe family (the only d-varying dataset family provided in `ann-benchmarks`). While this range demonstrates the α_d crossover, extending our analysis to higher-d regimes (e.g., $d \geq 512$ for modern transformer embeddings) is a key direction for future work (Section 3.5).

3.4. Total cost analysis reveals competitive regimes for multiprobe grid

Figure D10 shows build time scaling with N for the Pareto-optimal configuration achieving recall@k = 10 ≈ 0.80. Multiprobe grid and FAISS-IVF are fastest across all N. At $N = 1.18 \times 10^6$, multiprobe grid builds its index in 4–36 sec depending on the selected hyperparameter configuration (4 sec for the smallest config, m=2, G=4; 36 sec for the largest, m=7, G=7). We measure 206 sec for FAISS-IVF, 333 sec for Annoy, 500 sec for PyNNDescent, and 1,569 sec for Voyager. Multiprobe grid indexing requires only PCA fitting, cell assignment, and BFS precomputation, whereas baselines rely on data-dependent operations such as graph insertion by repeated nearest-neighbor queries (Voyager, PyNNDescent).

Recent work by Sun et al. established power-law scaling relationships for different ANN methods like brute-force, partitioning-based, and graph-based [26]. The study proposed a framework for the total cost J of ANN search that is a function of query-time compute, indexing, and storage costs: $J(N) = f_I \cdot I(N) + f_C \cdot C(N) + S(N)$, where I, C, and S denote indexing, compute, and storage costs, and f_I, f_C their respective frequency of occurrence. We adapt the cost framework with one substitution, reporting memory footprint $M(N)$ in place of $S(N)$, because `ann-benchmarks` measures process RSS during indexing rather than on-disk storage size. Table E2 shows empirical scaling exponents at recall@k=10 $= 0.80$ on GloVe-200-angular. $C(N)$ corresponds directly to the values shown in Figure 2a for recall@k=10 $= 0.80$. $I(N)$ and $M(N)$ are measured directly from the Pareto-optimal configuration at each N (Figures D10 and D11). At $N = 1.18 \times 10^6$, multiprobe grid builds its recall@k=10 $= 0.80$ index $\sim$190$\times$ faster than Voyager (8.4 sec vs. 1,569 sec), but exhibits per-query latency approximately 120$\times$ slower. Thus, multiprobe grid achieves lower total cost when the rebuild-to-query ratio f_I/f_C is sufficiently high: approximately one index rebuild per 2,600–20,400 queries depending on the baseline algorithm (see Appendix C.4). This regime may include applications in recommendation systems and retrieval-augmented generation.

3.5. Implications for high-dimensional learning systems

Building on the formalization of attention as ANN search [13, 19, 21], our scaling characterization maps dataset size N to sequence length, native dimensionality d to token embedding dimension, and index build cost to KV-cache or index construction overhead. The d-scaling crossover we identify on the GloVe family may bear relevance for transformer embeddings operating in higher-dimensional spaces (typically $d \geq 512$); whether the α_d crossover persists in this regime is an empirical question for future work. More broadly, our results underscore that ANN method selection should consider N- and d-scaling relationships, total cost J, and operational frequencies such as rebuild rate.

References

[1] facebookresearch/faiss, April 2026. URL https://github.com/facebookresearch/faiss. original-date: 2017-02-07T16:07:05Z.

[2] spotify/annoy, April 2026. URL https://github.com/spotify/annoy. original-date: 2013-04-01T20:29:40Z.

[3] spotify/voyager, April 2026. URL https://github.com/spotify/voyager. original-date: 2023-04-13T13:07:27Z.

[4] Sunil Arya. An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. *Journal of the ACM*, 45(6):891–923, November 1998.

[5] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87: 101374, January 2020. ISSN 03064379. doi: 10.1016/j.is.2019.02.006. URL https://linkinghub.elsevier.com/retrieve/pii/S0306437918303685.

[6] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Reproducibility protocol for ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor search algorithms. 2021. URL https://zenodo.org/records/4607761.

[7] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975. ISSN 0001-0782, 1557-7317. doi: 10.1145/361002.361007. URL https://dl.acm.org/doi/10.1145/361002.361007.

[8] Timothy M Chan. Approximate Nearest Neighbor Queries Revisited. *Proceedings of the thirteenth annual symposium on Computational Geometry (SoCG 1997)*, 1997. doi: https://doi.org/10.1145/262839.263001.

[9] Morgan Roy Cooper and Mike Busch. Capacity-Limited Failure in Approximate Nearest Neighbor Search on Image Embedding Spaces. *Journal of Imaging*, 12(2):55, February 2026. ISSN 2313-433X. doi: 10.3390/jimaging12020055. URL https://www.mdpi.com/2313-433X/12/2/55.

[10] Trevor Darrell, Gregory Shakhnarovich, and Piotr Indyk, editors. *Nearest-neighbor methods in learning and vision: theory and practice*. Neural information processing series. MIT Press, Cambridge, Massachusetts, 2005. ISBN 978-0-262-19547-8 978-0-262-25695-7.

[11] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, Brooklyn New York USA, June 2004. ACM. ISBN 978-1-58113-885-6. doi: 10.1145/997817.997857. URL https://dl.acm.org/doi/10.1145/997817.997857.

[12] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. *Proceedings of the 25th VLDB Conference*, 1999.

[13] Themistoklis Haris. kNN Attention Demystified: A Theoretical Exploration for Scalable Transformers. *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2411.04013.

[14] Patrick Iff, Paul Bruegger, Marcin Chrapek, David Kochergin, Maciej Besta, and Torsten Hoefler. Benchmarking Filtered Approximate Nearest Neighbor Search Algorithms on Transformer-based Embedding Vectors, April 2026. URL http://arxiv.org/abs/2507.21989. arXiv:2507.21989 [cs].

[15] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC '98*, pages 604–613, Dallas, Texas, United States, 1998. ACM Press. ISBN 978-0-89791-962-3. doi: 10.1145/276698.276876. URL http://portal.acm.org/citation.cfm?doid=276698.276876.

[16] Omid Jafari and Parth Nagarkar. Experimental Analysis of Locality Sensitive Hashing Techniques for High-Dimensional Approximate Nearest Neighbor Searches. volume 12610, pages 62–73. 2021. doi: 10.1007/978-3-030-69377-0_6. URL http://arxiv.org/abs/2006.11285. arXiv:2006.11285 [cs].

[17] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://papers.neurips.cc/paper_files/paper/2019/hash/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Abstract.html.

[18] H Jégou, M Douze, and C Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, January 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.57. URL http://ieeexplore.ieee.org/document/5432202/.

[19] Mingi Kang and Jeová Farias Sales Rocha Neto. Attention Via Convolutional Nearest Neighbors, November 2025. URL http://arxiv.org/abs/2511.14137. arXiv:2511.14137 [cs].

[20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models, January 2020. URL http://arxiv.org/abs/2001.08361. arXiv:2001.08361 [cs].

[21] Jingwen Liu, Hantao Yu, Clayton Sanford, Alexandr Andoni, and Daniel Hsu. Fast attention mechanisms: a tale of parallelism, September 2025. URL http://arxiv.org/abs/2509.09001. arXiv:2509.09001 [cs].

[22] Yu A. Malkov and D. A. Yashunin. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, April 2020. ISSN 0162-8828, 2160-9292,

1939-3539. doi: 10.1109/TPAMI.2018.2889473. URL https://ieeexplore.ieee.org/document/8594636/.

[23] Leland McInnes. lmcinnes/pynndescent, April 2026. URL https://github.com/lmcinnes/pynndescent. original-date: 2018-02-07T23:23:54Z.

[24] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL http://aclweb.org/anthology/D14-1162.

[25] Chanop Silpa-Anan and Richard Hartley. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, Anchorage, AK, USA, June 2008. IEEE. ISBN 978-1-4244-2242-5. doi: 10.1109/CVPR.2008.4587638. URL http://ieeexplore.ieee.org/document/4587638/.

[26] Philip Sun, Felix Chern, Yaroslav Akhremtsev, Ruiqi Guo, David Simcha, and Sanjiv Kumar. Scaling Laws for Nearest Neighbor Search. *Proceedings of the 1st Workshop on Vector Databases at International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=nXAlM7xci6.

[27] Wentao Xiao, Yueyang Zhan, Rui Xi, Mengshu Hou, and Jianming Liao. Enhancing HNSW Index for Real-Time Updates: Addressing Unreachable Points and Performance Degradation, July 2024. URL http://arxiv.org/abs/2407.07871. arXiv:2407.07871 [cs].

Appendix A. Related work

Approximate nearest neighbor (ANN) search retrieves high-quality candidate sets approximating true nearest neighbors, trading exactness for speed [15, 17]. Indexing strategies include tree-based, hash-based, quantization-based, and graph-based methods [27]. Among these, hashing and quantization are often grouped under a broader partitioning-based framework, as both divide the search space into discrete regions to prune candidates [26]. We adopt this framing in our work.

Grid-based methods, a variant of partitioning-based methods, were explored in early theoretical studies on ANN. For example, translated grids were used to mitigate exponential dependence on dimensionality and remove logarithmic factors in batched query settings [8]. Subsequent work, such as the bucketing method, achieved constant-time hash evaluations independent of dataset size [15]. These approaches are notable for their conceptual simplicity and minimal indexing overhead. As ANN research shifted toward large-scale, high-dimensional datasets, practical focus moved toward more flexible methods, including Locality-Sensitive Hashing (LSH) [11, 12, 16], k-d trees [4, 7, 25], and Production Quantization [18]. As a result, grid-based methods became relatively underexplored in modern ANN systems, despite offering a distinct set of structural and theoretical properties.

In addition to partitioning-based methods, modern ANN systems frequently rely on graph-based indices, particularly Hierarchical Navigable Small World (HNSW), which achieve strong empirical performance by traversing multi-layer proximity graphs [22]. However, graph-based methods introduce distinct operational challenges. For example, they can exhibit capacity-limited failure, where insufficient exploration leads to abrupt degradation in neighbor quality [9]. They are also sensitive to dynamic updates, as repeated insertions and deletions can produce unreachable nodes and degrade recall over time [27]. More broadly, these limitations highlight the complex trade-offs between query efficiency, index construction, and robustness that characterize modern ANN methods.

Beyond algorithmic design, recent work has emphasized empirical evaluation and scaling behavior as the primary lens for understanding ANN performance. Standardized benchmarks such as `ann-benchmarks` evaluate methods across datasets, reporting recall–latency tradeoffs under fixed resource budgets [5, 6]. Benchmarking studies have helped establish empirical baselines across a wide range of regimes, with HNSW and Facebook AI Similarity Search (FAISS) often exhibiting state-of-the-art performance. To better generalize benchmarks, studies also seek to evaluate how performance scales with dataset size N and search budget. For example, Sun et al. established power-law scaling relationships for partitioning- and graph-based methods [26]. The study proposed a framework for the total cost J of ANN search that is a function of query-time compute, indexing, and storage costs: $J(N) = f_I \cdot I(N) + f_C \cdot C(N) + S(N)$, where I, C, and S denote indexing, compute, and storage costs, and f_I, f_C their respective frequency of occurrence. We adapt this cost framework in Section 3.4.

While benchmarking and scaling studies have significantly improved our understanding of ANN performance, they primarily focus on graph-based, tree-based, and adaptive partitioning methods. Grid-based approaches have remained comparatively underexplored despite their unique properties in the ANN design space, such as structural simplicity, low indexing cost, and predictable query behavior. We anticipated that grid-based methods may therefore exhibit different performance trade-offs from the other, more characterized methods - particularly in regimes where indexing cost, rebuild frequency, or robustness to dimensionality are dominant concerns. Thus, we revisit grid-based ANN methods in this work, aiming to expand understanding of how different ANN design principles translate into distinct scaling behaviors.

Appendix B. Derivation of the Multiprobe Grid Scaling Model

We derive a probabilistic upper-bound approximation for the query cost–recall relationship of multiprobe grid search, averaged over many queries. The derivation assumes uniform query and data distribution at the cell scale and does not account for rank distortion introduced by the PCA projection; extending to PCA-induced distortion is left for future work. Throughout, we use the same notation as the main text: N data points in $\mathbb{R}^d$, projected via PCA to $\mathbb{R}^m$ and partitioned into a uniform G^m grid with cell side length C.

B.1. Probe Setup and Cell Offsets

Given a query $\boldsymbol{q} \in \mathbb{R}^d$ and its PCA projection $\boldsymbol{q}' = (q'_1, \ldots, q'_m)$, let c_h denote the home cell. For each dimension j, define the binary offset

$$o_j \;=\; \begin{cases} +1 & q'_j \geq \text{midpoint of } c_h \text{ in dim. } j, \\ -1 & \text{otherwise.} \end{cases}$$

Because the position of $\boldsymbol{q}'$ within c_h determines the sign per dimension, the set of candidate cell displacements is the Cartesian product of the m binary choices, giving 2^m offset vectors. Removing the all-zero vector (the home cell) leaves $2^m - 1$ neighboring cells, rather than the $3^m - 1$ full-neighborhood count. This orthant pruning is employed to prune the number of cells checked as m increases.

Two-dimensional example. For $m = 2$ with q' above the midpoint in both dimensions ($o = (+1, +1)$), the non-home offset vectors are $\{(0, 1), (1, 0), (1, 1)\}$ in Cartesian choice notation, corresponding to the horizontal, vertical, and diagonal neighbors in the upper-right orthant.

B.2. Wall distance ordering and cost model

For each candidate cell, define the squared wall distance

$$w_i^2 \;=\; \sum_{j\,:\,o_j \neq 0} (q'_j - \mathrm{wall}_j)^2, \tag{5}$$

where wall_j is the boundary of c_h shared with the candidate cell in dimension j. The $2^m - 1$ values are sorted ascending as $(w_0^2, \ldots, w_{2^m-2}^2)$, and at query time only the first $n_{\mathrm{probe}} \leq 2^m - 1$ cells in this order are probed. Candidates from all probed cells are re-ranked in the native d-space.

Each probed cell contributes on average N/G^m points to the candidate set, and ranking is linear in the candidate count. Thus

$$\mathrm{cost} \;=\; 1/\mathrm{QPS} \;=\; K \cdot n_{\mathrm{probe}} \cdot \frac{N}{G^m}, \tag{6}$$

where $K > 0$ absorbs per-candidate constants (distance computations, comparisons, bookkeeping).

B.3. Recall as a sum of cell-membership probabilities

Let x^* denote the true nearest neighbor of q and define the cell-membership probability $P_i = \Pr[x^* \in c_i]$. Because probed cells are disjoint,

$$R(n_{\mathrm{probe}}) \;=\; \sum_{i=0}^{n_{\mathrm{probe}}-1} P_i.$$

Under uniform query/data distribution at the cell scale, P_i decreases monotonically with w_i^2; since the w_i^2 are sorted ascending, $P_0 \geq P_1 \geq \cdots \geq P_{n_{\text{probe}}}$. We adopt the exponential fall-off

$$P_i \;\approx\; P_h\, e^{-\mu w_i^2}, \qquad P_h := P_0,\ \mu > 0, \tag{7}$$

as a mean-field approximation.

B.4. Expected wall distance via offset geometry

To obtain an analytical form, we replace the cell-specific w_i^2 with its expected value under uniform queries. From equation 5, $w_i^2 = \sum_{j:o_j \neq 0} \varepsilon_j^2$ where $\varepsilon_j := q'_j - \text{wall}_j$. Under the uniform-distribution assumption, $\varepsilon_j \sim U(0, C/2)$ (the query sits in a given half of c_h per dimension), giving

$$\mathbb{E}[\varepsilon_j^2] \;=\; \int_0^{C/2} x^2 \cdot \frac{2}{C}\, dx \;=\; \frac{(C/2)^2}{3} \;=:\; L.$$

Let A_i denote the set of nonzero offset coordinates of the i-th cell. By linearity,

$$\mathbb{E}[w_i^2] \;=\; |A_i| \cdot L. \tag{8}$$

For each $j \in \{1, \ldots, m\}$, there are $\binom{m}{j}$ offset vectors with exactly j nonzero entries, all sharing $\mathbb{E}[w_i^2] = jL$. Sorting the $2^m - 1$ cells in ascending order of expected wall distance produces a step function that takes value jL on a plateau of length $\binom{m}{j}$, for $j = 1, \ldots, m$.

Example: $m = 3$**.** There are $\binom{3}{1} = 3$ face-adjacent cells, reachable by the offset vectors (0,0,1), (0,1,0) and (1,0,0) with $\mathbb{E}[w^2] = L$, $\binom{3}{2} = 3$ edge-adjacent cells with offset vectors (0,1,1), (1,0,1) and, (1,1,0) with $\mathbb{E}[w^2] = 2L$, and $\binom{3}{3} = 1$ corner cell with offset vector (1,1,1) with $\mathbb{E}[w^2] = 3L$, totaling $2^3 - 1 = 7$.

Arithmetic mean gap (telescoping). Define $\mathbb{E}[w_{-1}^2] := 0$ for the home cell. The arithmetic mean of first differences is

$$\theta \;=\; \frac{1}{2^m - 1} \sum_{i=0}^{2^m - 2} \big(\mathbb{E}[w_i^2] - \mathbb{E}[w_{i-1}^2]\big).$$

The sum telescopes: within each plateau the increment is 0, and at each of the m transitions between consecutive plateaus the increment is L. Therefore

$$\theta \;=\; \frac{m \cdot L}{2^m - 1} \;=\; \frac{m\,(C/2)^2}{3\,(2^m - 1)}. \tag{9}$$

For $m = 3$, $\theta = 3L/7 = (C/2)^2/7$. We approximate the step function linearly as

$$\mathbb{E}[w_i^2] \;\approx\; \theta\, i. \tag{10}$$

B.5. Geometric closed form

Substituting $\mathbb{E}[w_i^2] \approx \theta i$ into equation 7 with $\Delta := e^{-\mu\theta} \in (0,1)$, we obtain $P_i \approx P_h \Delta^i$, and the recall becomes a geometric series:

$$R(n_{\text{probe}}) \;=\; P_h \sum_{i=0}^{n_{\text{probe}}-1} \Delta^i \;=\; P_h \cdot \frac{1-\Delta^{n_{\text{probe}}}}{1-\Delta}. \tag{11}$$

The achievable recall range is bounded by

$$R_{\min} \;=\; P_h \quad (n_{\text{probe}} = 1), \qquad R_{\max} \;=\; P_h \cdot \frac{1-\Delta^{2^m}}{1-\Delta} \quad (n_{\text{probe}} = 2^m).$$

For $\Delta \in (0,1)$ and moderately large m, $\Delta^{2^m} \to 0$ very rapidly, so

$$R_{\max} \;\approx\; \frac{P_h}{1-\Delta} \qquad \Longleftrightarrow \qquad \frac{1}{R_{\max}} \;\approx\; \frac{1-\Delta}{P_h}. \tag{12}$$

B.6. Log-QPS derivation: from geometric closed form to exponential

Starting from the geometric closed form equation 11 and solving for n_{probe},

$$\Delta^{n_{\text{probe}}} \;=\; 1 - \frac{R(1-\Delta)}{P_h}, \qquad n_{\text{probe}}(R) \;=\; \frac{\ln(1 - R/R_{\max})}{\ln \Delta}.$$

Using the approximation $(1-\Delta)/P_h \approx 1/R_{\max}$ from equation 12,

$$n_{\text{probe}}(R) \;=\; \frac{\ln(1 - R/R_{\max})}{\ln \Delta} - 1. \tag{13}$$

Substituting equation 13 into the cost formula equation 6 and factoring out $\ln \Delta$ from the numerator,

$$\frac{1}{\text{QPS}} \;=\; \frac{KN}{G^m} \cdot \frac{\ln(1 - R/R_{\max}) - \ln \Delta}{\ln \Delta}. \tag{14}$$

Both $\ln(1 - R/R_{\max})$ and $\ln \Delta$ are negative (since $R/R_{\max}, \Delta \in (0,1)$), so we may rewrite equation 14 with all signs made explicit:

$$\text{QPS}(R) \;=\; \frac{|\ln \Delta|}{KN/G^m} \cdot \frac{1}{-\ln(1 - R/R_{\max}) + \ln \Delta}. \tag{15}$$

Taking the log. To expose the exponential structure, we take the logarithm of both sides of equation 15:

$$\log \text{QPS}(R) \;=\; \log\!\left(\tfrac{|\ln \Delta| G^m}{KN}\right) \;-\; \log\!\Big(-\ln(1 - R/R_{\max}) + \ln \Delta\Big). \tag{16}$$

The first term is constant in R; all the R-dependence lives in the second term.

Taylor expansion of the inner log. Let $u := R/R_{\max} \in (0,1)$. The function $-\ln(1-u) = \ln\big(1/(1-u)\big)$ admits the convergent Taylor series

$$-\ln(1-u) \;=\; u + \tfrac{u^2}{2} + \tfrac{u^3}{3} + \cdots$$

on $(0,1)$. Because $u < 1$, the higher-order terms are dominated by the linear term, and we truncate at first order:

$$-\ln(1 - R/R_{\max}) \;\approx\; \frac{R}{R_{\max}}. \tag{17}$$

The first-order truncation in equation 17 has relative error $1 - u/[-\ln(1-u)]$, which is approximately 28% at $u = 0.5$, 50% at $u \approx 0.8$, and diverges as $u \to 1$. The closed form derived below is therefore a local approximation, accurate for u bounded away from 1. As we discuss in Appendix B.7, this is one of two approximations whose validity degrades in the high-recall regime; the empirical Pareto front in Figure 1 nonetheless follows the predicted log-linear shape over most of the operating range, suggesting that the structural prediction survives quantitative slack in the underlying assumptions. Substituting equation 17 into equation 16,

$$\log \mathrm{QPS}(R) \;\approx\; \log(\mathrm{const}) \;-\; \log\Big(R/R_{\max} + \ln \Delta\Big). \tag{18}$$

Mean-field approximation. We now make the mean-field argument rigorous. The derivation rests on the following assumption:

> **Assumption (mean-field).** The query points are uniformly distributed within the home cell, and the dataset points are drawn from a distribution that is approximately uniform at the scale of a single grid cell.

Three consequences follow. First, the wall distances w_i^2 are expectations taken over the uniform query position distribution within the cell, so θ is the mean inter-neighbor gap averaged over all possible query locations (Appendix B.4). Second, the decay parameter μ characterizes the rate at which the NN-membership probability falls off with wall distance, and is itself determined by the local point density at the cell scale. Lastly, since the point distribution is approximately uniform at the cell scale, the cell-membership probability mass $P_i = P_h e^{-\mu w_i^2}$ does not vary sharply between adjacent cells. That is, neighboring cells have similar density to the home cell, so the decay is gradual. This implies

$$\mu\theta \;\ll\; 1 \qquad\Longleftrightarrow\qquad \Delta = e^{-\mu\theta} \approx 1 \qquad\Longleftrightarrow\qquad |\ln \Delta| \;\ll\; 1. \tag{19}$$

This assumption is equivalent to requiring that the grid cell size be small relative to the scale at which the data distribution varies, which is the standard regime underlying any local approximation in partition-based ANN. It is also the regime in which the grid index is well-calibrated: if cells were so large that density varied significantly within them, the grid partition itself would be a poor indexing structure regardless of multiprobe.

Practical domain of the mean-field assumption. Real embedding distributions such as GloVe and SIFT exhibit substantial non-uniformity, with cluster structure, manifold geometry, and density variations corresponding to semantic structure. The assumption is therefore not literally satisfied by any of the datasets we evaluate. What the assumption requires *in practice* is that the optimizer

selects (m, G) such that adjacent cells have similar local density on average, a weaker condition than global uniformity, and one that is implicitly enforced, since configurations straddling severe density gradients incur recall penalties and are filtered from the Pareto front. The assumption nevertheless degrades as n_{probe} grows, because probing further from the home cell increases the likelihood of crossing a density gradient. High recall therefore corresponds to the regime where mean-field is most strained, by the same mechanism that makes the Taylor truncation in equation 17 least accurate. The two limitations are not independent, both reflecting the algorithm operating outside the local neighborhood of the home cell (where the mean-field description applies). The predicted log-linear form nevertheless holds empirically over most of the operating range in Figure 1, which we discuss further in Appendix B.7.

Dropping $\ln \Delta$. Under equation 19, we drop $\ln \Delta$ from the argument of the outer log in equation 18. To confirm this is a well-controlled approximation: because $\ln \Delta < 0$,

$$\frac{R}{R_{\max}} + \ln \Delta \;<\; \frac{R}{R_{\max}},$$

and monotonicity of $\log$ gives

$$\log\Big(R/R_{\max} + \ln \Delta\Big) \;<\; \log(R/R_{\max}),$$

so $\log(R/R_{\max})$ is a valid upper bound, with the gap shrinking to zero as $\mu\theta \to 0$. Applying this approximation to equation 18,

$$\log \mathrm{QPS}(R) \;\approx\; \log(\text{const}) \;-\; \log(R/R_{\max}). \tag{20}$$

Expanding the log-ratio. Expanding $\log(R/R_{\max}) = \log R - \log R_{\max}$ and substituting into equation 20,

$$\log \mathrm{QPS}(R) \;\approx\; \log(\text{const}) - \log R + \log R_{\max} \;=\; \log(\text{const} \cdot R_{\max}) - \log R. \tag{21}$$

Absorbing $\log R_{\max}$ into the constant, we define $\text{const}' := \text{const} \cdot R_{\max}$ and obtain

$$\log \mathrm{QPS}(R) \;\approx\; \log(\text{const}') - \log R. \tag{22}$$

Equation equation 22 is a power-law relationship, $\mathrm{QPS} \propto 1/R$.

Local linearization of $\log R$. To recover the exponential form observed empirically, we exploit the fact that R is confined to a bounded interval $[R_{\min}, R_{\max}]$ and linearize:

$$\log R \;\approx\; a + b\,R, \qquad R \in [R_{\min}, R_{\max}], \tag{23}$$

with $b > 0$ because $\log$ is increasing. Geometrically, this replaces the concave $\log R$ curve with a tangent line over the achievable recall range, as shown in Figure A1. Substituting equation 23 into equation 22,

$$\log \mathrm{QPS}(R) \;\approx\; \log(\text{const}') - a - b\,R. \tag{24}$$

Exponentiating. Since $\log = \log_{10}$ throughout, exponentiating equation 24 in base 10 yields

$$\mathrm{QPS}(R) \;\approx\; 10^{\log(\mathrm{const}')-a-bR} \;=\; \mathrm{const}' \cdot 10^{-a} \cdot 10^{-bR}. \tag{25}$$

Absorbing the two R-independent factors into a single constant $A' := \mathrm{const}' \cdot 10^{-a}$ and setting $B := b$, we obtain the log-linear throughput–recall relation:

$$\boxed{\mathrm{QPS}(R) \;\approx\; A' \cdot 10^{-BR}, \qquad A' > 0,\; B > 0.} \tag{26}$$

Equivalently, $\mathrm{QPS}(R) \approx A' e^{-B'R}$ with $B' = B \ln 10$, confirming the log-linear relationship observed empirically in Section 3.1. We note that the slope B is set by the local derivative of $\log R$ on $[R_{\min}, R_{\max}]$, which scales inversely with $R_{\max}$.

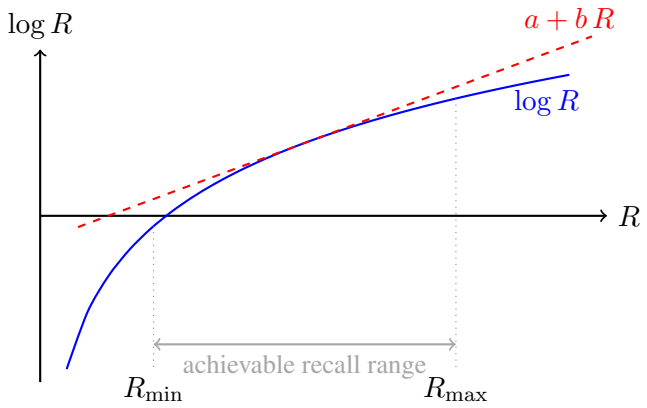

Figure A1: Local linearization of $\log R$ on the achievable recall interval $[R_{\min}, R_{\max}]$. The concave $\log R$ curve (blue, solid) is approximated by a tangent line $a + b\,R$ (red, dashed) over the narrow operating range of the algorithm. Substituting this linear approximation into $\log \mathrm{QPS} \approx \log(\mathrm{const} \cdot R_{\max}) - \log R$ converts the power-law dependence into an exponential one, yielding $\mathrm{QPS}(R) \approx A' e^{-B'R}$.

B.7. Validity, robustness, and empirical correspondence

The closed form equation 26 relies on multiple approximations: a Taylor truncation equation 17, a mean-field assumption equation 19 that real embedding distributions do not fully satisfy, and a local linearization of $\log R$ (equation 23). Taylor truncation and mean-field degrade in the high-recall regime, which is also the regime where the empirical d-scaling crossover reported in the main text emerges. Here, we argue that the qualitative log-linear form is structural and robust to its underlying assumptions, with Figure 1 corroborating our reasoning. In addition, the empirical d-scaling characterization in the main text is methodologically independent of the closed-form model.

Robustness of the log-linear form. It is not obvious *a priori* that equation 26 should describe the empirical Pareto front at all, given that the mean-field assumption is not fully satisfied by the non-uniform embedding distributions on which we evaluate the algorithm. We argue that the qualitative log-linear shape is structural rather than approximation-dependent. Two mechanical features of the algorithm produce the log-linear shape: (i) recall is a saturating, monotonically increasing function

of n_{probe}, because cells are probed in order of decreasing membership probability P_i; and (ii) cost is linear in n_{probe}, because each probed cell contributes a fixed expected candidate count N/G^m. Combining a saturating-concave recall function with a linear cost function yields cost growing faster than linearly in recall as recall approaches its ceiling, which on log-y vs. linear-recall axes traces out a near-straight line over the operating range with downward curvature near the ceiling. Importantly, neither (i) nor (ii) requires the mean-field assumption. (i) requires only that cells can be ordered by membership probability, which the algorithm enforces by construction, and (ii) follows from each probed cell contributing a fixed expected candidate count. Thus, the approximations affect the predicted slope B and ceiling $R_{\max}$, but not the functional form itself, which we observe on real, non-uniform embedding data.

Relevance to the d-scaling crossover. The closed form and the d-scaling characterization reported in Figure 2b answer fundamentally different questions. While equation 26 models the shape of the QPS–recall Pareto front for a fixed dataset, α_d characterizes how QPS (at fixed recall) scales across datasets of different dimensionality. The closed form contains no d-dependence in its functional form. Rather, d manifests empirically by influencing the Pareto-optimal (m, G, n_{probe}) configuration. Thus, the absence of explicit d-dependence in equation 26 underscores that the d-scaling crossover is an authentic, empirical finding that reflects the algorithm's ability to adapt to dataset dimensionality.

Appendix C. Implementation and Experimental Details

C.1. Multiprobe grid search

C.1.1. INDEX CONSTRUCTION

We employ PCA to project each point in $\mathbb{D}$ from $\mathbb{R}^d$ to $\mathbb{R}^m$, where the grid dimensionality $m \ll d$ is a hyperparameter. The projected space is partitioned into a uniform grid of G^m cells, where G is the number of splits per dimension. Each cell stores the indices of points whose projections fall within its boundaries. To handle empty cells encountered during querying, we precompute a multi-source breadth-first search (BFS) over the grid graph in which every occupied cell is seeded simultaneously as a source. The BFS expands outward through cell neighbors, recording for each cell the identity of its nearest occupied cell. This process is easy to parallelize and each non-empty cell only needs to be updated once. The result is a static lookup table mapping every grid cell to the nearest occupied cell by Manhattan distance. Insertion and deletion of new points are also efficient under this regime. Figure D12 shows examples of how often the BFS fallback is triggered as a function of dataset characteristics (i.e., N and d) and hyperparameters (G and n_{probe}).

C.1.2. QUERY PROCESSING

Given a query vector $\boldsymbol{q} \in \mathbb{R}^d$ and a probe count $n_{\text{probe}} \geq 1$, we project $\boldsymbol{q}$ to the m-dimensional PCA space and identify the cell containing the projected vector $\boldsymbol{q}'$. When $n_{\text{probe}} = 1$, only the primary cell is searched. When $n_{\text{probe}} > 1$, we additionally probe the $n_{\text{probe}} - 1$ nearest neighboring cells ranked by wall distance (the squared Euclidean distance from $\boldsymbol{q}'$ to the shared boundary of each neighboring cell). To bound the search space, we consider only the 2^m cells in the geometric orthant (the m-dimensional generalization of a quadrant) toward which $\boldsymbol{q}'$ is displaced from the primary cell's midpoint. Candidates from all probed cells are gathered and ranked using the full d-dimensional metric (cosine similarity for angular datasets, Euclidean distance for L_2 datasets) to return the top-k results. If all probed cells are empty, the precomputed BFS fallback is triggered to return the nearest occupied cell.

C.2. Experimental setup

All experiments use the `ann-benchmarks` framework [5, 6], which provides standardized evaluation with Docker containers enforcing single-CPU execution per algorithm. We report recall@$k = 10$ as the accuracy metric and queries per second (QPS) as the throughput metric.

C.2.1. BASELINES

We compare the multiprobe grid search with the following algorithms: Voyager (a HNSW variant, graph-based) [3] , PyNNDescent (graph-based) [23], Annoy (tree-based) [2], and FAISS-IVF (quantization-based partitioning) [1]. We selected these state-of-the-art algorithms to represent the major ANN algorithm families. Brute-force search serves as a reference for several performance metrics (e.g., queries per second) and to identify ground truth nearest neighbors.

C.2.2. HYPERPARAMETER TUNING FOR MULTIPROBE GRID

The multiprobe grid algorithm has three hyperparameters: grid dimensionality m, grid splits G, and probe count n_{probe}. The pair (m, G) determines the index structure, while n_{probe} impacts the recall-

latency tradeoff at query time. We run NSGA-II multi-objective optimization (200 trials) using the Optuna package to identify Pareto-optimal (m, G) pairs for the GloVe-200-angular dataset; the same configurations are applied to subsampled GloVe-200 and to GloVe-25, -50, and -100-angular. A separate optimization was performed on SIFT-128-euclidean because of the distinct data modality. From each Optuna study, representative (m, G) pairs spanning recall >0.10 to 0.98 are selected for all subsequent experiments. For each pair, n_{probe} is swept across a range of values up to 2^m to trace the recall-QPS frontier. Baseline algorithms use their respective `ann-benchmarks` default parameter sweeps.

C.2.3. SCALING ANALYSIS

We characterize scaling behavior by fitting $\log(\text{QPS}) = \alpha \log(x) + b$, where x corresponds to either dataset size (N) or dataset dimensionality (d). The exponent α_x captures how query throughput changes with x. For each algorithm, dataset, and recall target, we construct the Pareto front over all benchmarked configurations and interpolate QPS to the target recall by linear interpolation in $\log_{10}(\text{QPS})$ between the two Pareto-optimal configurations whose measured recalls bracket the target. If the target recall falls outside the Pareto front's recall range for a given dataset, that point is omitted from the fit.

- For N-scaling studies, GloVe-200-angular is sampled at $N \in \{10^4, 2.0\times10^4, 4.0\times10^4, 7.5\times 10^4, 1.5\times10^5, 3.0\times10^5, 6.0\times10^5, 1.18\times10^6\}$ points. SIFT-128-euclidean is sampled at $N \in \{10^4, 5\times10^4, 10^5, 5\times10^5, 10^6\}$ points. Ground truth ($k = 10$ nearest neighbors) is recomputed for each subsample via brute-force.
- d-scaling uses the GloVe word embedding family [24] (GloVe-25, 50, 100, 200), which provides four datasets of identical size ($N = 1.18\times10^6$) and source distribution with varying native dimensionality $d \in \{25, 50, 100, 200\}$.

C.3. Profiling analysis of multiprobe grid

Our multiprobe grid implementation is in Python, while baseline algorithms (Voyager, PyNNDescent, Annoy, FAISS-IVF) use highly optimized C++. Here, we investigate whether Python interpreter overhead, memory allocation, and orchestration loops have significant impact on our reported scaling exponents.

First-principles reasoning. The per-query cost primarily decomposes into two components: an algorithmic component involving calls into NumPy/BLAS C kernels (e.g., PCA projection, candidate gathering via concatenation and fancy indexing, re-ranking), and an overhead component involving interpreter-bound orchestration (e.g., primary cell lookup, neighbor cell enumeration, candidate-list assembly). The interpreter-bound operations operate on a fixed number of cells (at most 2^m) and are therefore independent of N, contributing a constant, additive term to the per-query cost. In the regime where the algorithmic term dominates, which we expect for any modern dataset of appreciable N, this additive overhead becomes negligible and does not affect the slope of a log–log fit. The reported scaling exponents therefore reflect algorithmic complexity, not implementation language.

Empirical profiling. We profiled the multiprobe query path with `cProfile` on GloVe-200-angular across three N scales spanning two orders of magnitude, at a representative configuration (m=6, G=5, n_{probe}=16, 1,000 queries per N). The query was decomposed into named phase functions so `cProfile`'s per-function attribution gives per-phase timings (Table C1). Phases dominated by NumPy/BLAS calls (PCA projection, candidate gathering via `np.concatenate` and fancy indexing, and re-ranking via norm/matmul/argpartition) scale near-linearly with candidate count. In contrast, phases that are pure Python orchestration (cell index lookup, neighbor enumeration, list assembly) remain at $O(1\mu s)$ cost across the entire range of N. Altogether, we observe a factor of ~ 2.6 increase from N=10^4 to N=1.18×10^6 in the Python overhead and a factor of $\sim 865\times$ growth of the NumPy/BLAS-backed phases. At N=1.18×10^6, Python orchestration accounts for under 0.01% of total query time. The reported scaling exponents in the main text should therefore hold whether multiprobe grid is implemented in Python or C++.

Table C1: Per-query phase times (µs) for multiprobe grid on GloVe-200-angular (m=6, G=5, n_{probe}=16), profiled with `cProfile` over 1,000 queries per N.

Phase	Bucket	N=10^4	N=1.5×10^5	N=1.18×10^6
PCA projection	NumPy/BLAS	1.4	4.8	8.9
Cell index lookup	Python	0.3	0.8	1.1
Neighbor enumeration	Python	0.9	1.5	2.3
Assemble candidate indices	Python	0.2	0.4	0.6
Gather candidate vectors	NumPy/BLAS	64.9	5,354.3	62,334.9
Re-rank (norm/matmul/argpartition)	NumPy/BLAS	9.4	311.9	3,125.7
NumPy/BLAS subtotal		**75.7**	**5,670.9**	**65,469.6**
Python orchestration subtotal		**1.5**	**2.7**	**3.9**
Total (sum of phases)		77.2	5,673.6	65,473.5

C.4. Derivation of the rebuild-to-query crossover

Section 3.4 reports a rebuild-to-query crossover threshold for GloVe-200 of approximately one rebuild per 2,600–20,400 queries depending on the baseline. Here, we derive that range from the cost-framework from Sun et al [26].

Crossover condition. The total cost is

$$J(N) = f_I \cdot I(N) + f_C \cdot C(N) + M(N), \tag{27}$$

where f_I and f_C denote the frequencies of index rebuild and query, respectively. Multiprobe grid achieves lower total cost than a given baseline when $J_{\text{grid}} < J_{\text{baseline}}$. Because the memory term $M(N)$ enters identically on both sides, it cancels and the inequality reduces to

$$f_I \cdot I_{\text{grid}} + f_C \cdot C_{\text{grid}} \;<\; f_I \cdot I_{\text{baseline}} + f_C \cdot C_{\text{baseline}}. \tag{28}$$

Collecting the f_I terms on the right and the f_C terms on the left,

$$f_C \cdot \big(C_{\text{grid}} - C_{\text{baseline}}\big) \;<\; f_I \cdot \big(I_{\text{baseline}} - I_{\text{grid}}\big). \tag{29}$$

Multiprobe has lower indexing cost ($I_{\text{grid}} < I_{\text{baseline}}$) and higher per-query latency ($C_{\text{grid}} > C_{\text{baseline}}$), so we define $\Delta I := I_{\text{baseline}} - I_{\text{grid}} > 0$ and $\Delta C := C_{\text{grid}} - C_{\text{baseline}} > 0$. Both sides of the inequality are positive. Dividing both sides by $f_I \cdot \Delta C > 0$ (which preserves the inequality direction),

$$\frac{f_C}{f_I} < \frac{\Delta I}{\Delta C} =: N_{\text{cross}}. \tag{30}$$

The ratio f_C/f_I is the expected number of queries per index rebuild; multiprobe wins on total cost whenever this ratio falls below the crossover N_{cross}.

Empirical crossovers for GloVe-200 ($N = 1.18 \times 10^6$). For each algorithm we use the Pareto-optimal configuration nearest to recall@k=10 =0.80. Multiprobe's anchor is (m=6, G=5). Per-algorithm values and the resulting crossovers are reported in Table C2; ΔI and ΔC are converted to a common time unit (seconds) before computing the ratio. The 2,600–20,400 range cited in Section 3.4 is set by the FAISS-IVF crossover (2,591) and the Voyager crossover (20,432), the two extremes among the four baselines.

Table C2: Rebuild-to-query crossover threshold N_{cross} for multiprobe grid against each baseline at $N = 1.18 \times 10^6$, recall@k=10 =0.80. Multiprobe achieves lower total cost when $f_C/f_I < N_{\text{cross}}$ (i.e., when index rebuilds happen at least once per N_{cross} queries).

Algorithm	I (s)	C (ms)	ΔI (s)	ΔC (ms)	N_{cross}
Multiprobe Grid	**8.4**	**76.98**	—	—	—
FAISS-IVF	206	0.72	197.6	76.26	2,591
Annoy	333	7.20	324.6	69.78	4,652
PyNNDescent	500	0.56	491.6	76.42	6,431
Voyager (HNSW)	1,569	0.62	1560.6	76.36	20,432

Appendix D. Supplementary Figures

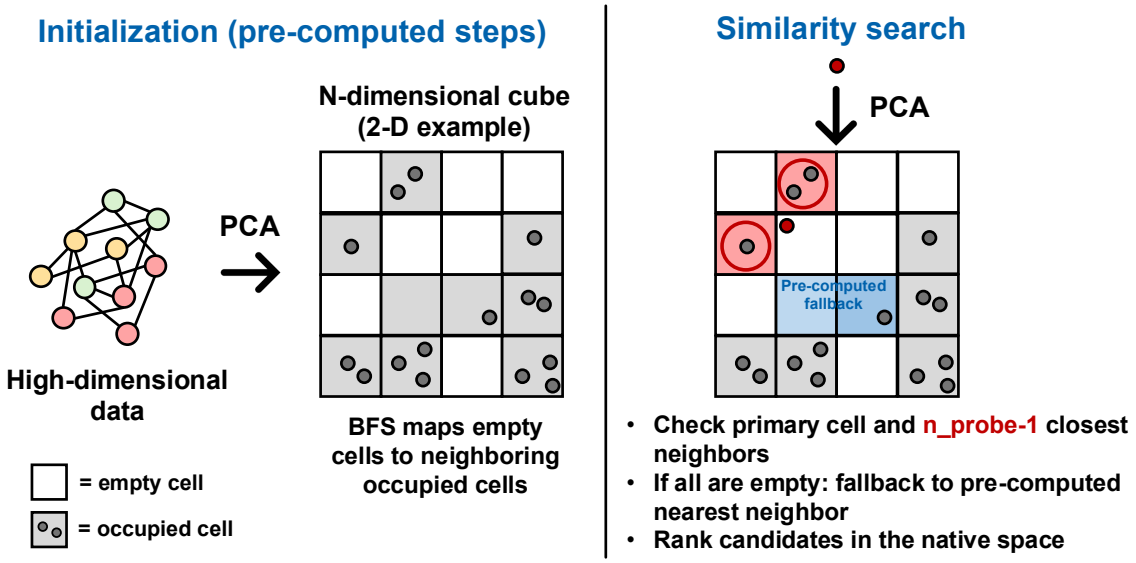

Figure D1: Overview of the multiprobe grid algorithm. **Initialization:** dataset points are projected from $\mathbb{R}^d$ to $\mathbb{R}^m$ ($m \ll d$) via principal component analysis and assigned to cells in a uniform G^m grid. Empty cells store a pointer to the nearest occupied cell, as computed by multisource breadth-first-search. **Similarity search:** the query $\mathbf{q}$ is projected to $\mathbf{q}'$, which identifies a primary cell. For $n_{\text{probe}} > 1$, additional cells in the orthant toward which $\mathbf{q}'$ is displaced are probed in order of wall distance. Candidates from all probed cells are gathered and re-ranked in the native d-dimensional space to return the top-k neighbors.

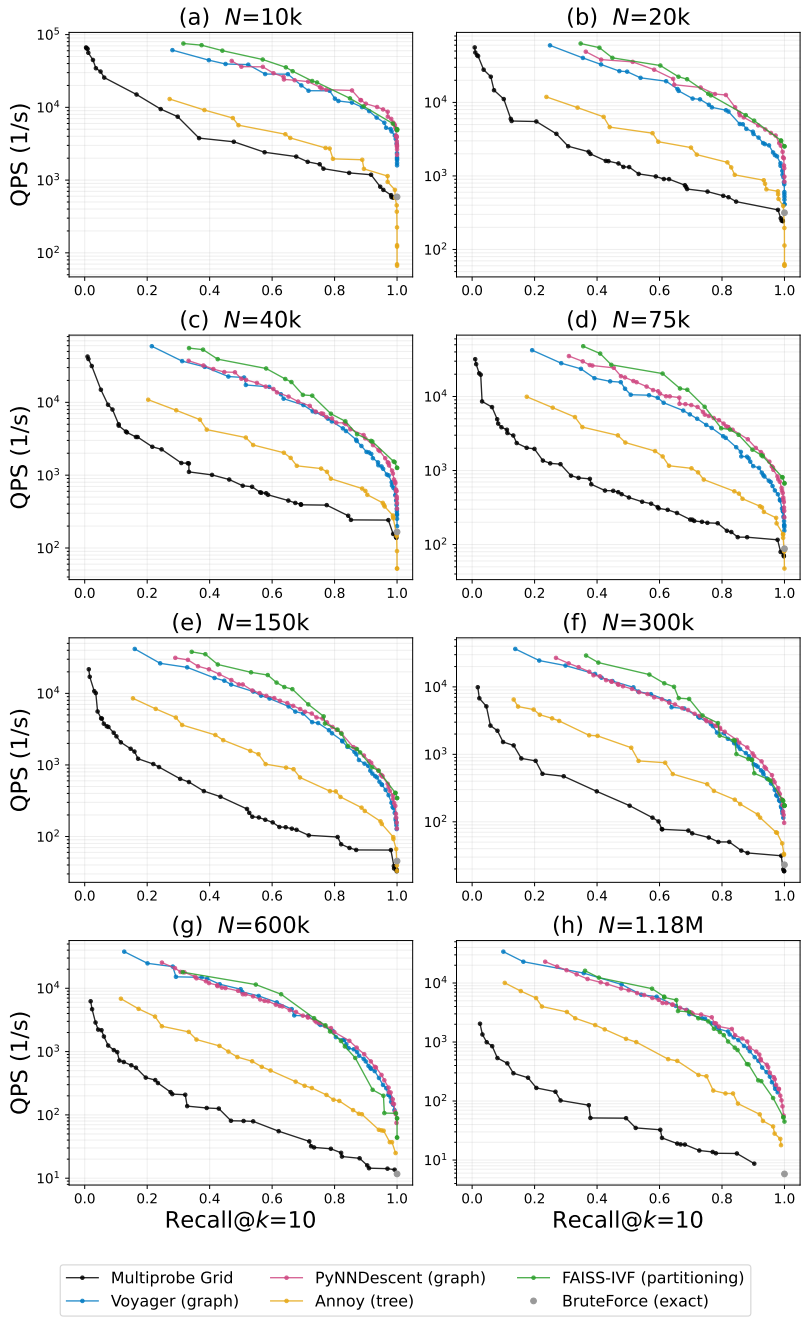

Figure D2: Pareto fronts across subsampled N on GloVe-200-angular ($d = 200$), where each panel shows all five algorithms + brute-force (which appears as a single point at recall=1.0). Note that panel (h) corresponds to Figure 1 in the main text.

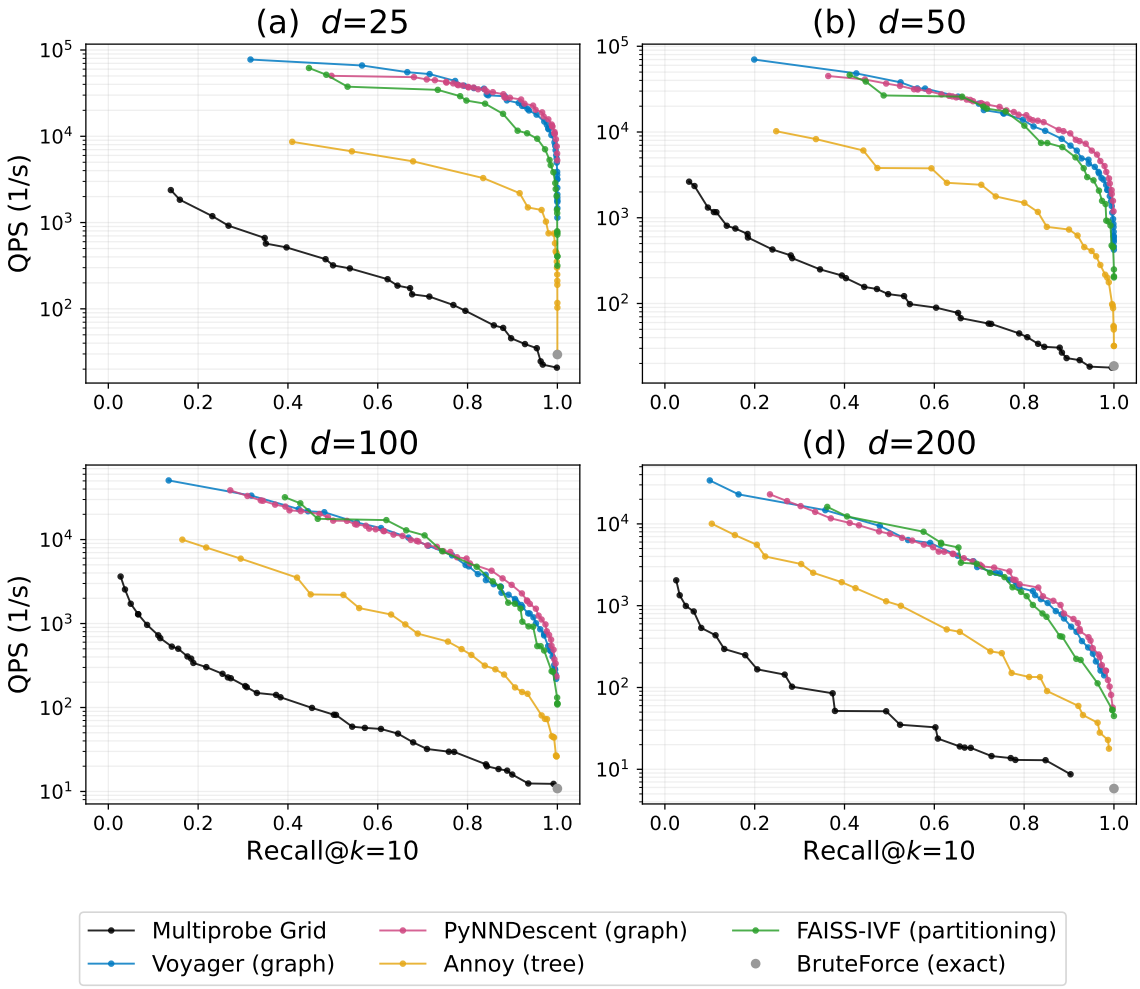

Figure D3: Pareto fronts across the GloVe-25-, 50-, 100-, and 200-angular datasets ($N = 1.18 \times 10^6$), where each panel shows all five algorithms + brute-force (which appears as a single point at recall=1.0). Note that panel (d) corresponds to panel (h) in Figure D2 and Figure 1 in the main text.

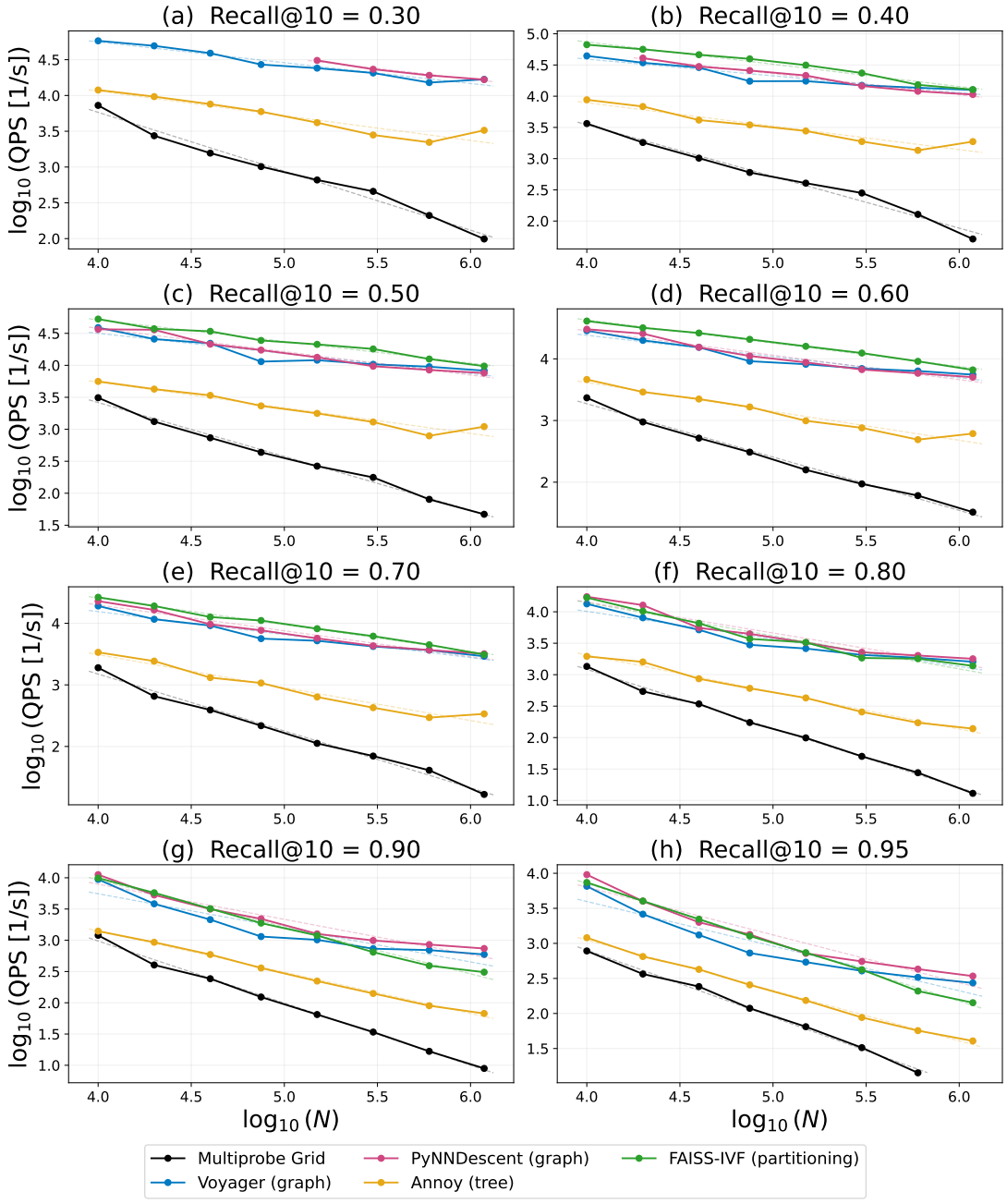

Figure D4: $\log_{10}(\text{QPS})$ vs. $\log(N)$ plots at each target recall@k=10 used to derive the exponents in Figure 2a. Datasets: GloVe-200-angular ($d = 200$) subsampled at $N = 10^4$, 2.0×10^4, 4.0×10^4, 7.5×10^4, 1.5×10^5, 3.0×10^5, 6.0×10^5, and the full 1.18×10^6 points. The slope of each curve gives an α_N value for the respective algorithm and target recall (one data point in Figure 2a). Note that QPS values are interpolated along the Pareto front for each N and target recall. Points are omitted where no configuration in the benchmark sweep achieved the target recall.

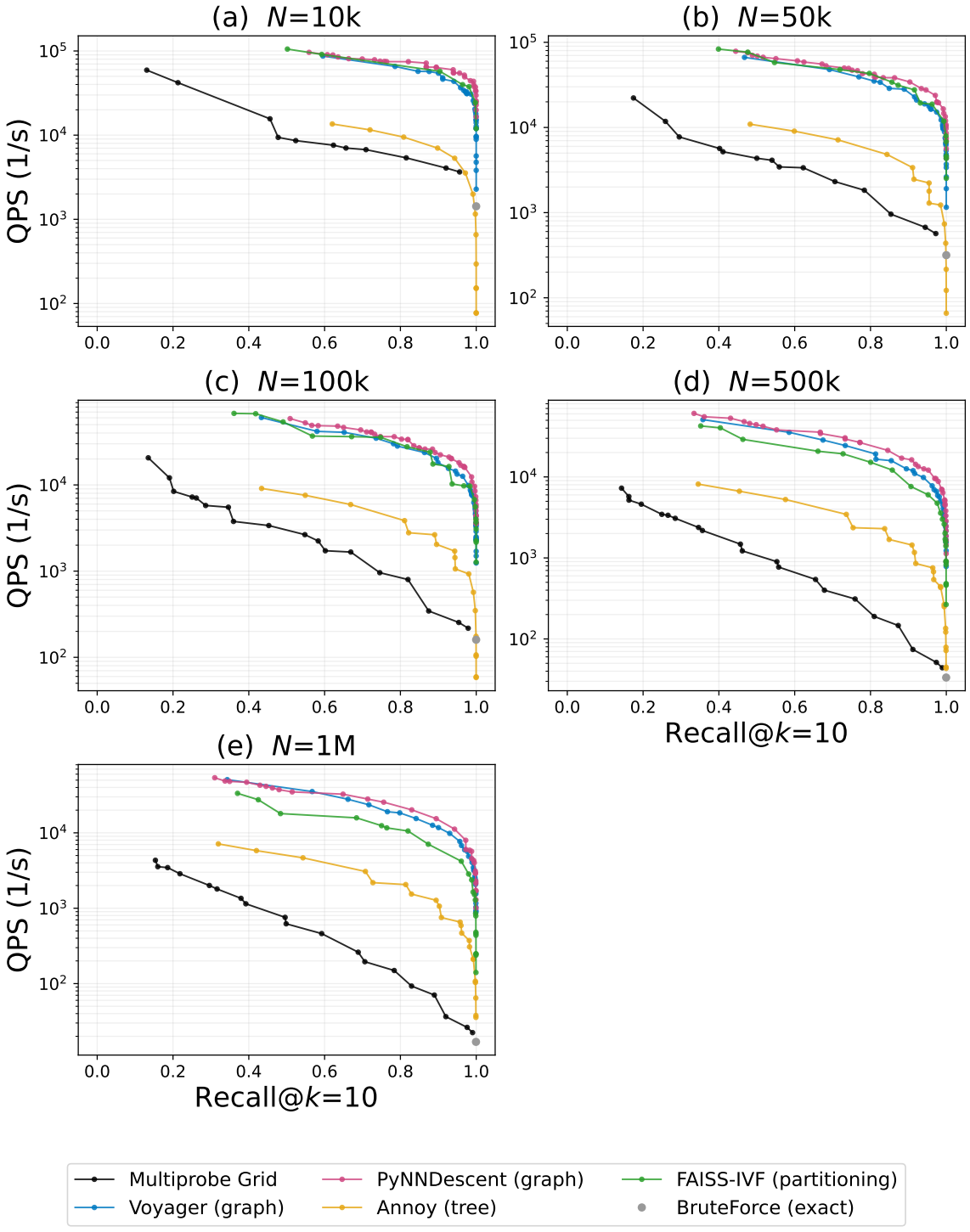

Figure D5: Pareto fronts across subsampled N on SIFT-128-euclidean ($d = 128$), where each panel shows all five algorithms + brute-force (which appears as a single point at recall=1.0).

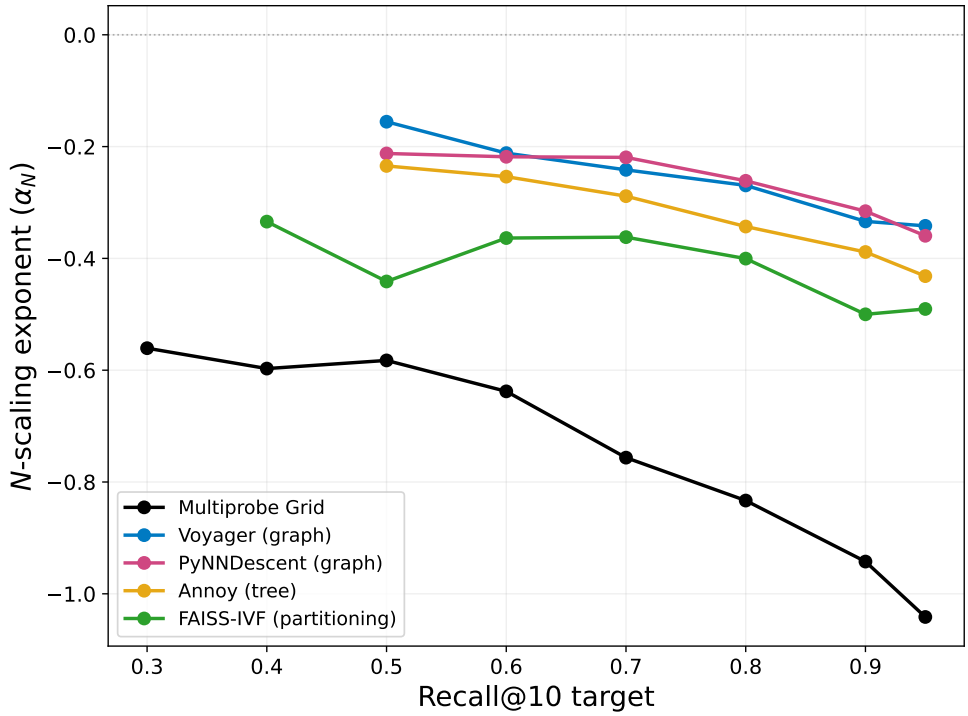

Figure D6: N-scaling exponent α_N vs. recall@k=10 target on SIFT-128-euclidean. Datasets: SIFT-128-euclidean subsampled at $N \in \{10^4, 5 \times 10^4, 10^5, 5 \times 10^5, 10^6\}$.

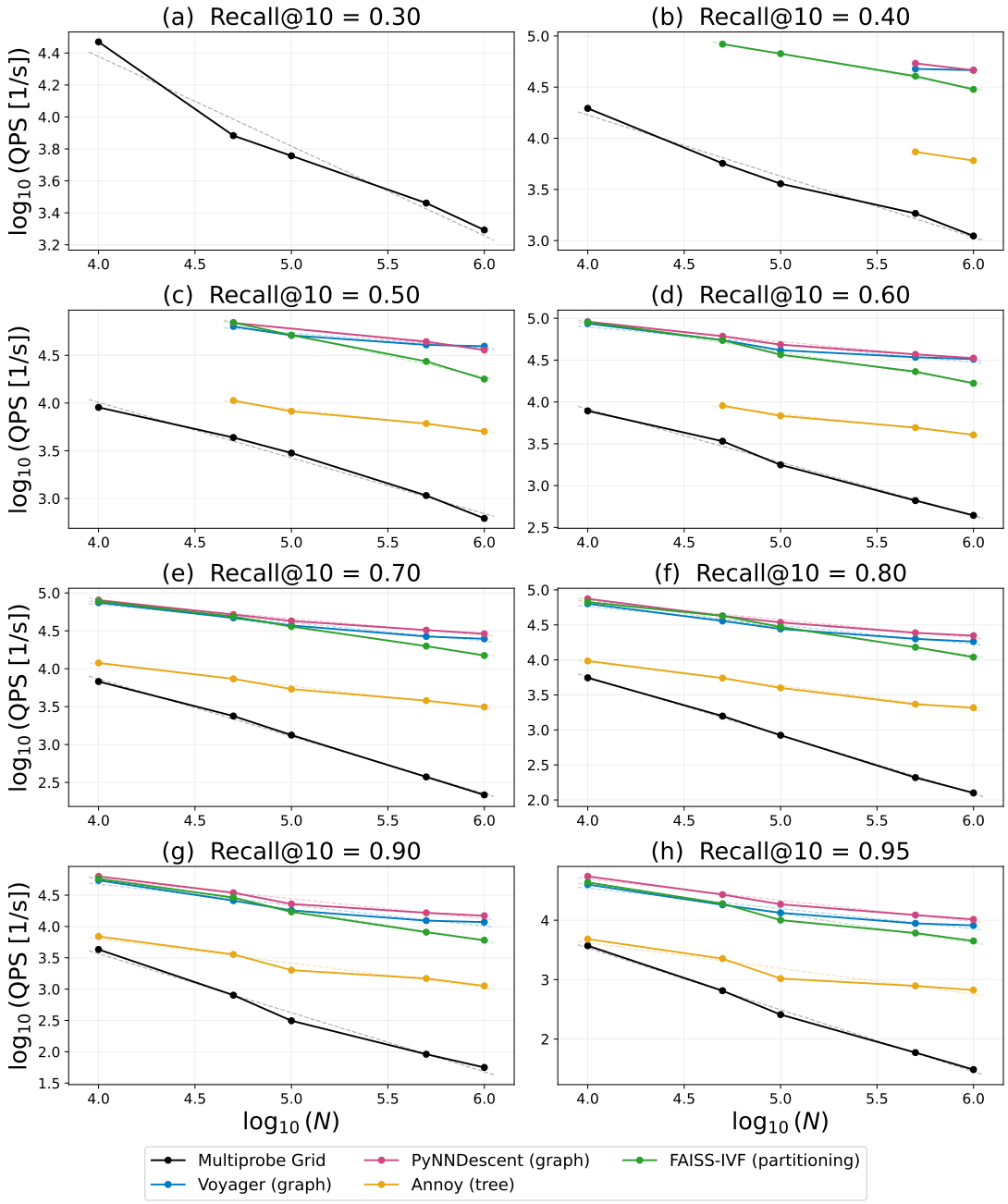

Figure D7: $\log_{10}(\text{QPS})$ vs. $\log_{10}(N)$ plots at each target recall@k=10. Datasets: SIFT-128-euclidean ($d = 128$) subsampled at $N \in \{10^4, 5 \times 10^4, 10^5, 5 \times 10^5, 10^6\}$. The slope of each curve gives an α_N value for the respective algorithm and target recall (one data point in Figure D6). Note that QPS values are interpolated along the Pareto front for each N and target recall. Points are omitted where no configuration in the benchmark sweep achieved the target recall

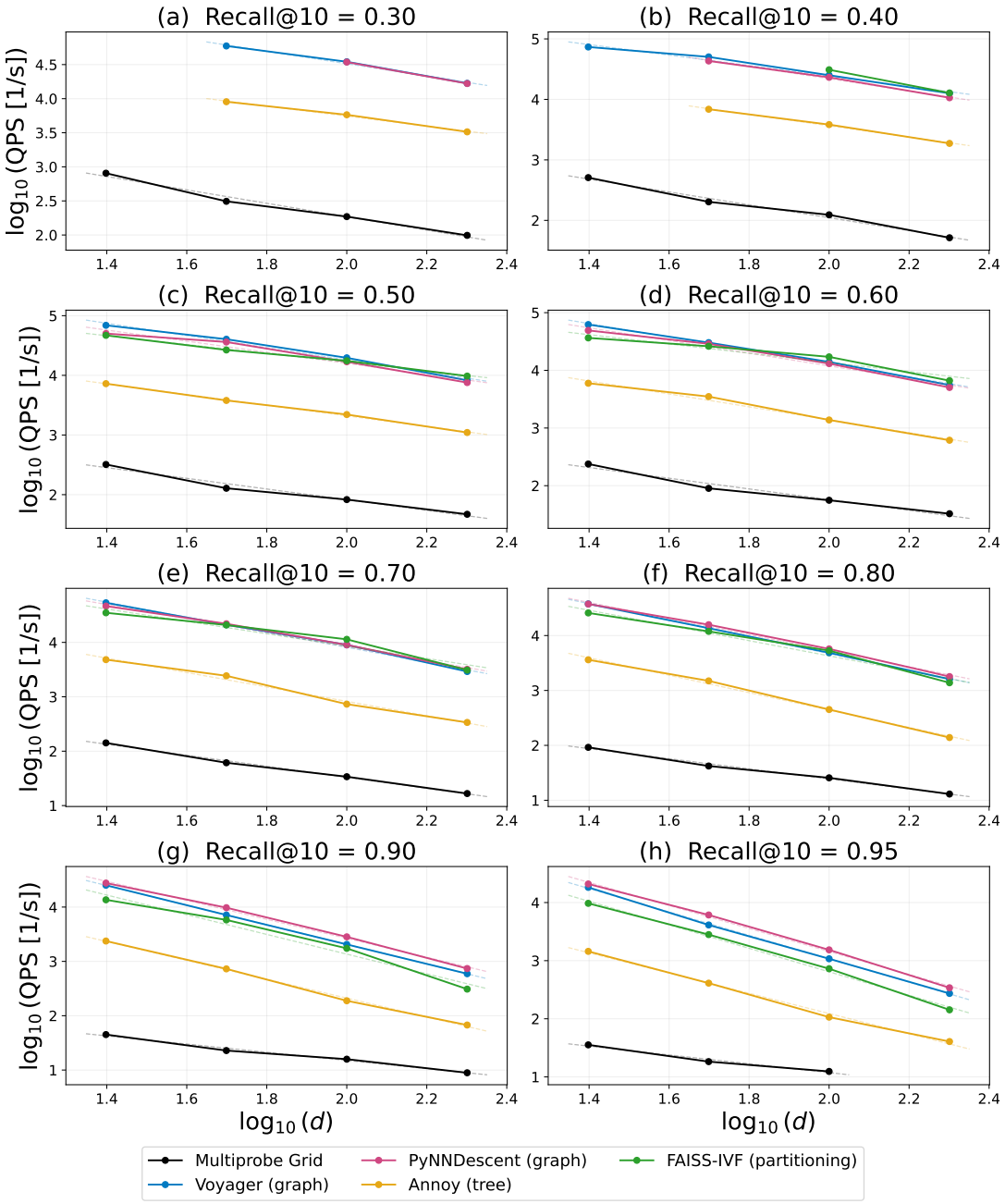

Figure D8: $\log_{10}(\text{QPS})$ vs. $\log(d)$ plots at each target recall@k=10 used to derive the exponents in Figure 2b. Datasets: GloVe-25, -50, -100, and -200-angular ($N = 1.18 \times 10^6$). The slope of each curve gives an α_d value for the respective algorithm and target recall (one data point in Figure 2b). Note that QPS values are interpolated along the Pareto front for each d and target recall. Points are omitted where no configuration in the benchmark sweep achieved the target recall.

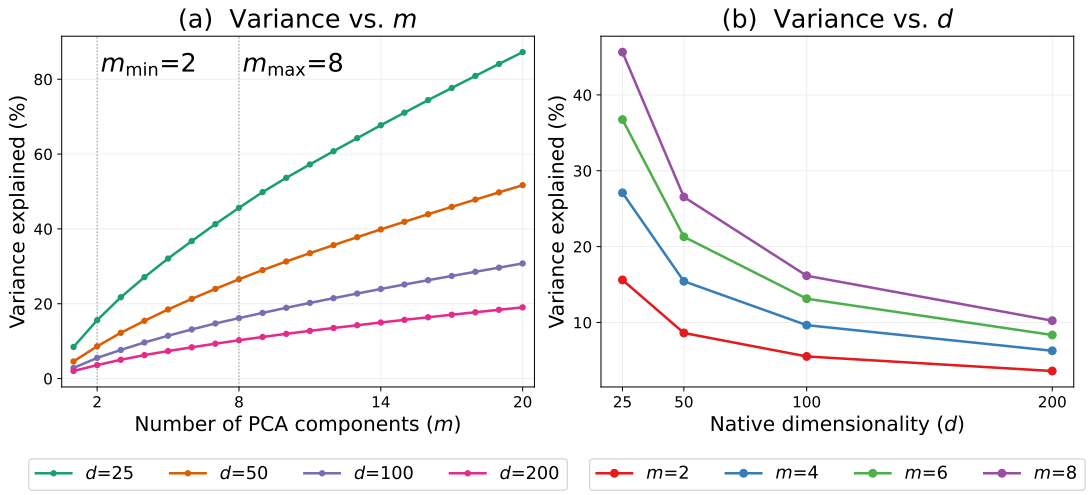

Figure D9: Variance retained by PCA projection as a function of the number of PCA components m and native dimensionality of the dataset d. **(a)** Cumulative variance vs. m for the GloVe-25, -50, -100, and -200-angular datasets ($N = 1.18 \times 10^6$). Dotted vertical lines denote the minimum and maximum PCA components explored during hyperparameter optimization of multiprobe grid. **(b)** Variance explained at fixed $m \in \{2, 4, 6, 8\}$ as a function of native dimensionality d. Although the absolute variance explained with m components decreases with d, the grid algorithm compensates by adapting its projection depth at query time (Table E1).

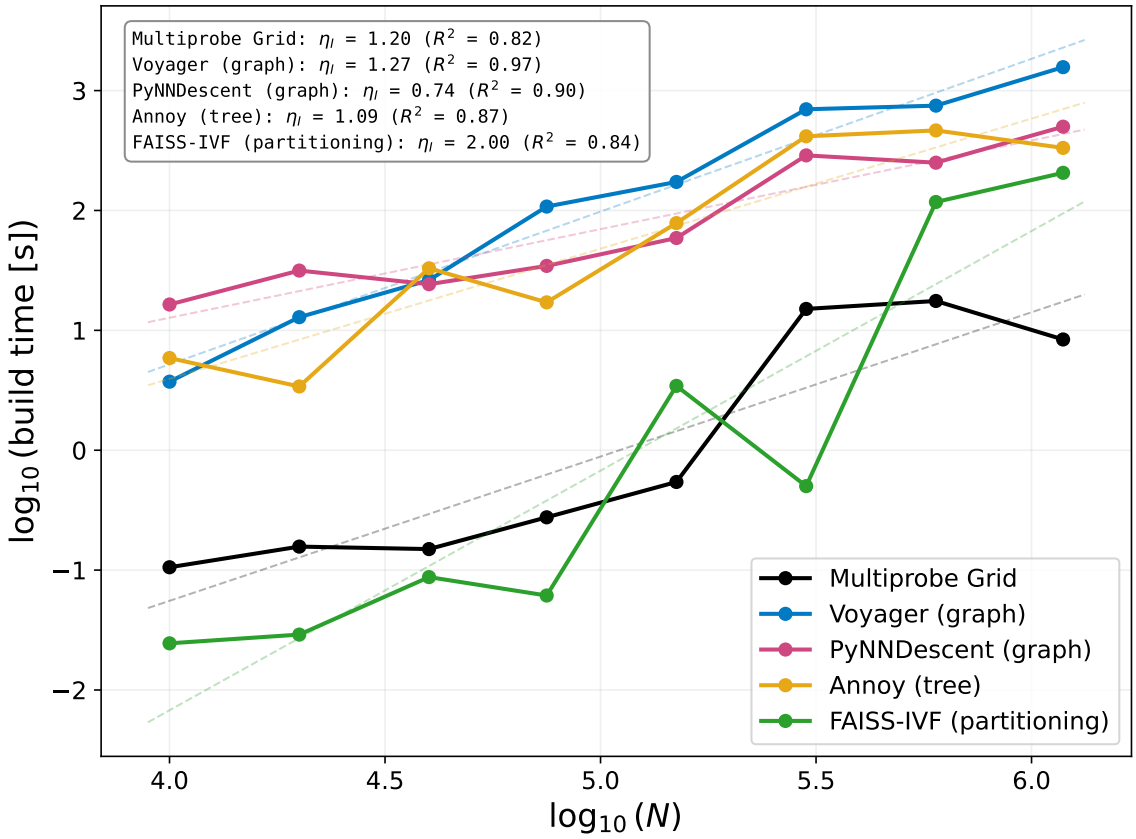

Figure D10: Build time scaling with N across all five algorithms. Datasets: GloVe-200-angular ($d = 200$) subsampled at $N = 10^4$, 2.0×10^4, 4.0×10^4, 7.5×10^4, 1.5×10^5, 3.0×10^5, 6.0×10^5, and the full 1.18×10^6 points. For each algorithm and N, we report the build time of the Pareto-optimal configuration nearest to recall@k=10 $= 0.80$; because this configuration can vary across N, fit quality (reported as R^2) differs across algorithms. The slope of each fit gives the indexing scaling exponent η_I, as reported in Table E2.

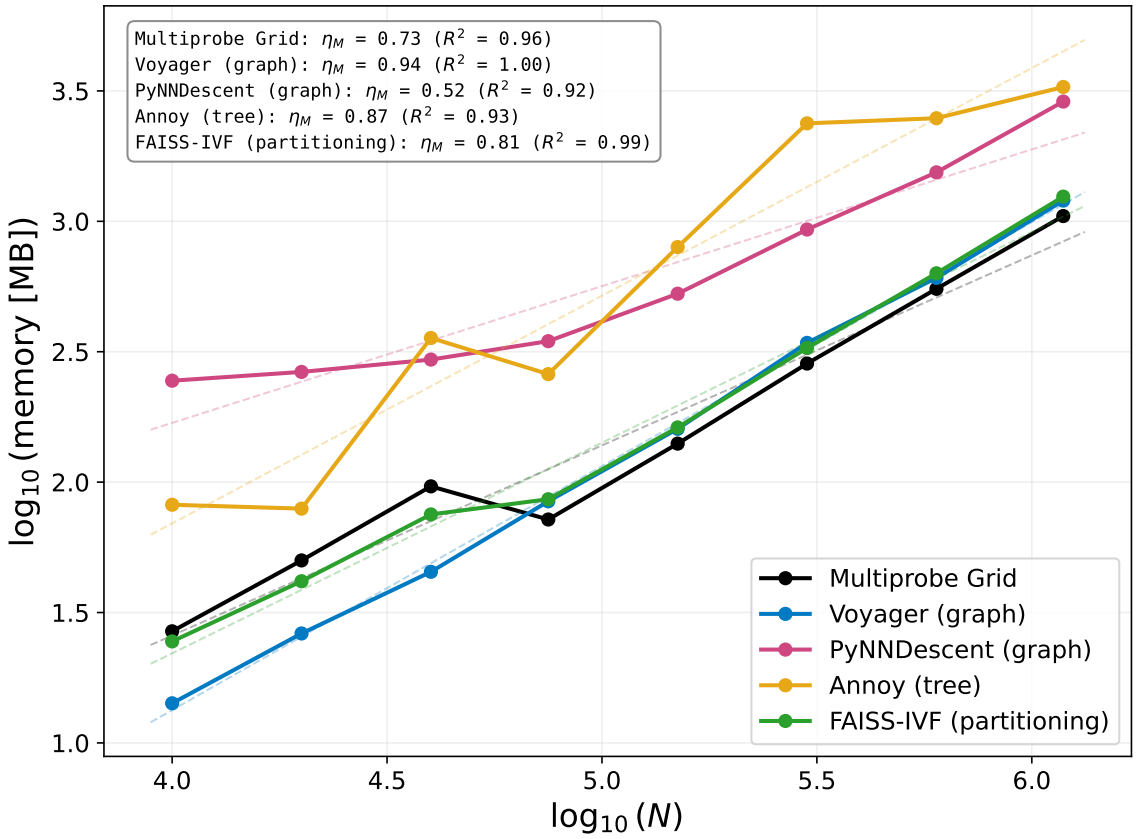

Figure D11: Memory footprint scaling with N across all five algorithms. Datasets: GloVe-200-angular ($d = 200$) subsampled at $N = 10^4$, 2.0×10^4, 4.0×10^4, 7.5×10^4, 1.5×10^5, 3.0×10^5, 6.0×10^5, and the full 1.18×10^6 points. Following `ann-benchmarks` [5], memory footprint is measured as the difference in process resident set size (RSS) before and after index construction. We plot values for the Pareto-optimal configuration nearest to recall@k=10 $= 0.80$ at each N. The slope of each fit gives the memory scaling exponent η_M, as reported in Table E2. Because memory footprint is measured via process RSS rather than on-disk index size, it may include differences in Python runtime and loaded data overhead in addition to the index itself.

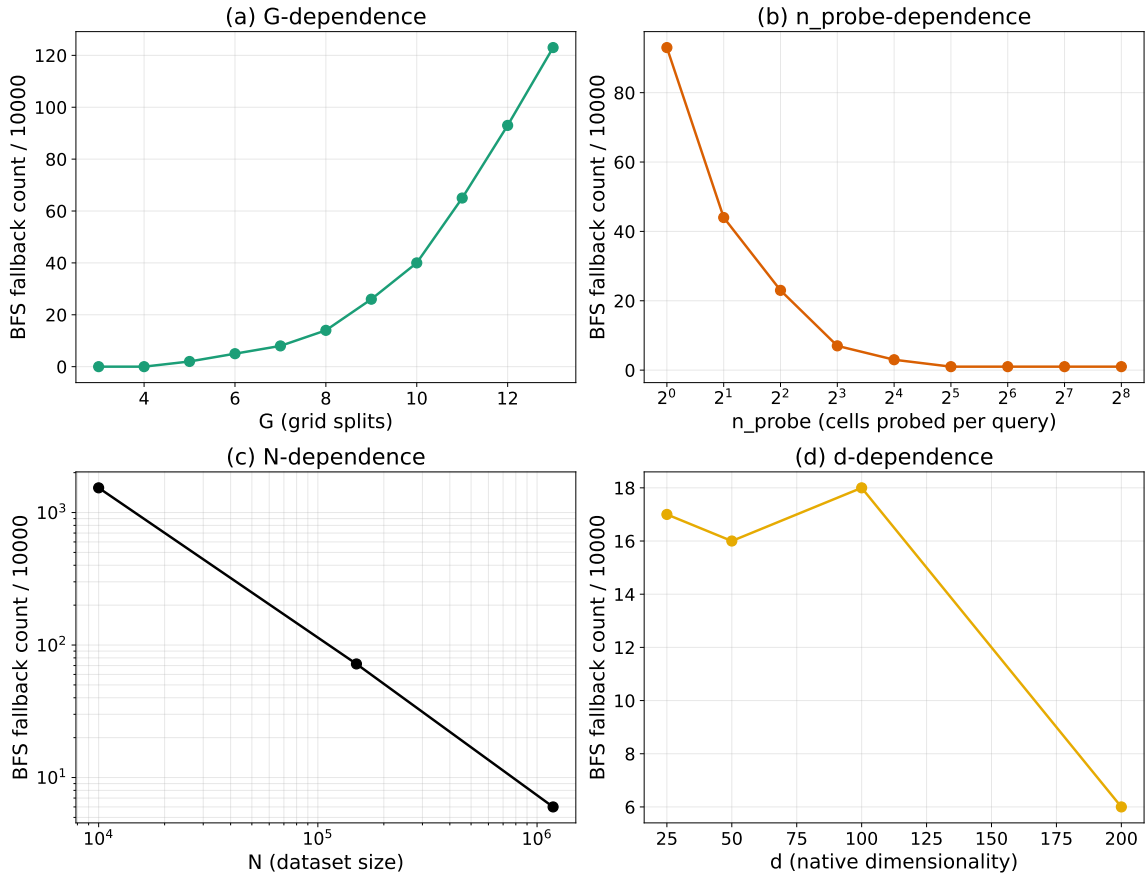

Figure D12: BFS fallback frequency with grid splits (G), probe budget (n_{probe}), dataset size (N), and dataset dimensionality (d). Each panel varies one of the variables while holding the other three constant. Hyperparameter configurations are representative of the range explored during Optuna hyperparameter optimization (Table E1). **(a)** BFS fallback count vs. grid splits G on GloVe-200-angular ($N = 1.18 \times 10^6$), with $m = 6$ and $n_{\text{probe}} = 1$ held fixed. **(b)** BFS fallback count vs. probe budget n_{probe} on GloVe-200-angular ($N = 1.18 \times 10^6$), with $m = 6$ and $G = 12$ held fixed. **(c)** BFS fallback count vs. dataset size N on GloVe-200-angular subsampled at $N \in \{10^4, 1.5 \times 10^5, 1.18 \times 10^6\}$, with $m = 6$, $G = 10$, and $n_{\text{probe}} = 4$ held fixed. **(d)** BFS fallback count vs. dataset dimensionality d across the GloVe-25, -50, -100, and -200-angular datasets ($N = 1.18 \times 10^6$ each), with $m = 6$, $G = 10$, and $n_{\text{probe}} = 4$ held fixed.

Appendix E. Supplementary Tables

Table E1: Pareto-optimal multiprobe grid configurations (m, G, n_{probe}) from benchmark sweeps at each target recall. Dataset: GloVe-25, -50, -100, and -200-angular ($N = 1.18 \times 10^6$). Each table entry represents the Pareto-optimal configuration whose measured recall is closest to the target recall. Parenthetical values indicate the actual measured recall@k=10. Note that the entry for d=200 at a target recall of 0.95 matches the entry for d=200 at a target recall of 0.90, reflecting the algorithm's recall ceiling at that dimensionality.

Recall@k=10	d=25	d=50	d=100	d=200
0.30	(5,13,2) *(.27)*	(6,9,4) *(.28)*	(5,14,32) *(.30)*	(5,14,32) *(.28)*
0.40	(5,13,4) *(.40)*	(6,9,8) *(.40)*	(6,9,16) *(.38)*	(5,7,4) *(.38)*
0.50	(6,9,4) *(.50)*	(5,13,32) *(.50)*	(6,9,32) *(.50)*	(7,7,32) *(.49)*
0.60	(5,14,16) *(.62)*	(7,7,16) *(.60)*	(8,2,8) *(.61)*	(7,7,64) *(.60)*
0.70	(5,14,32) *(.71)*	(7,7,32) *(.72)*	(7,2,8) *(.71)*	(8,5,32) *(.68)*
0.80	(7,7,16) *(.80)*	(7,7,64) *(.81)*	(8,4,32) *(.77)*	(6,5,16) *(.78)*
0.90	(6,9,64) *(.90)*	(6,5,16) *(.89)*	(6,5,32) *(.90)*	(6,2,4) *(.90)*
0.95	(2,4,2) *(.95)*	(6,5,32) *(.95)*	(8,4,128) *(.94)*	(6,2,4) *(.90)*

Table E2: Scaling exponents at recall@k=10 $= 0.80$ (GloVe-200, angular; R^2 in parentheses).

Algorithm	Family	$C(N)$	$I(N)$	$M(N)$
Multiprobe Grid	**grid**	$N^{0.94}$ (1.00)	$N^{1.20}$ (0.82)	$N^{0.73}$ (0.96)
Voyager (HNSW)	graph	$N^{0.44}$ (0.92)	$N^{1.27}$ (0.97)	$N^{0.94}$ (1.00)
PyNNDescent	graph	$N^{0.49}$ (0.94)	$N^{0.74}$ (0.90)	$N^{0.52}$ (0.92)
Annoy	tree	$N^{0.59}$ (0.99)	$N^{1.09}$ (0.87)	$N^{0.87}$ (0.93)
FAISS-IVF	partition	$N^{0.53}$ (0.96)	$N^{2.00}$ (0.84)	$N^{0.81}$ (0.99)