technische universität
dortmund

Master Thesis

Bottom-Up Hierarchical Search Graph Generation

Colin Kolbe

July 27, 2025

Begutachtung:
Prof. Dr. Erich Schubert
Dr. Erik Thordsen

Technische Universität Dortmund
Fakultät für Informatik
Data Mining Arbeitsgruppe
https://dm.cs.tu-dortmund.de/

Abstract

Graphs have been shown to be the best-performing approximate nearest neighbor search (ANNS) structure in terms of recall and queries-per-second (QPS). The Hierarchical Navigable Small World (HNSW) graph is constructed by stacking multiple subgraphs on top of each other, where each contains a subset from the previous layer, thus creating a hierarchy graph with a corresponding redundancy. The hierarchy architecture enables the HNSW graph to reach state-of-the-art search performance by efficiently selecting entry points to its bottom level, which contains the complete training data. In this master's thesis, we present the Hierarchical Search Graph Framework (HSGF), which allows extending any established ANNS graph into a hierarchy graph by following a similar hierarchy design as for the HNSW graph, albeit with a different construction schema. We present a novel evaluation technique for hierarchy graphs that allows us to measure the effects of the hierarchy design on the search performance by querying each level-stacked subgraph separately. This technique allows us, for example, to show that the effects of a hierarchy design on the search performance on a dataset with a high local intrinsic dimensionality (e.g., GLOVE-100) are insignificant for HNSW and negative for an HSGF configured with a Dynamic Exploration Graph (DEG). Our findings suggest that ANNS graphs with a non-constant and low average out-degree, like Relative NN-Descent, are less receptive to improvements from a hierarchy design and show gains only in terms of QPS. Additionally, we find that HSGF configurations that use the same graph type on all their levels perform better than mixed configurations. However, we find no significant performance differences for the tested strategies for the selection of the subsets of the training data for the higher levels. The additional overhead for constructing an HSGF over its established bottom-level graph is marginal because of significantly smaller subsets on higher levels. For an HSGF configuration with the DEG we show on the LAION-10M dataset that the hierarchy architecture can improve the base graph's search performance between 3-15% in terms of QPS in high recall ranges and improve the recall in low recall ranges between 2-12% above the performance curve for the parameter for the size of the extended search results list of the greedy search. Overall, we find for all HSGF configurations, as well as for the HNSW, that the (positive) hierarchy effects for both recall and QPS converge to zero and may even turn negative for higher recall ranges and search sizes of $k > 100$.

Zusammenfassung

Graphen sind die aktuell am besten performenden approximativen Nächste-Nachbarn Suchstrukturen (ANNS) für Suchergebnisse mit hoher Trefferquote und viele Suchanfragen pro Sekunde. Der Hierarchical Navigable Small World (HNSW) Graph wird konstruiert in dem mehrere Sub-Graphen, die jeweils eine Teilmenge der Punkte der vorherigen Ebene enthalten, übereinandergeschichtet werden und so einen redundanten, hierarchischen Graphen erzeugen. Die Hierarchie-Architektur ermöglicht es dem HNSW, durch eine effiziente Auswahl der Startpunkte für die Suche auf dem untersten Sub-Graphen, welcher die kompletten Trainingsdaten enthält, modernste Suchleistung zu erreichen. In dieser Masterarbeit präsentieren wir das Hierarchical Search Graph Framework (HSGF), welches es ermöglicht jeglichen, etablierten ANNS-Graphen zu einem hierarchischen ANNS-Graphen zu erweitern. Dabei wird eine ähnliche Hierarchie-Architektur wie beim HNSW angewandt, um eine effiziente Auswahl der Startpunkte zu reproduzieren, jedoch mit einem anderen Konstruktionsschema. Wir präsentieren eine neue Evaluationstechnik für hierarchische Graphen, welche es ermöglicht die Effekte einer Hierarchie-Architektur auf die Suchleistung zu messen, indem jeder Ebenen-gestapelte Sub-Graph separat durchsucht wird. Anhand dieser Technik zeigen wir zum Beispiel, dass die Effekte einer Hierarchie-Architektur auf die Suchleistung für Datensätze mit hohen lokaler intrinsischer Dimensionalität (wie z.B. GLOVE-100) für HNSW nicht signifikant und für einen HSGF basierend auf einem Dynamic Exploration Graph (DEG) negativ sind. Unsere Ergebnisse deuten an, dass ANNS-Graphen mit einem nicht konstanten und niedrigen Durchschnittsgrad, wie z.B. Relative NN-Descent, weniger empfänglich sind für Leistungsverbesserungen durch eine Hierarchie-Architektur und nur Verbesserungen in der Anzahl der Suchanfragen pro Sekunde zeigen. Darüber hinaus finden wir heraus, dass HSGF-Konfigurationen, die den gleichen Graph-Typen auf allen Ebenen verwenden bessere Leistung zeigen als gemischte Konfigurationen. Allerdings finden wir keinen signifikanten Einfluss der unterschiedlichen Strategien zur Auswahl der Punkt-Teilmenge für die höheren Ebenen auf die Suchleistung. Der zusätzliche Aufwand für die Konstruktion eines HSGF-Graphen gegenüber einem existierenden Graphen ist dabei minimal, auf Grund der signifikant kleiner werdenden Teilmengen der höheren Ebenen. Für HSGF-Konfigurationen basierend auf dem DEG zeigen wir auf dem LAION-10M Datensatz, dass die Hierarchie-Architektur die Suchleistung des Graphen der untersten Ebene um 3-15% in Bezug auf die Suchanfragen pro Sekunde für hohe Trefferquoten-Bereiche verbessert, und die Trefferquote in niedrigen Trefferquoten-Bereichen um 2-12% über der Suchleistungskurve für den Parameter der erweiterten Suchergebnissliste der Greedy Search verbessert. Insgesamt finden wir heraus, dass für alle HSGF-Konfigurationen, sowie für den HNSW-Graph, die (positiven) Hierarchie-Effekte in hohen Trefferquoten-Bereichen, sowie für Suchergebnisse mit $k > 100$, bezüglich der Trefferquote und der Suchanfragen pro Sekunde gegen null konvergieren und sogar negativ werden können.

Contents

1 Introduction

The digitalization and continuous technological advancements have led to the rapid rise of large quantities of data, where individual datasets often consist of billions of data points. Handling these large quantities of data has, as a result, become a key activity in today's digital world and is, as such, an active field of research. Generating new insights from data is of particular interest, which makes its efficient processing and analysis ever more important. One fundamental task in this area is the k nearest neighbor search (KNNS). As the name suggests, it is concerned with the search for the k closest points, the neighbors, to a given query point within a given dataset. However, calculating the exact neighbors from a raw dataset is computationally expensive and therefore impractical on large datasets, especially in higher dimensions. These requirements and constraints for fast nearest neighbor search on large datasets gave rise to a variety of approximate nearest neighbor search (ANNS) methods.

ANNS methods work by constructing search structures that allow a search algorithm to compute the distances only to a small subset of the points in a dataset to find close neighbors. This reduction of the search scope makes ANNS methods approximate but also magnitudes faster than an exact search. However, this approximation means that a set of approximate neighbors might not include all exact neighbors as found by the exhaustive KNNS, but instead other points from the training data, such as relatively close neighbors. For many use-cases, this drop in recall is acceptable considering the significantly shorter search time for a given query.

A popular and illustrative example for the application of ANNS, which has direct touchpoints with a person's everyday life, is in product recommendation systems. For instance, on a music streaming platform, a user receives recommendations for new songs based on other users with a similar listening history [51]. The search in this case should work on demand, or in other words, be fast, while small errors in the neighbors, here songs, will probably not negatively impact the quality of the overall music recommendation for the user. Recently, ANNS methods got an additional influx of attention through the rise in popularity of large language models and particularly retrieval augmented generation applications, which rely on fast vector databases [30].

Designing the architecture of ANNS methods is an active field of research and is concerned with balancing different trade-offs. For example, to reach higher levels of recall, the search algorithm could search more exhaustively, but consequently, this leads to longer search times. Computing more information in advance might enable the search to reach higher recall without being more exhaustive, but it leads to longer construction times and potentially larger memory consumption. Selecting which information to compute in advance and which not becomes, therefore, an essential task when developing the architecture and construction algorithm of an ANNS structure.

ANNS methods can be divided into five approaches: hashing-, quantization-, tree-, clustering-, and graph-based methods [18, 10]. The idea of graph-based methods is to represent the points in a dataset as vertices and let the edges represent neighbor relations between points. Current state-of-the-art ANNS methods are primarily graph-based and can achieve average recall values of more than 95% depending on the parameter configuration and dataset [6]. At the same time, these methods also deliver high performance in terms of queries-per-second (QPS), making them very attractive for a wide range of use cases. Nonetheless, each of those methods has its limitations. For example, the research in recent years has been focused on achieving high recall and fast query speeds, and only recently started focusing more on short construction times.
Next, we look at the Hierarchical Navigable Small World (HNSW) graph and its limitations more closely to motivate towards our work. In Chapter 3, Related Work, we discuss different ANNS methods, especially graph-based methods, in more detail.

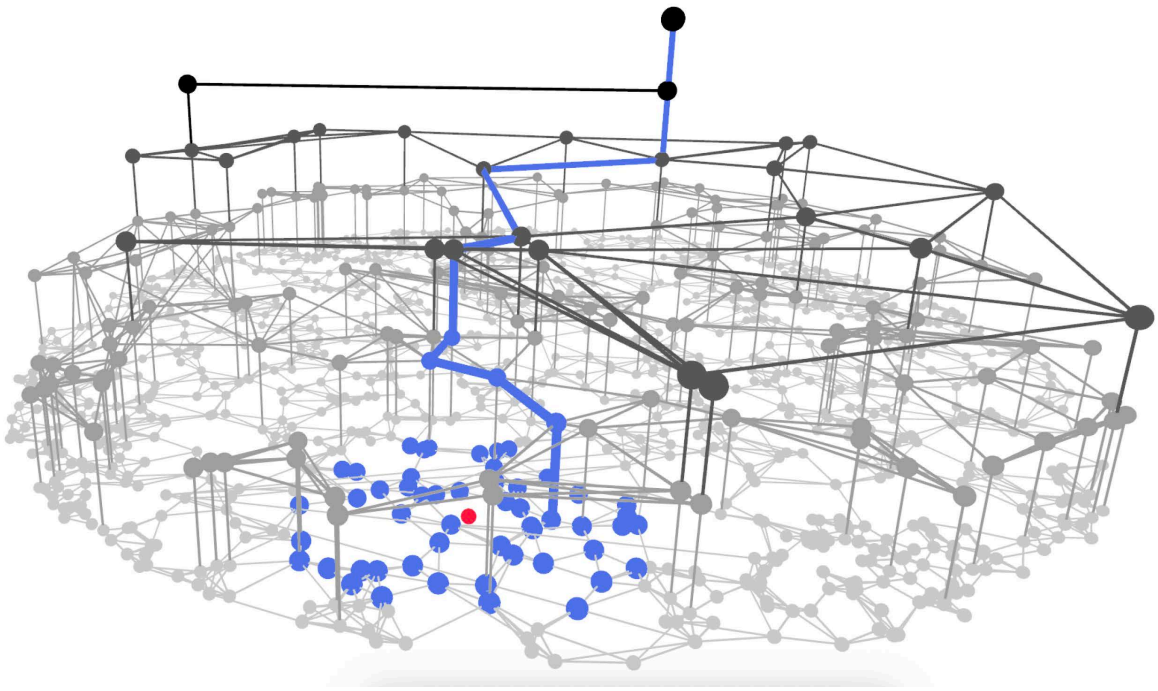

Figure 1.1: Visualizing the HNSW graph and an exemplary search for a query point (red) and its search path, as well as the found neighbors (blue). This Figure is a copy of Figure 3.4 in [24]; we therefore do not take ownership of it.

The HNSW graph by Malkov and Yashunin [35] is currently one of the most popular and best-performing ANNS methods. HNSW graphs are considered state-of-the-art due to their fast search speed in high recall ranges on large and high-dimensional datasets. An HNSW graph is a search structure with multiple subgraphs, also called levels, where the bottom level contains all points and each higher level contains a subset of points from the level below. The points from one level are connected to their *peer-points* on the level directly below. The search on a hierarchical graph, like HNSW, works by sequentially searching each level from the top level down to the bottom level and restarting the search on each level with the *peer-points* of the so far found points as entry points. The search

starts with a random set of entry points from the top level. The idea behind this hierarchical design is the so-called ”zoom-out” and ”zoom-in” phase during the search [35], so that the higher levels serve as an efficient strategy to find *good* entry points to the bottom level, which are closer to the target neighborhoods, thus improving the search performance. A visualization for the HNSW graph and exemplary query can be seen in 1.1.
During the construction phase of the HNSW, however, this results in many computations due to the structure's inherent redundancy coupled with the extensive greedy search. Each point is sequentially added into the HNSW graph starting from the top of the graph and traversing down to the bottom, and inserting it on its designated highest level as well as all the levels below, including the bottom level. The consecutive build-up of the search structure is necessary for HNSW because of logical relations between the points during insertion, as during construction, the current structure is used to navigate to the place of insertion. Consequently, HNSW does not achieve major speedups when constructed in a parallel setting, which results in significant construction times and is one of its main drawbacks [35]. We return in more depth to HNSW in Chapter 3.

We believe that a hierarchy design is a feature that works not only on the HNSW. Therefore, we present in this work the Hierarchical Search Graph Framework (HSGF), which builds on the hierarchy idea of Malkov and Yashunin [35] but takes a more generalized approach to build hierarchical approximate search graphs. The objective of HSGF is to achieve performance levels that are competitive to current state-of-the-art methods, like HNSW, in terms of recall and search speed, while constructing a hierarchy graph in less time than HNSW. In addition, our architecture is meant to work as a framework such that it extends any established ANNS graphs with a hierarchy design and, as a result, offers a wide range of configurations. In contrast to many other works in the current literature, we focus in our work on a parallel setting when constructing graphs. This offers a more practical view on the construction time of an ANNS graph, which is therefore closer to real-life applications and additionally allows us to run more extensive experiments as part of our evaluation due to the shorter construction times.

We follow the hierarchy idea of [35], with the goal to achieve similar search performance, but plan to treat the level-wise graph construction as a black box and separate it from the level-wise subset selection. A method for the black box is a graph builder, and as such, it must take a set of points as input and return an (approximate) nearest neighbor graph for those points. A subset selection method must select a reasonably smaller subset from the set of points that are part of the respective level's graph, which are then used for the next higher level and connected to and from the current level. The applied heuristic for the subset selection becomes, therefore, the second key parameter, next to the level graph builder, for our search structure. This means that each level, or sub-graph, of the HSGF structure is built sequentially from the bottom level up to the top level, instead of sequentially inserting each point top-down as done for the HNSW graph. Note that for the search, the graph is still traversed top-down by using a greedy search because, like the HNSW graph, all points are part of the bottom level of the HSGF graph structure. The HSGF's overall construction is separated into the construction of its levels, which are independent, regarding their construction, of the other levels. Consequently, each level can be built with a different graph construction algorithm as well as a subset selection heuristic. This is a key architecture choice as it allows us to take advantage of ANNS graphs, which during their

construction see a high speed-up from a parallel setting. This applies similarly to the subset selection heuristics. The HSGF serves, therefore, as an extension for established ANNS graphs by adding hierarchy layers on top of a respective graph with the aim of improving their overall search performance. The black box approach, however, means that the HSGF as a framework has no influence on the construction of the level-graphs and, therefore, a potential speed-up in a parallel setting is almost exclusively dependent on the speed-up of the level-graphs.

Furthermore, this architecture allows us to configure the search structure to suit different use cases. For example, we could choose graph methods for the level graphs, which are more suitable for certain types of data(sets). Another possibility is to select more exact graph methods on higher levels, on which, by design, the number of points is decreasing, and therefore the trade-off between computational costs and accuracy becomes acceptable. At the same time, we can imagine that the opposite approach might be beneficial as well. To achieve high recall and fast search performance, we need to quickly find the best entry points, or neighborhoods, and then, within those neighborhoods, find the closest neighbors to a given query point. The idea behind a hierarchical ANNS graph is to use the hierarchy design to quickly navigate to the right entry points on the bottom level and use the graph on the bottom level to find the closest neighbors. This means that a high-quality graph on the bottom level might be more important than a high-quality graph on higher levels. Additionally, the navigation on the upper levels is largely dependent on the subset selection heuristic, which determines which points serve as potential entry points coming from the level above. Finding the right heuristics could therefore be a key factor in determining the overall recall and search speed of an HSGF graph.

In our evaluation, we focus on analyzing the average recall, search speed, and construction time in a range of different (hyper-)parameter configurations and search settings. We follow common methodology and select established ANNS datasets among others presented in [6]. We compare different HSGF graphs to other graphs on established ANNS datasets. Given that we derive our main idea from the HNSW method, a focal point is the direct comparison to HNSW based on our objectives. As we focus on the search performance and construction time, the ideal outcome would be to reduce the construction time while achieving better search performance. However, improving only either the recall, QPS, or construction time while the other two measures are on par with HNSW or the used graph builder for the bottom level of HSGF would also be considered a success. We define our research questions in more detail at the beginning of the Evaluation Chapter 5.

We implement the HSGF in the language Rust[1] and make use of two Rust libraries, which are provided by the supervisor of this thesis to support future research and development[2]. Additionally, we implement three established ANNS graph methods in Rust, namely DEG by Hezel et al. [25], EFANNA by Fu and Cai [17], and NSSG by Fu, Wang, and Cai [18], which allows us to evaluate our search structure more broadly.

[1] https://www.rust-lang.org/

[2] The libraries are currently unpublished but are briefly explained in the Main Chapter.

1.1 Thesis Structure

This thesis is structured into five Chapters. First, we introduce our thesis's field of research and present current limitations of a state-of-the-art ANNS method to demonstrate the relevance for our work. In the second Chapter, Preliminaries 2, we define the necessary notation and definitions for this work, while also going into more depth about the fundamentals of approximate nearest neighbor search. In the third Chapter, we discuss Related Work 3. The fourth and main Chapter is where we present the Hierarchical Search Graph Framework as our primary contribution 4. In the Evaluation Chapter 5, we then present our methodology and present the results of the evaluation of our new search structure in comparison to the HNSW graph. Lastly, in the Discussion Chapter 6 we discuss our evaluation results, examine our limitations, summarize our contributions, and highlight potential avenues for future research.

2 Preliminaries

In this chapter, we define the notation that is used throughout this work and examine the fundamentals of approximate nearest neighbor graphs in more detail. We closely follow the common notation in this field and specifically follow [26].

2.1 Notation

We specify the following notation for our work. Let $P = \{p_1, .., p_n\}$ denote a finite dataset with $p_i \in \mathbb{R}^d$, where d is number of (data-)dimensions, and $n \in \mathbb{Z}$ points. Let $q \in \mathbb{R}^d$ denote a query point and $\delta(\cdot, \cdot)$ the distance between two points. In the context of search structures, we refer to the query point as indexed when $q \in P$ is part of the search structure itself, and otherwise as unindexed. We therefore refer to P also as our training data. In the context of ANNS methods, we assume the dataset as well as the dimensionality of its data points to be reasonably large to avoid special cases that appear in lower dimensions and small datasets. For example, a brute force approach of calculating all distances to find the nearest neighbors might be more practical in those scenarios.

Let $KNN(P, q, k)$ denote the set of $k \in \mathbb{Z}$ points which are the k closest neighbors of q in P, which we also refer to as the true or exact neighbors (see Equation 2.1). Let $ANN(P, q, k, ef)$ denote the set of k points out of P which is returned by a given approximate nearest neighbor search for a given q. Let $ef \geq k$ with $ef \in \mathbb{Z}$ denote the size of the extended neighbor candidates list of the *greedy search*, which is an important parameter influencing the recall of a greedy search algorithm on ANNS graphs [35].

Let $k@ef$-recall denote the average recall over all $q_j \in Q$ given a set of query points Q. We use the intersection between $ANN(P, q, k, ef)$ and $KNN(P, q, k)$ to define the recall, instead of defining it via an ϵ-approximation based on the distances to the query point (see Equation 2.3 and 2.4, respectively). We therefore do not use the additional parameter ϵ in our work but focus on the true recall, which we want to highlight as an important choice, as an ϵ based recall might deliver significantly different results. Consequently, we denote the $KNN(P, q, k)$ as the ground truth, which is to be approximated. A measure that captures both approaches might show a more comprehensive view of the result of an ANNS method, but we are not aware of such a measure and consider it beyond the scope of this work. The same choice is reflected in our decision not to define the ANNS set by using an ϵ-neighborhood as shown in Equation 2.2, but define that "only the probability of finding the true nearest neighbor is guaranteed" [36]. We define Equations 2.1, 2.2 and 2.3 as defined in [26] and Equation 2.4 as defined in [6] with our slightly extended notation. However, for our work, only Equation 2.1 and Equation 2.3 are directly relevant.

$$\forall x \in \mathrm{KNN}(P, q, k) \qquad \forall x' \in P \setminus \mathrm{KNN}(P, q, k) \qquad \delta(x, q) \leq \delta(x', q) \tag{2.1}$$

$$\forall x \in \mathrm{ANN}(P, q, k, ef) \quad \forall x' \in P \setminus \mathrm{ANN}(P, q, k, ef) \quad \delta(x, q) \leq (1+\epsilon)\delta(x', q) \tag{2.2}$$
$$\text{for } \epsilon > 0$$

$$\text{k@ef-recall} = \frac{1}{|Q|} \sum_{q \in Q} \frac{|\,\mathrm{ANN}(P, q, k, ef) \;\cap\; \mathrm{KNN}(P, q, k)\,|}{k} \tag{2.3}$$

$$\epsilon\text{-k@ef-recall} = \frac{1}{|Q|} \sum_{q \in Q} \frac{|\,\{p \in \mathrm{ANN}(P, q, k, ef) \mid \delta(p, q) \leq (1+\epsilon)\delta(p_k^*, q)\}|}{k} \tag{2.4}$$
$$\text{for } \epsilon > 0 \quad \text{and} \quad p_k^* \text{ as the k-farthest point in KNN}$$

Proximity Graphs

Let $G = G(V, E)$ denote a directed graph. Each data point in P is represented by a vertex in the graph, creating the set of vertices V. Each edge in the set of edges E represents a relationship between two vertices. We call edges *short* or *long* depending on the distance between a pair of connected vertices. The out-degree of a vertex is defined as the number of outgoing edges from that vertex, and the in-degree is defined as the number of incoming edges to that vertex. In the case of an undirected graph, these two would collapse to the degree of a vertex, as an edge would represent a symmetric relation between two vertices. Vertices which have a large (out-)degree are referred to as *hubs*. For most of our purposes, it does not matter if the edges are weighted, besides implementation details. Let $v, u \in V$ denote two vertices and $(v, u) \in E$ the edge connecting them. We refer to points, vertices, or nodes interchangeably and therefore similarly define the distance between two vertices as $\delta(v, u)$. Let $N(G, v) = \{u \mid (v, u) \in E\}$ refer to the set of neighboring vertices of v in G. We define a path between two vertices as the sequence of vertices in which each consecutive pair of vertices has an outgoing edge from the first element to the second of that pair. The length of that path, denoted as l, is the number of edges that need to be traversed from the start to the end of that path. A monotonic path is a path where the distance from the start node to the end node is getting smaller with each step (also hop) on that path. We call an undirected graph *connected* if there exists a path between all pairs of vertices in the graph. More generally, in this work, we consider a directed graph *connected* as long as its undirected version would be connected. A graph is called *sparse* or *dense* depending on how strongly connected it is.
In this work, we generally denote a graph as directed. Furthermore, we would like to point towards the *not always intuitive* subtleties of percolation theory when thinking about graphs on large and especially high-dimensional datasets [48].

2.2 Fundamentals

In this section, we describe how the greedy search algorithm works, explore some of the fundamental types of nearest neighbor graphs, and then transition to approximate nearest neighbor graphs. In the next Chapter Related Work 3, we discuss modern ANNS methods which often approximate one or multiple of the following fundamental graphs.

2.2.1 Search on Proximity Graphs

Real-life datasets consist of numerous points that are often embedded in a high-dimensional vector space. Exact nearest neighbor search algorithms suffer from the *curse of dimensionality*, which posits that these algorithms are exponentially dependent on the dimension of the data's representational state [1, 11]. Although the datasets' representational dimension might be significantly larger than their intrinsic dimension, which is "the real number of dimensions in which the points can be embedded while keeping the distances among them" [p.281, [11]]. Approximate nearest neighbor search was proposed to mitigate this issue by allowing errors in the set of returned points. ANNS structures themselves were proposed to enable an ANNS algorithm to search effectively and repeatedly on a large dataset. Another problem that arises in high-dimensional spaces and is also referred to as the *curse of dimensionality* is that the distance to the closest neighbor approaches the distance to the furthest neighbor [9]. Consequently, the distance comparisons during nearest neighbor search come down to ever-smaller differences, which can become problematic for the precision of the result because of the inherent imprecision of floating-point computations.

We define the most commonly used algorithm for the approximate nearest neighbor search on graph-based indices, namely the greedy search. The *greedy search* as shown in Algorithm 1 is based on the definition for the *Search-Layer* algorithm from [35].[1] The algorithm is a gradient-descent-based and greedy approach for searching on a graph, as a locally optimal decision is made at each step. Additionally, it is modified by using an extended list of candidates, whose size is configured via *ef*, to allow the algorithm to "backtrack" during the search from the best vertex to other previously found vertices and search further from these [23, 35].
In short, the search algorithm traverses the graph from a set of given entry points towards the query point by choosing the path towards the closest neighbor at each step. This process continues until a stopping condition is reached, while collecting an extended candidates list from which the k closest ones to the query are returned at the end [35]. Next to the greedy search is the range-search, used for example in [26], a very similar approach which introduces ϵ as a parameter to configure an extended range to select new candidates, compared to *ef*. The underlying assumption for the search on ANNS graphs, and their construction, is that "a neighbor of a neighbor is also likely to be a neighbor" [16].

The algorithm receives as input parameters: a graph G, the query point q, the number of search results k, a set of entry points C (also called candidates) which serves as a dynamic

[1] We change the following variable names for the Greedy search compared to the source: $ep \mapsto C$ (while C stays the candidates list), $v \mapsto B$, $W \mapsto X$, $e \mapsto n$

neighbor candidates list and contains one or multiple points, and the parameter $ef \geq k$ as the maximum size of the extended neighbor. The *greedy search* algorithm iterates over the list of candidates and in each step chooses the current candidate c which is closest to the query. The search considers the candidate's neighbors as new candidates if the candidate is closer to q than the furthest element in the extended neighbor list X. Each unvisited neighbor is added to the candidates and extended neighbors list if the point is closer to q than the current furthest point in X or the extended neighbors list contains fewer than ef points. If, after this addition, X is larger than ef, the list's furthest element to q is removed. The algorithm stops once there are no more new candidates in the list or when the current candidate is further away from q than the furthest element in X, and lastly returns the k closest points in X to q.

The ef parameter sets the maximum size of the extended neighbors lists and thereby controls the amount of "backtracking" during a search pass [35]. The extended search radius is therefore based on the furthest element in the list. Once the list is full, new insertions to the list push the current furthest element out, which consequently reduces the extended search radius with each new addition to the list and is how the algorithm terminates. However, when ef, or also ϵ for the range search, becomes large enough, both algorithms start to visit all points in the graph, assuming such routing exists, and thus are guaranteed to find the true neighbors. By indirectly controlling the number of visited candidates, the parameter ef generally has a positive effect on the recall while increasing the number of computations of an ANNS search.

Nearest Neighbor Graph

A Nearest Neighbor Graph is an intuitive representation of the nearest neighbor relationship among a set of points. The 1-NN graph is a directed graph, which means that each vertex has exactly one outgoing edge to its nearest neighbor. We can easily generalize this to the k-NN graph such that each vertex has k outgoing edges.

Relative Neighborhood Graph

The Relative Neighborhood Graph (RNG) is undirected and connected and was proposed by Toussaint [50] in 1980. The RNG, instead of connecting a vertex to its k nearest neighbors, connects two vertices with an edge if there exists no vertex that is closer to either vertex (see Equation 2.5). The 1-NN graph is a subgraph of the RNG. Like the k-NN graph, the RNG can be computed in $O(n^2)$ time ($d > 2$).

$$E_{RNG} = \{(u,v) \mid u,v \in V, \forall w \in V \setminus \{u,v\} \nexists \delta(u,w) < \delta(u,v) \wedge \delta(v,w) < \delta(u,v)\} \tag{2.5}$$

Algorithm 1 GreedySearch(G,C,q,k,*ef*)

Input graph G, set of candidates (entry points) $C \subset V$, query $q \in \mathbb{R}^d$, number of search results k, size of extended neighbors list *ef*
Output k closest found vertices to q

```
1:  B ← C                                          ▷ set of visited vertices
2:  X ← C                                          ▷ dynamic list of extended neighbors
3:  while |C| > 0 do
4:      c ← argmin_{c∈C} δ(c, q)                   ▷ nearest vertex from C to q
5:      C ← C \ {c}
6:      f ← argmax_{x∈X} δ(x, q)                   ▷ furthest vertex from X to q
7:      if δ(c, q) > δ(f, q) then
8:          break                                  ▷ all vertices in X are evaluated
9:      N ← N(G, c)                                ▷ neighbors of c
10:     N ← N \ B                                  ▷ unchecked neighbors of c
11:     for all n ∈ N do                           ▷ update C and X
12:         f ← argmax_{x∈X} δ(x, q)               ▷ (current) furthest vertex from X to q
13:         if δ(n, q) < δ(f, q) or |X| < ef then
14:             C ← C ∪ {n}
15:             X ← X ∪ {n}
16:             if |X| > ef then
17:                 remove argmax_{x∈X} δ(x, q) from X
18:     B ← B ∪ N                                  ▷ add checked neighbors to visited vertices
19: return k nearest vertices in X to q
```

Gabriel Graph

The Gabriel Graph presented in 1969 by Gabriel and Sokal [20] contains the RNG as a subgraph and is therefore also undirected and connected. In a Gabriel graph, two vertices are adjacent when there exists no vertex that is closer to the two vertices' midpoint (see Equation 2.6).

$$E_{Gabriel} = \{(u,v) \mid u,v \in V, \forall w \in V \backslash \{u,v\} \nexists\, \delta(w,m) < \delta(u,m) \vee \delta(w,m) < \delta(v,m)\} \quad \text{with } m = 0.5(u+v) \tag{2.6}$$

Delaunay Triangulation

The Delaunay Triangulation creates a graph (DG) which contains the Gabriel graph as a subgraph and was presented by Delaunay [14] in 1933. A Delaunay Triangulation graph is an undirected graph in which three vertices are adjacent if there exists no point closer to the center of their circumcircle (or in higher dimensions their (hyper-)sphere), and is the dual graph to the Voronoi diagram [32]. However, for high-dimensional datasets, the DG often approaches the complete graph [23].

Hierarchy Graphs

We define a hierarchy graph as a graph that consists of multiple level graphs, which are stacked on top of each other. The bottom level contains the complete training data, while each layer above the bottom layer contains a subset of the points from the level below itself. The size of each level (also, subset) is the number of points it contains and is meant to be significantly smaller than the previous level. The level sizes are therefore a parameter of a hierarchy graph. We define the height of a hierarchy graph as the number of levels, also layers or level-graphs, that it consists of. Furthermore, we define a level-stacked subgraph of a hierarchy graph as a graph that consists of one to the graph's height many, bottom-up sequentially stacked levels, and always starts with the bottom level. A hierarchy graph, therefore, has its height many level-stacked subgraphs.

2.2.2 Approximate Nearest Neighbor Graphs

In contrast to the precise definitions of the aforementioned types of graphs, approximate nearest neighbor graphs are more a category than a clearly defined graph. For example, a graph with k randomly sampled outgoing edges for each vertex in the graph already serves as an approximation to the k-NN graph, albeit very likely not a good one. Nonetheless, the k-NN graph on a local level is ideal for the given k and serves therefore often as the general approximation target; however, this makes the underlying assumption that all queries are indexed, and secondly, it ignores the role entry points play for the ANNS on large graphs. Additionally, ANNS structures, next to high recall, are concerned with high query speeds for which a large out-degree, as a direct result of a large k, is generally obstructive. Furthermore, k is also a parameter of the ANNS, and thus an ANNS graph might not have been constructed with the ideal approximation target when the k of a query is larger than the targeted k used during construction.

Altogether, this clearly illustrates the necessary balancing of trade-offs when designing the architecture and construction algorithm of an ANNS graph. Therefore, we generalize our definition for an ANNS graph so that the construction of an ANNS graph is concerned with finding an *ideal* selection or sparsity of edges, given a set of points and requirements. The primary metric used for measuring the performance of ANNS methods is QPS over recall, followed by the time needed for constructing the graph.
To construct a graph in a reasonable time, or put differently, with a complexity significantly smaller than $O(n^2)$, different types of existing ANNS graphs try to approximate one of the previously mentioned graphs, as well as additional properties like monotonicity, because these graphs can more easily be approximated with fast algorithms. Knowing which graph is used as an approximation target is therefore a key indicator for the properties and limitations of an ANNS graph.

There are two types of approaches to construct ANNS graphs: (1) the direct approach and the refinement-based approach [40]. Algorithms that use the first approach build up a graph by sequentially inserting each point as a new vertex into the graph and connecting it to existing vertices based on the algorithm's criteria until all points are contained in the graph. In the second approach, an existing graph is refined, such that the edges in

the graph are changed to improve the graph on a metric defined by that algorithm. Furthermore, ANNS algorithms can be differentiated if they use a greedy search algorithm during construction to select or refine a point's neighbors or not. As previously mentioned, the underlying assumption to find new (better) neighbors is to search the current point's neighbors' neighbors [16]. Combinations of both construction approaches are also possible.

We follow Hezel et al. [26] and summarize the following properties an ideal ANNS graph should have: (a) have fast search speed to retrieve a flexible number of k approximate neighbors, (b) the set of retrieved approximate neighbors should have high recall, (c) scale to large datasets with high dimensionality, (d) work with any distance function, (e) have low index construction time, (f) perform well also on unindexed queries, (g) allow points to be efficiently added or removed from the graph. As already mentioned, these requirements mostly stand in trade-off with each other. Therefore, an ideal ANNS graph should have a range of parameters to be able to approximate the *ideal* edge selection of different requirements in practice. A step beyond that would be methods that can auto-tune their parameters, which would allow the user to avoid extensive manual parameter tuning.

Search on Approximate Nearest Neighbor Graphs

Describing the problem differently, we can view the construction of an ANNS graph as an optimization problem of finding the selection of edges that minimizes the (average) number of distance calculations the greedy search needs to make to find the k (approximately) closest neighbors for a set of queries.
The number of distance comparisons is the determining factor for the search speed of ANNS methods. This number itself is determined by the search path length and the out-degree of the nodes on that path [19], which can be indirectly controlled with the parameter ef. Thus, to improve the search speed of ANNS graphs, the search path length needs to be reduced while finding an out-degree that balances QPS and recall [18]. This brings us back again to the role entry points play in determining search speed. In a k-NN graph, the best entry point would be the closest point to the query, which in the case of indexed queries would generally be the vertex corresponding to the query point itself. However, for unindexed queries, the best entry point cannot be known beforehand, which is part of the reason why, in practice, entry points are often selected randomly. In return, this means that an ANNS graph needs to enable the routing from arbitrary entry points to virtually any point in the graph. A different selection process for the entry points might therefore play an important role in improving the search performance of an ANNS graph. As we will see later, one option is to introduce redundancy into the graph's architecture to aid the routing process of the greedy search.

Regardless of how the entry points are selected, the search process can then illustratively be understood as two phases: (1) the navigation phase in which, starting from the entry point, the greedy search tries to find *good* neighborhoods, and (2) the neighbor search itself in which most of the final approximate neighbors are found - the two phases are also described as "zoom-out" and "zoom-in" in [35]. Following this idea, the threshold between the two phases would be defined as the point at which the greedy search stops finding

vertices which are closer to the query [23]. However, we want to be clear that this is not how the greedy search algorithm itself works, as it begins filling a candidate list once the search is started until it cannot find any better candidates and uses the (size of the) extended neighbors list to move beyond local minima. Nonetheless, we will come back to this idea when designing the HSGF search structure.

Miscellaneous

Different distance functions for the R^d space have been proposed, which weigh certain parts of the information between two vectors differently and thus calculate different values for the distances. Some distance functions are designed to suit particular types of data, like text or images. Although distance functions are the fundamental component for comparing neighbors and, therefore, fundamental for ANNS and the construction of ANNS structures, we will not dive deeper into the various existing distance functions as we defined previously that the ideal ANNS graph should work independently of the used distance function.

3 Related Work

In this chapter, we explore related work, specifically works regarding nearest neighbor graphs. We first give an overview of different ANNS approaches, followed by deeper dives into the works that are relevant for this thesis, and finally, explore more recent works.

In addition to the different approaches for constructing ANNS structures, various techniques exist to improve the efficiency of the construction of existing ANNS structures and the search algorithms by further approximating different steps inside those algorithms.
Hashing-based methods, which are primarily based on locality-sensitive hashing (LSH) algorithms [1, 2], use hash functions that map close points onto the same bucket and far-away points onto different buckets. The *sensitivity* of a hash function is then determined by the gap between the probability for collisions of close points and the probability for collisions of distant points. A search function then uses the same hash function to find the *closest* bucket to retrieve the nearest neighbor from that pool of points. For high-dimensional data, multiple hash tables can be used, which consequently results in increased memory consumption.
Quantization-based methods [22, 31] approximate the distance computations by compressing the vectors while retaining the relative distance information of each dimension, which is another approach, next to reducing the number of visited points during search, to reduce the scope and therefore the computational costs. Quantization works by reducing the scope of the values of each dimension of the vector by, for example, clustering all values of a dimension and then representing similar values with their closest centroid. While these methods offer a significant speed-up for the distance calculations, especially for large datasets, they do not perform well in high recall settings.
Tree-based methods [47, 38, 4] build up hierarchical search structures by recursively partitioning the search space. While this partitioning allows for an efficient traversal, it necessitates large amounts of backtracking for high recall settings, as incorrect branch selections cannot be handled downstream.
Clustering-based methods partition the dataset into multiple *shards*. The search algorithm consists of two phases *routing* and *scoring*, which can be broken down to selecting a significantly smaller and suitable set of shards based on the query and finding the approximate nearest neighbors within those shards, respectively. Clustering-based approaches, which are effectively Inverted File Indexes (IVF), are often combined with quantization approaches and achieve reasonable performance levels with very low index construction times [29, 8, 49].
In the previous chapter, we already introduced different types of nearest neighbor graphs. Compared to the other approaches, graph-based ANNS methods generally need more time to be constructed; however, they have been shown to perform significantly better than all other approaches in terms of both recall and QPS on large datasets. For the remainder of this chapter, we therefore focus on graph-based ANNS methods.

3.0.1 Hierarchical Navigable Small World Graph

Malkov et al. [36] propose the Navigable Small World (NSW) graph, which approximates the DG. The graph is constructed by randomly selecting and then incrementally inserting each point into the graph and connecting it to its f closest neighbors. Additionally, this process aims to keep long edges in the graph from the beginning of the construction phase, when the expected distance between points is bigger. However, no measures for limiting the node degree are taken, resulting in hub vertices for large datasets and poly-logarithmic search complexity [43]. They argue that the short edges approximate the Delaunay graph, and the long edges induce the navigable small world properties into the graph, such that the average path length is comparatively low. Given this property and random entry points, the greedy search method can navigate on a short path towards the query point and its nearest (approximate) neighbors.

In a later work, Malkov and Yashunin [35] propose the HNSW graph, which is based on the ideas of the NSW graph and has similarities to the probabilistic skip list [44]. Their objective is to mitigate the limitations of NSW, which are primarily the poly-logarithmic search complexity and the risk of ending up in a false local minimum. They build on the idea of the ”zoom-out” and ”zoom-in” phases of the greedy search on NSW graphs to construct a hierarchical graph structure that maps the two phases onto multiple levels by creating long edges on higher levels and shorter edges on lower levels. As such, HNSW is designed to approximate the DG and the RNG. The construction process of the HNSW graph first assigns each point randomly its highest level, based on an exponentially decaying probability. Each point is then sequentially inserted into the graph by traversing the already existing hierarchy graph by using the greedy search from the top level of the structure downwards. The point is then first inserted on its highest-assigned level, followed by all levels below, including the bottom level. Furthermore, the point is on each level connected to its M neighbors of that level while the maximum degree of a vertex in the graph is limited by a hyperparameter (see M, M_0, and M_{max} in [35]). The neighbor selection follows a defined heuristic, as it does not simply select the m nearest neighbors, but tries to ”create connections in diverse directions” and ensure global connectivity. Additionally, to allow the traversal of the graph specifically between levels, the same point on two consecutive levels is connected to *itself* (also called *peer-point* in our work). This results in a hierarchical graph structure with a graph on each level consisting of a subset of points from the previous level, and the bottom level containing all points of the dataset. This redundancy results in an increased memory consumption for storing the index. However, the applied highest level selection heuristic controls the levels' subset size and thereby the amount of redundancy and additional memory footprint of the index. Next to the hyperparameter for the degree, the already introduced parameter ef of the greedy search plays a significant role in determining the performance of an HNSW graph. A larger $ef_{construction}$ leads to more points visited and thus better neighbors for each point in the graph, but consequently also to a significantly longer construction time as the greedy search is used for the insertion of each point.
They argue that HNSW achieves logarithmic routing complexity by limiting the maximum number of connections of each vertex to a constant. The search uses on the higher levels an $ef_{higher} = 1$ and only on the bottom level an $ef_{bottom} \geq k$ to limit the number of visited

nodes, which underscores the idea of the two-phased search approach of first finding entry points for the bottom level and then finding the neighbors. Nonetheless, HNSW does not remove the risk of ending up in local minima because of limited connectivity on the bottom level [19]. HNSW is considered state-of-the-art because it achieves high (average) recall and fast search performance on a variety of large datasets. However, its major drawback is its comparatively long construction time, which does not see significant speed-ups in a parallel setting. This is primarily because each point is inserted by traversing the current graph via a greedy search, which cannot be completely parallelized as it would lead to inconsistencies in the graph.

3.0.2 Navigating Satellite System Graph

In [13], the authors introduce the Monotonic Search Network (MSNET). An MSNET graph is a strongly connected and search-monotonic graph. These properties allow the greedy search to find the shortest path from one node to the other in the graph; however, without an upper limit on the nodes' degree, finding this shortest path generally involves an excessive number of distance calculations. A minimal MSNET graph has only the minimal amount of edges needed to ensure the monotonic path property and therefore requires fewer distance calculations during search. The authors in [5] present the monotonic Randomized Neighborhood Graph (RNG*(S); not to be mistaken with the RNG), which guarantees polylogarithmic search time complexity.

Fu et al. [19] propose the Monotonic Relative Neighborhood Graph (MRNG) and a practical approximation to the MRNG called the Navigating Spreading-out Graph (NSG) due to the MRNG's long construction time. The MRNG is a directed graph approximating the RNG while also being related to the RNG* and ensuring the monotonic path property, resulting in a sparse but not minimal graph of the MSNET family [18]. The NSG selects exactly one navigating node from where the search always starts, and for which a path, desirably a monotonic path, to all other nodes in the graph is guaranteed. During construction, the node out-degree is limited to avoid hubs; however, to ensure connectivity, a method based on the Depth-First-Search tree is deployed, resulting in a sparser graph with similar low search complexity as the MRNG.

In a follow-up work, Fu, Wang, and Cai [18] propose the Satellite System Graph (SSG) and a practical approximation to the SSG called the Navigating Satellite System Graph (NSSG). The authors focus in their work on three limitations of the NSG: the over-sparsity, the assumption of an indexed query, and the limited scalability. They argue that there exists an optimal degree and therefore an optimal sparsity for an ANNS graph, depending, among other things, on the dataset. The SSG continues to be part of the MSNET family, and its design is based on the argument that a uniform distribution of its outgoing edges in terms of their direction around a node enables efficient traversal in any direction in the graph. As such, the construction of the SSG, specifically the edge pruning strategy, takes not only the distance between two points but also the angles between the edges of a point and its neighbors into account. Additionally, it introduces a hyperparameter to control the graph's sparsity and therefore its recall. The NSSG is designed to approximate the SSG and overcome its high indexing complexity of $O(n^3)$ and thus be more practical. The authors

observe that most long edges in an exact SSG do not contribute significantly to the search routing. The NSSG's construction process, therefore, uses the same algorithm as the SSG, which takes into account the angle between edges but limits the scope of considered candidates by selecting a smaller candidate set from a pre-computed ANNS graph. Extending the NSG's principles, the NSSG selects multiple navigating nodes to ensure connectivity of the graph from these points and uses them as starting points for the search. By starting the *actual* search only from the navigating node, which is closest to the query, they indirectly introduce a two-level hierarchy to the graph. The NSSG algorithm is a refinement strategy and therefore relies on an existing nearest neighbor graph as input. The construction time of an NSSG is therefore also largely dependent on the input graph's construction time, as, for example, a random initialization shows only limited search performance.

3.0.3 Dynamic Exploration Graph

Hezel et al. [25] propose the continuous refining Exploration Graph (crEG; in a first version also called DEG [26]), which approximates the MRNG, but also properties of the DG and RNG, and is designed to additionally perform well on search exploration tasks. The crEG is a directed and weighted graph and is constructed by incrementally inserting each point into the graph and connecting it to the M neighbors that fulfill selected criteria, e.g., the closest neighbors or the neighbors with the longest edge, and are found via a range search. At the same time, the construction algorithm replaces edges from the neighbors of the newly inserted point in such a way that it ensures a degree of M for all vertices in the graph. Additionally, the crEG's construction algorithm applies a dynamic edge optimization process to minimize the average neighbor distance, a metric proposed by the authors. Furthermore, this optimization process balances the number of short and long edges to enable the range search to quickly navigate the graph as well as to find close vertices. The crEG achieves state-of-the-art performance in terms of query speed in high-recall ranges on common benchmark datasets. In a single-threaded scenario, the crEG also shows comparatively long construction times, although this might be in large part because of the optional and relatively slow optimization process. However, the construction algorithm can be well parallelized for a known local intrinsic dimensionality (LID) of the data. In a follow-up paper Hezel et al. [27] propose the Dynamic Exploration Graph (DEG) as an extension to the crEG for dynamically changing datasets and develop their original construction and refinement algorithms of the crEG further.

3.0.4 RNN-Descent

In [16], the authors propose the NN-Descent algorithm for approximating the k-NN graph with arbitrary similarity measures based on the principle that "a neighbor of a neighbor is also likely to be a neighbor". NN-Descent starts with a randomly initialized k-NN graph and then iterates repeatedly over all vertices to improve each vertex's neighbors by searching through its neighbors' neighbors for more similar neighbors until a stopping condition holds. This process involves calculating the reverse neighbors as the k-NN graph is a directed graph. The simplicity and the relative independence of each step of this process

enable the method to see significant speed-ups in a parallel setting. An advantage of NN-Descent is that the construction process does not rely on an ANNS method to find neighbors, which is often a computationally expensive step in other methods.

Ono and Matsui [41] propose the RNN-Descent algorithm, which combines the NN-Descent algorithm, a direct approach, with the RNG construction strategy, a refinement approach. By combining both approaches, RNN-Descent can refine a graph without having to compute the complete graph first. Like NN-Descent, RNN-Descent avoids an expensive ANNS to find new neighbors. As a result, RNN-Descent achieves significantly faster index construction times on common benchmarks while achieving competitive performance levels in terms of recall and QPS compared to state-of-the-art methods.

Additional Related Work

The authors of the FLANN[1] library presented in [38] one of the first approaches to automatic parameter configuration in the context of ANNS. In [23], the authors present the "Fast Approximate Nearest Neighbour Graph" (FANNG), which approximates the RNG and tries to ensure monotonic paths with short edges. FANNG offers a hyperparameter that allows control of the trade-off between computations and recall. In [17], the authors introduce the EFANNA algorithm, which is based on the *NN-Descent* algorithm and also uses a randomly initialized input graph. The authors propose using randomized truncated KD-trees among other structures as input graphs as well, while balancing the trade-off between shorter construction time and potentially higher search performance. Aumüller, Bernhardsson, and Faithfull [7] propose a framework for evaluating ANNS methods and evaluate the performance of a variety of mostly graph-based ANNS methods in terms of their recall, QPS, and index construction time in a single-threaded setting. They publish an online overview of a range of current ANNS methods and their performance. Unfortunately, it currently does not include the most recent methods, e.g., RNN-Descent[2]. Additionally, because the authors also compare different implementations of the same method (e.g., HNSW), they observe significant differences in the performance of different implementations of the same algorithm, which highlights the importance of efficient implementations. In the following years, similar evaluation works have been performed, giving a comprehensive overview of the current state of ANNS methods and their characteristics [33, 46, 53, 49].

Other approaches to improve performance in the context of ANNS focus on increasing the amount of code that can be efficiently executed on the GPU. In [55], the authors propose a framework that splits the graph search into 3 stages, and an optimization method to avoid dynamic GPU memory allocations to improve the overall parallel speedup. Wang et al. [52] introduce a new selective graph update strategy to adapt the NN-Descent algorithm for the GPU architecture. They achieve a 100-250x faster construction time for NN-Descent compared to the single-thread variant.

[1] https://www.cs.ubc.ca/research/flann/
[2] https://ann-benchmarks.com/index.html - accessed 21.02.2025

Recently, new techniques to improve the efficiency of the distance computations have been proposed as a different approach towards approximation, thus increasing the overall performance of existing ANNS methods. Gao and Long [21] propose an algorithm that uses randomization to improve the efficiency of the distance comparisons, which dominate the computational time of the greedy search. Chen et al. [12] observe that most distance computations do not lead to updates in the search result. Therefore, the authors propose an algorithm to approximate the distance function by estimating angles between neighbors to reduce the computational costs and thus speed up the greedy search.

Diwan et al. [15] perform a theoretical analysis of the upper and lower bounds of the average degree needed to expect a navigable graph. Albeit these bounds consider the average degree in a graph, they nonetheless indicate that a large number of edges is needed to achieve high performance levels. In [54], the authors propose the crossing sparse proximity framework (CSPG) to enhance existing ANNS graph methods to reduce the number of unnecessary explorations and thus computations during the search. CSPG works by partitioning the data randomly, selecting random routing vectors which are part of each partition, and then constructing a graph for each of those partitions. Correspondingly, they introduce a new search algorithm to efficiently expand the search to additional partitions. Their results show that CSPG-enhanced ANNS graphs perform slightly faster in terms of QPS in the same recall range. However, they introduce redundancy to their index, resulting in larger memory consumption and longer construction time. Additionally, they find that the ideal number of partitions is in the range of 2-8, while the performance gains decrease with the dataset size.
Although most methods in recent history focused on graph-based ANNS methods, new works based on other approaches continue to be proposed as well. In the context of clustering-based approaches, the authors in [28] propose the reduced-rank regression to improve the efficiency of the score computations needed for clustering-based ANNS methods. An advantage of this method is its scalability to high-dimensional datasets. *The Faiss library* [49] present a summary paper for the FAISS[3] library from Facebook AI Research, which offers a range of functionality for vector similarity search and has become one of the most popular libraries in this area.

The proposed HSGF is designed to extend an existing ANNS graph into a hierarchy graph. We therefore do not propose a new type of graph in itself, but a method to construct hierarchy graphs. We aim to improve the search performance of existing ANNS graphs with our method but without introducing additional approximation, as proposed by some of the previously discussed works. While our work is in that sense related to ANNS *improvement methods*, we see it most closely related to HNSW because of the hierarchy architecture, albeit we do not intend to improve the HNSW graph itself. As the upper levels of the hierarchy design itself are concerned with the entry point selection for the bottom level, our work is related to the work of Oguri and Matsui [39]. The authors introduce a generalization to the MSNET graph called the B-MSNET and additionally show that selecting entry points adaptively is up to 2.3x faster compared to having a fixed entry point. Thus, highlighting the importance of the entry point selection for searching on ANNS graphs.

[3] Facebook AI Similarity Search, Meta

4 Main

In this chapter, we first motivate our proposed ANNS graph structure, called HSGF. We continue with the architecture and parameter choices of our graph, followed by an overview of the implementation details as well as details about the reimplementation of three established ANNS graph algorithms, which will be used as part of our evaluation in the next chapter.

4.1 Motivation

State-of-the-art approximate nearest neighbor search structures are nowadays primarily graph-based, as these can reach the most competitive search performance in terms of recall and QPS. Essentially all graph construction algorithms have hyperparameters, which can be tuned to affect both recall and QPS in either direction, although generally having a counter-effect on the other one. At the same time, these parameters also have a significant impact on the construction time of the graph.
The HNSW graph by Malkov and Yashunin [35] has been shown to perform well both in terms of recall and QPS, minimizing the trade-off between those two; however, with the drawback of a longer construction time and a limited speed-up for the construction in a parallel setting. We argue that the key contribution of their work is highlighting the importance of an efficient selection of *good* entry points, which they manifested in their hierarchy architecture design alongside the redundancy this introduces. Building on this observation, we further believe that the interdependence of the levels during the construction of the HNSW graph is not necessary to improve the entry point selection but instead can be generalized. Therefore, we posit that an ANNS graph with virtually the same hierarchy architecture can be built level by level while showing comparable improvements to the recall and QPS as seen in HNSW. The removal of the dependence between the levels allows us to treat the level graph building as a black box algorithm, meaning that it receives a subset of points from the training data and needs to return a graph for those points. Consequently, it allows the use of practically any established (ANNS) graph construction algorithm on a per-level basis. To ensure the routing for the greedy search through the graph, we connect each level's points, which are a subset of the previous level's points, to their *peer* on the next higher level as done in HNSW. This allows us to design our new graph structure and its construction algorithm more like a framework than an actual specific type of (new) graph, which opens a new *dimension* of parameters to configure our graph structure. We therefore also want to be clear that our structure is not an ensemble of multiple graphs but is one (hierarchical) ANNS graph and constructed by extending an existing graph with a hierarchy design.

The subset selection for each level in the hierarchy graph is the other key component and parameter of our new structure. However, by treating the level building independently, the subset selection becomes more relevant, and similarly can be treated as a black box algorithm that selects a subset of points from a given set of input points. A random selection, as done for the HNSW, continues to be possible. This would allow the parallel construction of all levels at once, in contrast to building each level sequentially from the bottom up. Note that this is not referring to the parallel computing setting, which we will use for executing our code. However, we think a subset selection based on the constructed graph of each level might result in a better selection, as the selection strategy could benefit from the created distribution of edges. Consequently, our graph would need to be constructed from the bottom up, compared to HNSW, where the greedy search traverses the graph from the top down during construction. A subset selection dependent on the graph of the level below the current one would mean that all levels cannot be constructed in parallel at once (see previous note). Yet, this is not a major drawback as the level size on each higher level is designed to be significantly smaller than the level size below, which means that the bottom level is very likely the dominating factor in determining the overall construction time in either case. We call our new ANNS graph Hierarchical Search Graph Framework (HSGF), to indicate that the hierarchy architecture is fixed, but the specific subgraph types and subset selections are parameterized.

Our focus is exclusively on constructing HSGF graphs in a parallel setting, compared to the focus on a single-thread setting as seen most often in the literature when comparing ANNS graphs. We acknowledge that a single-thread setting enables a fairer comparison between different algorithms and generally does not influence the results of the primary metrics, which are recall and QPS. However, we think it largely ignores the realities of large-scale applications, as seen on the industry level, in which these algorithms are primarily deployed. This is because it hides the potentially large differences in construction times of a parallel setting, which is naturally another determining factor for the selection of an ANNS graph algorithm. Our aim is therefore to construct an HSGF graph with similar search performance to an HNSW graph but in less time than HNSW needs. However, in terms of construction time, this limits us to graphs, which can be faster constructed than an HNSW, as the HSGF, because of its black box approach, does not influence the level graphs themselves.

4.2 Hierarchical Search Graph Framework

The architecture of the HSGF graph is straightforward and essentially the same as for the HNSW graph, as it consists of multiple levels where each level is represented by an ANNS graph. The difference to HNSW lies in the construction of the HSGF graph, as shown in Algorithm 2, which receives a set of points as training data, a list of graph builders, a list of point selectors, and the number of levels to be created. We assume that each list contains as many elements as the number of levels. The construction starts by building the bottom level on the full training data with the first level builder, and then continues upwards to build the next levels. After each level's graph is constructed with the level's corresponding builder, the resulting level graph is used to select the subset of points for

the next level via the level's selector function and is then added to the overall HSGF graph. This loop continues until all levels are built, which completes the final HSGF graph.
For querying, the greedy search then traverses the HSGF graph, like the HNSW graph, top-down. It starts with a random set of entry points from the top level's points, and once the greedy search is finished on a level, it continues with the so far found candidates as entry points to the level below, until finally, when finished on the bottom level, the final set of candidates is returned. For the HNSW graph, the authors set the parameter ef_{higher} equal to 1 on all levels above the bottom level to limit the extensiveness of the search on the higher levels. This comes back to the two-phase search approach of finding entry points for the bottom level; however, we evaluate the effect of larger ef_{higher} on the search performance as well.

Treating the level graph builder and the level subset selector respectively as black box algorithms allows us to keep the construction process of the HSGF independent of the individual construction of the level graphs. Consequently, this enables future extensions, which is why we see our graph structure as a framework to create hierarchy graphs. Additionally, this opens another dimension of parameters as now each level's graph builder, as well as each level's point selector, becomes a parameter, next to the already existing (hyper-)parameters of the individual graph builders themselves. As a result, the HSGF offers more parameters compared to existing ANNS graphs, thus allowing a more fine-granular approach to tune the graphs' performance towards desired targets.

Algorithm 2 HSGFConstruction

Input data D, level graph builders B, level point selectors S, level subset sizes LS, number of layers nl
Output HSGF graph

```
1: Graphs ← {}                                   ▷ the list of constructed graphs
2: for i = 0 to nl do                            ▷ build each level from the bottom up
3:     LevelBuilder ← builder for level i in B
4:     graph ← LevelBuilder(D)                   ▷ build level graph
5:     LevelSelector ← point selector for level i in S
6:     D ← LevelSelector(graph, LS[i])           ▷ select subset for next level
7:     Graphs ← Graphs ∪ graph
8: return G
```

Subset Selection Heuristics

While different, established ANNS graphs as potential level builders already exist, the subset selection heuristic is a new component of our search graph framework. We consider different types of approaches to select points. The first and naive approach is to select the subset randomly. However, we believe determining the next subset based on the current level's constructed graph is a more promising approach. Our reasoning here is that the points on the higher levels are supposed to ultimately improve the routing and ensure reachability in the HSGF graph. A simple approach to benefit from a constructed graph would be to select hub nodes in the graph, although this makes only sense on graphs with

a varying out-degree, as it otherwise performs a random selection.
The goal of our hierarchy design is to find good entry points to the bottom level. We argue that the reachability in the graph from the top level to all points on the bottom level is a key factor here. Our hierarchy design allows the selection of entry points to the bottom level so that points are reached that otherwise have no short routing between them. Correspondingly, the points from one level must be able to reach all points on the level below, which by design includes the challenge of reaching points that are not part of the current level. We therefore want to select a subset of points in which each point has edges to many other points on that level to increase the reachability. At the same time, we want to ensure that the number of representatives is minimal. Naturally, a point is a representative for itself and its neighbors.
Our strategy, therefore, involves selecting a point and marking its neighbors as already represented. We refer to this strategy as *flooding*, like the flooding used in networking routing, such that we select points and flood their neighbors so that these are marked and cannot be selected themselves anymore. By iterating over all points in the graph, this strategy is exhaustive, meaning that it ensures that there is a representative selected for each point in the graph. Additionally, the flooding strategy is still inherently random but opinionated in the sense that it deselects other potential candidates based on the selection of another candidate. In Algorithm3, we define our strategy with a parameter for the flooding distance, which sets the number of hops to which neighbors of neighbors will be flooded. A bigger flooding distance, accordingly, results in one point being representative for not just its neighbors but a significantly larger number of points. Simultaneously, a larger flooding distance results in a significantly smaller subset size. For a graph with a constant degree, a broad estimation, given that some neighbors of selected points will be already flooded over time depending on the graph's edges, is that the flooding selection reduces the set of points by a factor of about 1 over degree * flooding-distance.

In Chapter 5, we evaluate the effect that different subset selection strategies have on the search performance of different HSGF graphs. We focus on the random selection and the flooding selection strategy with different flooding distances, while additionally testing modified versions of these.

Algorithm 3 FloodingSelect

Input graph G, flooding distance fd
Output subset of points from G

```
1: S ← {}                                          ▷ set of selected points
2: F ← {}                                          ▷ set of flooded points
3: V ← random permutation of the points in G
4: for all v ∈ V do                                ▷ iterate over all points in the graph
5:     if v ∉ F then                               ▷ select if not yet marked as flooded
6:         S ← S ∪ {v}
7:         F ← F ∪ FloodNeighbors(G, v, fd)        ▷ flood v's neighbors
8: return S
```

Algorithm 4 FloodNeighbors

Input graph G, entry point p, flooding distance fd
Output set of flooded points

```
1: F ← {}                                          ▷ set of flooded points
2: if fd == 0 then
3:     return F
4: N ← N(G, p)
5: for all n ∈ N do
6:     F ← F ∪ {n} ∪ FloodNeighbors(G, n, fd - 1)   ▷ recursively flood n's neighbors
7: return F
```

Additional Parameters

So far, we have assumed that the number of levels is simply a parameter. However, given the findings in [35], we believe there is an optimal range for the number of levels and the number of points on each level. We try to additionally approximate this range by deriving the right number of levels and the levels' subset sizes from the number of points in the training data and the expected out-degree on each level. Unfortunately, the exact out-degree can not be specified on each graph builder. We therefore add the expected degree, the range of minimum and maximum number of levels, and the direct configuration of the level subset sizes as (optional) parameters.

4.3 Implementation

In this section, we describe important implementation details of our newly proposed HSGF. We aim to provide a library that contains an efficient implementation of HSGF, which enables the easy configuration of its parameters as well as the parameters of the level graph builders themselves.[1]
We implement HSGF in the programming language Rust[2] because it is a low-level language that allows us to develop efficient implementations for ANNS graph construction algorithms. Among Rust's advantages is its ownership model, which makes it highly memory-safe and memory-efficient. Another key difference of Rust from many other languages, like C++, is that it uses traits instead of a "classical" object-oriented approach to define shared behavior, functionality, and relations. Additionally, the supervisor of this work provides us with two Rust libraries, also called crates in Rust, for graph-based ANNS indices.[3] Given the dominance of Python in the machine learning community, we additionally provide bindings in the form of a Python module to make the HSGF easy to use.

[1] We make our code base available as part of the artifact package for this work. However, at the time of writing, the code base is not publicly available (e.g., via GitHub), but we plan to make the code base publicly available in the future.
[2] https://www.rust-lang.org/
[3] The two provided libraries are currently unpublished.

4.3.1 Project structure and Framework

We set our HSGF project up as a Rust library crate[4]. We use the 2021 edition of Rust, PyO3[5] in version 23.4 to generate the Python bindings, and among other crates, the rayon[6] crate, which is used for parallelization. The two libraries, which are provided by the supervisor, are at the time of writing unpublished - we will refer to them as GraphIndexAPI and GraphIndexBaselines, respectively. The GraphIndexAPI library defines fundamental functionality needed for graph-based ANNS indices, like data structures, distance functions, and an efficient implementation of the greedy search, with a focus on using generic types where possible. Specifically, it already contains functionality for hierarchy graphs. The GraphIndexBaselines library uses the GraphIndexAPI library to implement different graph builders like HNSW and RNN-Descent. We follow the design choices and patterns of GraphIndexBaselines when implementing HSGF, as well as the graph builders of DEG, EFANNA, and NSSG. The HSGF library can therefore be viewed as an extension to the GraphIndexBaselines library. We use our implementations for the DEG, EFANNA, and NSSG graph builders as well as the implementations of HNSW and RNN-Descent, which are already available in GraphIndexBaselines, to evaluate the search performance of different HSGF graph configurations.

4.3.2 Reimplementation of DEG, EFANNA and NSSG

We reimplement the DEG[7], EFANNA[8], and NSSG[9] graph builders in the Rust language and the framework that is the GraphIndexAPI based on their respective original implementations. All three graphs have been implemented in the language C++ and are optimized for a parallel setting. While the code of EFANNA and NSSG share some similarities, all three code bases use, among other things, different data structures for the representation of their graphs and correspondingly different implementations of the greedy search. Translating the code bases to the traits, types, and design patterns of the GraphIndexAPI and GraphIndexBaselines library removes those differences, which enables a fairer evaluation when constructing and querying the different graphs.

We implement a single-threaded version and a version that is further optimized for a parallel setting in our framework for each graph builder. We primarily translate the original parallel-focused optimizations into our framework but do not develop a *new* parallel implementation. After all, the HSGF is designed as a framework in which the level graphs are treated as a black box, such that from the perspective of the HSGF, it has no influence on the construction algorithm itself and how optimized it is for a parallel setting. Therefore, the performance gains of an HSGF in a parallel setting are almost completely dependent on its black boxes for the level graphs and subset selectors.

[4] A more detailed overview can be found in the *READMEs* of the artifact package of this work.
[5] https://pyo3.rs/v0.23.4/
[6] https://docs.rs/rayon/latest/rayon/
[7] https://github.com/Visual-Computing/DynamicExplorationGraph
[8] https://github.com/ZJULearning/efanna_graph
[9] https://github.com/ZJULearning/SSG - Note that they call their implementation of the NSSG just SSG

During development, we test and compare different data structures that are used by the respective builders during construction and select the best-performing data structures. Each single-threaded version is, besides the removed parallelization, as close to the original code as possible, while the parallel version contains more optimizations that are partly specific to Rust and the GraphIndexAPI framework. Next, we will briefly go over important changes we made to the original implementation of each graph builder.

The construction algorithm for EFANNA uses a temporary graph, which is a vector of neighborhood objects. We implement the neighborhood structure with a max heap for the pool of candidate points while also testing a plain vector for the pool and only sorting it on demand. We observe that the max heap version, which is also most similar to the original implementation (they use a vector but sort it as a max heap), performs on average slightly faster than the plain vector version. Given EFANNA's constant out-degree, we can initialize the final graph exactly at once, which allows us to write to it in parallel when selecting the neighbors for each node from the temporary graph.

Like EFANNA, NSSG also uses a temporary graph, which for NSSG is a one-dimensional vector containing simple objects representing a (neighboring) point. We therefore use a list of lists, which more closely resembles the final graph data structure and is in line with our framework. We add the input graph, which NSSG refines as an optional parameter, and otherwise initialize the input graph randomly. By default, we use EFANNA as an input graph, because it is the recommended choice by the authors.

The construction algorithm of DEG does not involve a temporary graph structure and therefore uses significantly less memory compared to EFANNA and NSSG. Compared to the original code base, we initialize the graph with all vertices, which in the DEG have a constant out-degree, at once. This also allows us for the DEG to remove the locking during construction, which results in a significantly shorter construction time without impacting the graph quality. A reason why this is not part of the original implementation might be that they focus on a dynamic graph where edges can constantly be removed or added, and the size of the training data is therefore not fixed or known beforehand. However, we do not implement the dynamic capabilities of DEG because it is beyond the focus of this work. A major difference from the original implementation is that we use the greedy search instead of the range search during construction. While testing during the implementation phase, we observe that our range search implementation is significantly slower compared to the greedy search's implementation, at otherwise similar recall levels. This applies in the same way to the *hasPath* function, which checks if a path from a set of entry points to a query point exists. Our implementation of the optional edge optimization process seems to be a major factor in the construction time. However, given that it is optional and our observation that the search performance gains in terms of recall and QPS are marginal, we turn the optimization process off for the construction of the DEG during our evaluation. We investigated this issue further but could not find a clear reason, besides observing that the optimization process could find almost "endlessly" edges to improve, which resulted in the optimization process continuing without reaching a stopping condition in a reasonable time given a corresponding parameter configuration (see DEG's parameter *additional_swap_tries*).

4.3.3 HSGF

We treat the level graph building and subset selection as black box algorithms. However, we exclusively want to use and evaluate HSGF configurations with established graphs on their levels. Given Rust's strict type system, we implement two HSGF builders, which differ in the input argument for the level builders. The first one uses an enum to exactly specify the available graph builders, which in our case are DEG, EFANNA, NSSG, and a random graph, as well as RNN-Descent and a brute-force KNN graph from the GraphIndexBaselines library. The enum offers a factory method that takes in a parameter object for the respective graph builder and returns a valid HSGF enum level builder. We zip each level builder with the point selector for that level, or rather, the next level's subset. We define a SubsetSelector trait, which defines one function to select a subset from an input graph. The SubsetSelector trait allows us to specify the type for the level selector as dynamic by wrapping it in a *Box* and using the *dyn* keyword. Due to Rust's restrictions on dynamic compatibility, we can not use the same approach for the level graph builders, which is why we decided on the enum approach. Our second HSGF builder offers more flexibility by using Rust's closures (read lambda functions). We need to split the argument for the bottom-level builder and higher-level builders up into two arguments because of limitations with the GraphIndexAPI framework and Rust itself; however, this does not impact the performance. Given the range of already described parameters, the level subset sizes and minimum as well as maximum number of levels, we implement our HSGF builders so that the list of tuples of level builder and selector is exhausted sequentially. If the defined number of levels is bigger than the list of tuples, we use the last element in the list. During our evaluation, we exclusively use the HSGF enum builder.

The implementation of the random selection approach is straightforward. For an efficient parallel implementation of the flooding approach, we first create a random permutation of all currently available points, which adds the random component to the flooding strategy. We can then iterate in parallel over the points and check via a bit set if a point has been selected or flooded already. If the points are not contained in that set, the set is added, and the point's neighbors are recursively added to the bit-set up to the flooding distance deep. Once the permutation is completely iterated over, we additionally double-check the selected points for duplicates.
We conjecture that the effect of "errors", which can appear in a parallel setting, is negligible. Firstly, we believe the appearance of cases where a point is selected in one thread before or while being marked as flooded in another thread is limited and can be mitigated by our random iteration over the points. Secondly, we do not expect a negative effect from points that are fewer hops away from other points than specified by the flooding distance because this does not impact the targeted reachability.

5 Evaluation

We design the HSGF based on the hierarchy architecture of HNSW. We aim to replicate the apparent effects that the hierarchy and the corresponding redundancy have on the search performance. The two primary metrics in this space are recall and queries-per-second, which are commonly evaluated together as QPS over recall. This is because changes to a graph's parameters generally affect recall and QPS in opposite directions. In addition, our second focus is to achieve shorter construction times with our HSGF graphs than an HNSW graph has at similar search performance levels.

5.1 Methodology

We evaluate the construction of an ANNS graph in a parallel setting, but execute the search on a single thread for a better comparison of the QPS to the results of previous works. The hardware setup, the used framework, including the search function (e.g., range or greedy search), as well as the individual implementation of the same algorithm (even in the same language), have a significant influence on the search performance and construction time of an ANNS graph. Therefore, only the results from the same setup are truly comparable, although relative comparisons are still possible. Regardless, we expect the recall to be less affected than the QPS by the underlying setup because we argue that it is mainly dependent on the algorithm itself, compared to the implementation or other factors. For our work, the aim is not necessarily to advance the state-of-the-art search performance but to evaluate the effects a hierarchy design with redundancy has on the search performance. However, in the case of positive findings, this could result in the improvement of the performance over existing non-hierarchical ANNS graphs and consequently in state-of-the-art search performance.
We use the greedy search and use a range of different combinations in the search parameters, of which we focus exclusively on different values for the number of search results k and the size of the extended neighbors list ef. Additionally, for hierarchy graphs we differentiate between ef_{bottom} used on the bottom level and ef_{higher} for all higher levels.

The established ANNS graphs in our evaluation involve the following and already presented graphs construction algorithms: DEG, EFANNA, NSSG, RNN-Descent (further referred to as RNN), and HNSW. We use these graph builders, which are all implemented within our framework of the GraphIndexAPI, as level builders in our HSGF graphs. The exception to this is HNSW, which serves as our primary object of comparison.
Using the same framework for the implementation and the same search regime for all our experiments creates a fair setting for comparisons. By querying a given graph multiple times with a fixed k but a range of different $ef_{bottom} \geq k$, which are multitudes of k, we generate a QPS over recall (search performance) curve for the respective dataset, k, and

graph as well as respective ef_{higher} for hierarchy graphs. A comparison between an HSGF and an HNSW graph is established on the same dataset, and after tuning both graphs' parameters to the same recall range. We target a recall range for all experiments of 90-95% at $ef{=}k$, depending on the dataset. Although in certain cases, like for example for very large datasets and search result sizes of $k < 20$, the smallest measured recall might be lower than 90%. However, by querying the graph with increasing values for ef_{bottom}, starting at $ef{=}k$, the graphs' search performance almost always approaches the 99-100% range of recall.

As a first step, we evaluate each of these five graphs individually on standard datasets which were used in previous works to establish their performance characteristics given our setup, allowing for comparisons to previous works as well as validating our implementations.
The primary object of comparison for each HSGF graph is its respective graph on the bottom level, which is built on the complete training data and is only affected by its own parameters, but not by any parameter of the overall HSGF. Accordingly, it is in theory the "same" graph that would be constructed outside of the HSGF. As a result of the hierarchy and redundancy, we can select different level-stacked subgraphs (see Hierarchy Graphs in Preliminaries 2) from the HSGF graph, which are still complete in terms of the training data. This can, therefore, be similarly applied to an HNSW graph. We evaluate the search performance of an HSGF or HNSW graph by searching each level-stacked subgraph with the same set of queries and search parameters. The hierarchy effects for HNSW graphs have, to our knowledge, not been evaluated in this manner yet.

The GraphIndexAPI framework, together with our implementation of the HSGF graph and the implementation of HNSW in GraphIndexBaselines, allows us to easily extract level-stacked subgraphs from a given hierarchy graph. We could run the same search multiple times to control for potential deviations in the search process due to the random selection of the entry point(s) for the currently evaluated highest level. Instead, we modify the construction process and point selection of the HSGF and HNSW graph so that the first point of the training data is always contained on each level, and we always use this *point-zero* as the (first) entry point for the search starting from the top level of the respective subgraph. We can now compare all level-stacked subgraphs to the baselines, that is, the subgraph corresponding to only the bottom level. Positive effects of the hierarchy would then show as higher QPS and/or recall values for each taller stacked subgraph. Specifically, the search performance of the higher levels would always lie above the curve, which is spanned by the search performance of the bottom level for an increasing number of ef. However, this is under the assumption that the additional overhead for constructing a respective HSGF, which in our case is measured exclusively as the additional construction time, is smaller than constructing a bottom-level graph with parameters tuned for higher recall ranges.

Nonetheless, we still control for potential deviations of the search queries, which in our setup only come from the underlying hardware, by querying with a multitude of search parameter configurations and, more importantly, by querying each subgraph individually with each of those configurations. For example, the ef_{higher} does not affect the search of the bottom-level subgraph, so that we automatically query the bottom-level graph with the

same setup multiple times. We find minimal to negligible deviations between the results for the bottom level of the same configuration, which suggests that our results are stable.

The overarching object of comparison for us is the HNSW graph. In the ideal case we can build an HSGF graph in less time, which achieves higher QPS and higher recall levels over a range of *ef* values. However, we would already consider improving either QPS or recall, while the value for the other one stays on a comparable level, a success. Albeit with the restriction that the performance of the complete HSGF graph is better than its ”pure” bottom-level subgraph.
For the second objective of our evaluation, the construction time, we define that we have a positive result if the construction time of an HSGF graph is shorter than that of an HNSW graph at comparable search performance. As a result of our hierarchy design and sequential bottom-up construction process, a positive finding for this objective can only be expected for graph builders that have a significantly faster construction time than an HNSW graph. The second objective of our evaluation is therefore (one-directionally) dependent on positive results from the first objective.

All our experiments were run on a machine with an Intel(R) Xeon(R) W-2155 CPU @ 3.30GHz and 64 GB of RAM.[1].

Datasets

We used the following datasets, which have been established by previous works [7, 25], for our evaluation: AUDIO [34], DEEP-1M [3], ENRON [37], GLOVE-100 [42], SIFT [29], and a three-million as well as a ten-million subset of the LAION-5B [45] (which we will refer to as LAION-3M and LAION-10M, respectively). However, we focus primarily on the subsets of the LAION-5B, which consists of 768-dimensional vectors and has a set of 10,000 query points.

Each dataset consists of the training data, the set of queries, and the ground truth nearest neighbors corresponding to the set of queries. For each search configuration, we search the current index sequentially with each query in the respective set of queries and use the ground truth to calculate the average recall as defined in Chapter 2. The QPS is calculated by dividing the number of points in the set of queries by the time (in seconds) needed to run the search for all queries in the set.

Implementation, Parameter Configurations and Artifact Package

We run all experiments via the Python binding of our Rust HSGF crate. The result of each experiment is saved as a CSV file where the columns include, among others, the recall, QPS, construction time, search parameters, and the exact parameter configuration for the corresponding graph. The parameter information for the HSGF graph also includes the parameter configuration of each level graph and level subset selector, which supports the

[1]We do not explicitly log the thread count, but once we observe a number of lightweight processes (NLWP) of 40.

reproducibility of our experiments.
Furthermore, we also provide, next to the plots in this thesis document, additional plots in the form of .html files, which are exported via the Python library Plotly[2] as part of our artifact package. The HTML plots offer additional features to inspect the data in more detail, like zooming or tooltips. Additionally, we provide a Jupyter Notebook, which offers functionality to easily plot specific views of the raw data from our experiments.

5.2 Base Graphs

The evaluation of the graphs used in this work is important to create a baseline for comparisons. Although this kind of evaluation has been done before, the literature has generally focused on a single-threaded setting due to previously discussed reasons. Therefore, this evaluation offers a comparatively new view on the construction times of the different graph builders in a parallel setting. We expect the recall of each graph to be dependent on its parameters and mostly independent of its implementation, and therefore expect to find similar recall levels as in previous works. The QPS, however, is more dependent on the hardware setup.

We evaluate each graph on each dataset by first constructing the graph and then querying the graph and calculating the recall and QPS for a range of *ef* values, starting from the search size k. We plot the QPS over recall for all graphs in one plot for each dataset, with each point representing a different *ef* value. In the description of each plot, we mention the values of *ef* which were used for that dataset. The plots for the DEEP-1M, GLOVE-100, SIFT, and LAION-3M datasets can be seen in Figure 5.1. We want to point out that the recall range on the x-axis for all plots in this work is not fitted to be the same range due to deviations in the search performance for different datasets and search configurations.
Additionally, the construction time for each graph and dataset is shown in Figure 5.2.

While we try to tune the parameters to the best of our abilities so that the graphs are fairly compared, we are aware that our final parameters for each graph might not be the optimal representation for their maximum achievable performance in terms of our three main metrics. A major reason is the extensive resources (e.g., computational time) it takes to tune the parameters of ANNS graphs. At the same time, we want to avoid over-fitting the parameters. However, we believe that our results are still representative of the performance of the respective builder, especially in comparison to each other.

An observation we make during our parameter tuning phase is that for $ef{=}k$, certain graphs are easier to fit to high recall ranges than others. The primary example here is RNN, which achieves competitive recall and QPS ranges for large *ef* but generally achieves only comparatively low recall ranges for $ef{=}k$ depending on the dataset (e.g., AUDIO, ENRON, DEEP-1M, SIFT).
In the evaluation of our implementations, we do not reach similarly high levels in QPS over recall as seen, for example, in other works (see e.g., Figure 4 in [25][3]); however, we still

[2] https://plotly.com/

[3] They use a "Ryzen 2700x CPU, operating at a constant core clock speed of 4GHz, and 64GB of DDR4 memory running at 2133MHz".

find reasonably high-performance levels. Firstly, we do not use exactly the same graph parameter configurations as in [25], as, for example, we do not initialize EFANNA with *kdtrees* for the AUDIO and ENRON datasets. Additionally, they use for HNSW significantly larger $ef_{construction}$ than we need to specify for the HNSW implementation of the GraphIndexBaselines library to reach high recall ranges. We argue that the differences in terms of QPS stem primarily from the underlying hardware[4], so that we measure lower QPS at similar recall levels. Nonetheless, the deviations do not impact the validity of our results.

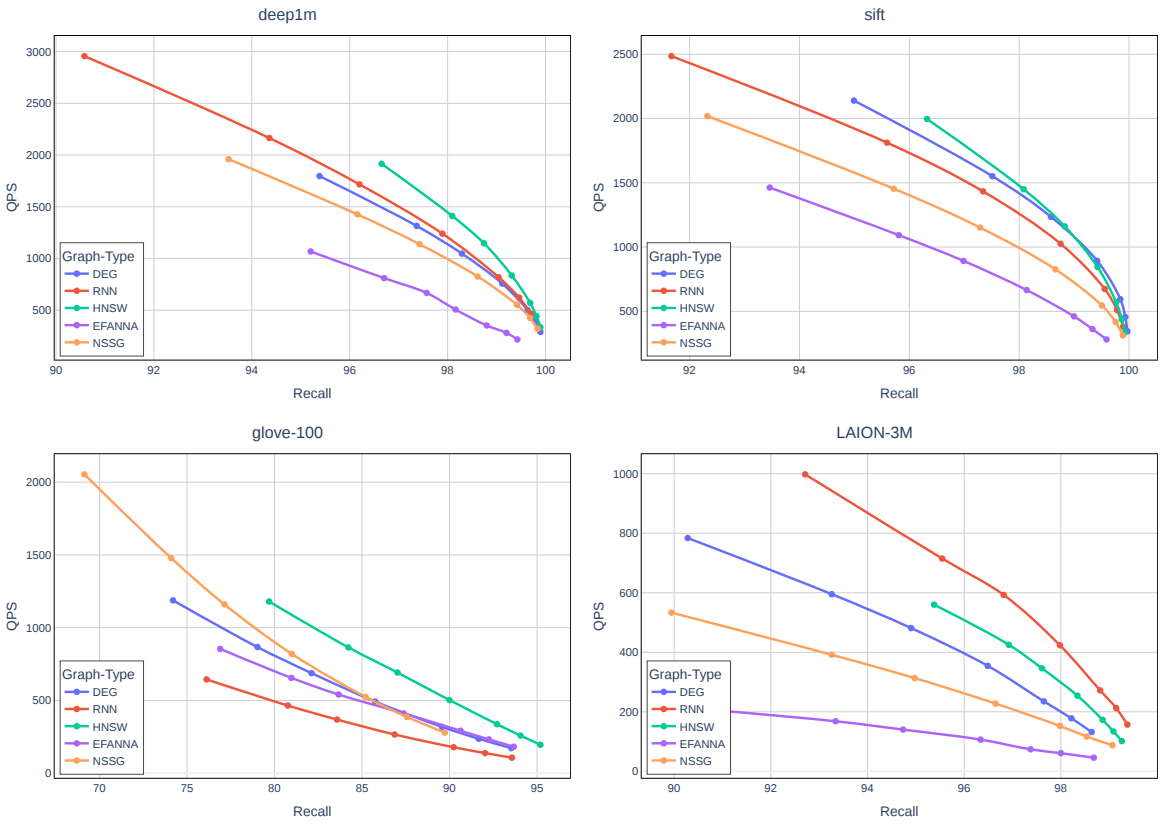

Figure 5.1: QPS over recall for four different datasets and the base graphs used in this work. To the top right is better. The search configuration for all four datasets is the following: k=100 and ef_{bottom}=[100, 150, 200, 300, 500, 700, 1000]. The plots for the Audio and Enron datasets can be found in the appendix.

On all datasets, the construction time of DEG is, besides a minimal gap to RNN on ENRON, the shortest, while EFANNA and NSSG are significantly slower than all other graphs. NSSG uses an EFANNA graph as its input graph, which is constructed with the same parameters as for its own, separate construction. The construction time of the NSSG includes, therefore, the comparatively long construction time of EFANNA; however, the added construction time of the NSSG is relatively short (see Figure 5.2). The particularly short construction times for DEG indicate that it strongly benefits from a parallel setting, even though the results from a study in a parallel setting from the corresponding paper did not suggest that

[4]We conjecture that the higher clock speed could be the main reason.

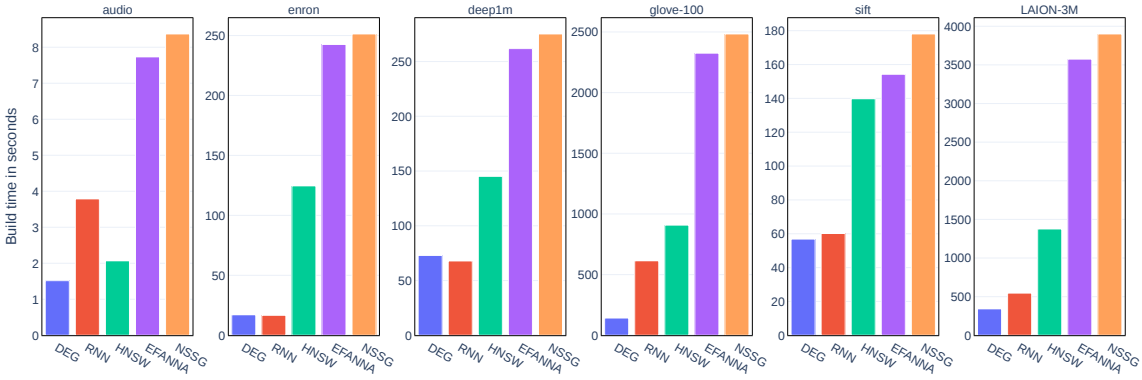

Figure 5.2: Build times for different graphs per dataset.

the DEG graph can be constructed quickly [25]. Additionally, tuning the parameters of the DEG, such as the out-degree and the $ef_{construction}$, to higher values likely results in better search performance at potentially still shorter construction time. Furthermore, on the GLOVE-100 dataset, the DEG significantly outperforms the other graphs in terms of construction time at comparable search performance, which suggests that it performs particularly well on datasets with a high LID (such as GLOVE-100). HNSW shows the best search performance on all datasets, except the LAION-3M, but also has a longer construction time than DEG or RNN, which underscores our work's focus on HNSW. We acknowledge the risk of over-fitting the graph parameter configurations to our work's assumptions about HNSW; however, our assumptions are based on previous research, which renders our risk minimal [40, 25, 7].

In our evaluation, we exclusively use the EFANNA builder as the input graph to NSSG - we use the same parameters for EFANNA when evaluated on its own and when used as an input graph to NSSG for a fair comparison. We use EFANNA because it is the recommended choice by the corresponding authors and allows us to easily create a graph with a large degree. During the implementation stage, we test a random initialization and the DEG or RNN graph as an input graph. However, we find extremely low performance for the random initialization, and in the other two cases, only marginal search performance gains over the DEG or RNN graph itself, which suggests that the NSSG graph is too dependent on the DEG or RNN as an input graph. This, however, would render the NSSG obsolete, especially given its additional construction time. In Figure 5.1, the performance curve for the NSSG graph always lies above the curve for the EFANNA graph on all datasets, except for the higher recall ranges on the GLOVE dataset. This indicates the positive gains of NSSG's pruning strategy over its input graph, which is an EFANNA graph of the same parameters.
Moreover, for the AUDIO and ENRON datasets, we observe atypical performance curves for the EFANNA and NSSG graph, respectively. After a short investigation, we cannot find the root cause but hypothesize that the search might have gotten stuck in a local minimum. Given the comparatively small dataset sizes for the AUDIO and ENRON datasets and otherwise typical performance curves of both graphs on the other four datasets used in this work, we do not investigate the issue in more depth.

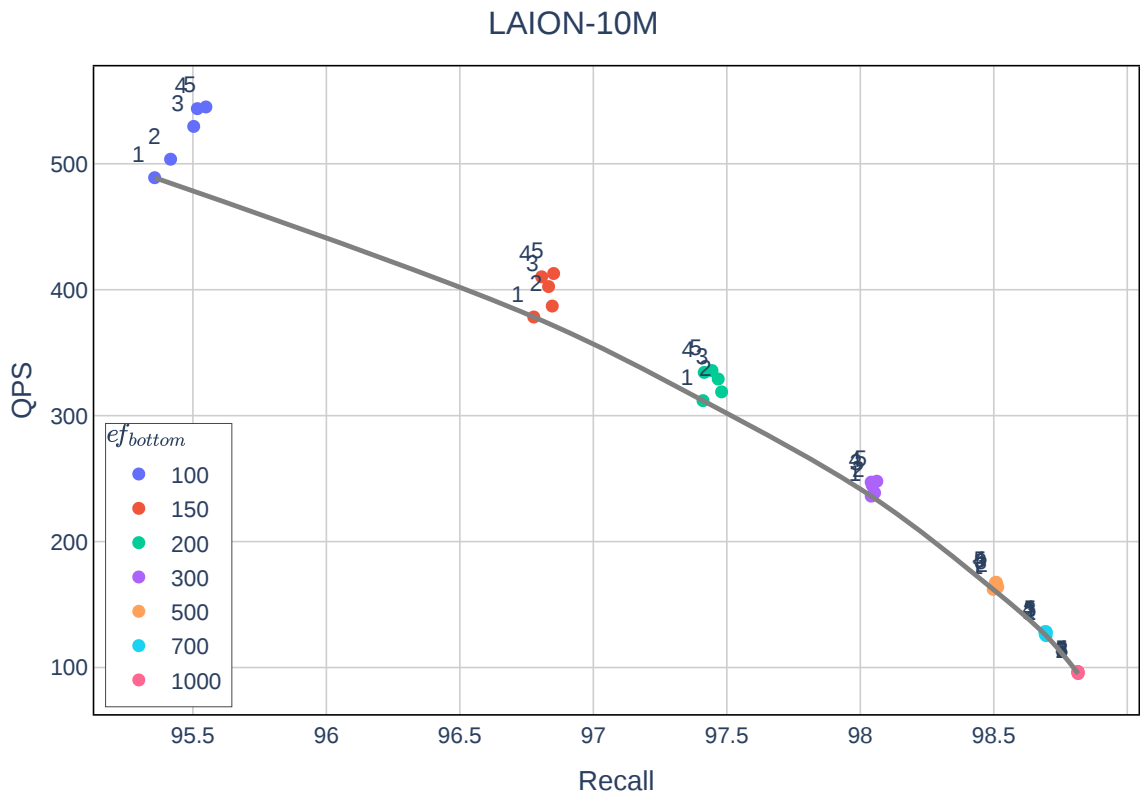

Figure 5.3: QPS over Recall for HNSW on the LAION-10M dataset with k=100 and ef_{higher}=1 for multiple ef_{bottom}=[100, 150, 200, 300, 500, 700, 1000]. The number next to each point refers to the size of the level-stacked subgraph.

5.3 HNSW

We evaluate the hierarchy effects of the HNSW graph on its search performance, as described in the Methodology section, by querying each level-stacked subgraph with the same search configuration. We focus on the high-dimensional and modern LAION dataset, compared to most of the other datasets used in this work, in the form of the 10M subset, as it is comparatively large. The used search configurations are combinations from ranges of result sizes k, ef_{bottom}, and ef_{higher}. Nonetheless, we evaluate the level-wise search performance also for the other datasets, albeit only for the k and different ef_{bottom} which were used during the base graph evaluation and an ef_{higher}=1, which is the default for HNSW. Visualizing the data point for each performance result in one plot is not reasonably possible; we therefore select certain search configurations as shown, for example, in Figure 5.3 for k=100.

We observe a positive hierarchy effect in terms of both recall and QPS for the HNSW graph as seen in the increasing curve of the search performance for each level-stacked subgraph for a respective ef_{bottom}. However, the effect decreases for an increasing ef_{bottom} and is almost negligible for very large ef_{bottom}, which naturally corresponds to high recall ranges, while the relative gains from the top level over its bottom level decrease faster

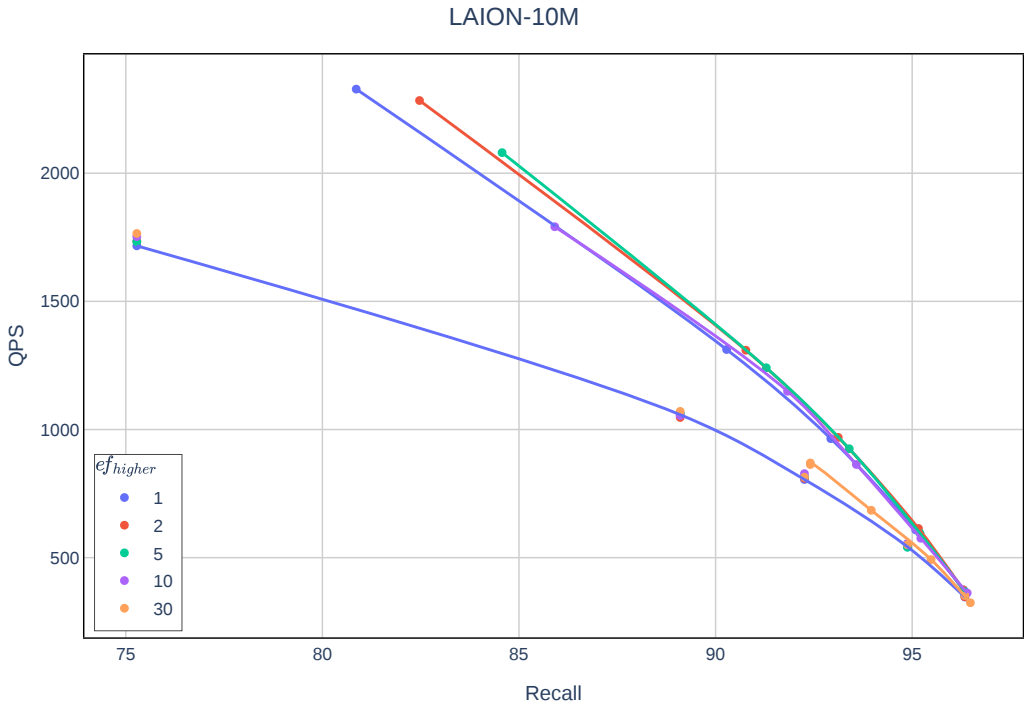

Figure 5.4: QPS over Recall for HNSW on the LAION-10M dataset with k=1, for ef_{higher}=[1, 2, 5, 30] and multiple ef_{bottom}=[10, 30, 50, 100, 200]. Note, the vertically clustered points on the (blue) bottom level line are the result of minor deviations in the (computer-)system's performance, and therefore only impact the QPS but not the recall. Therefore, we only plot the bottom level line for ef_{higher}=1. See in the Appendix also Figure 10 and Figure 11 for k=10 and k=100, respectively.

for the recall than for the QPS. Figure 5.4 shows the hierarchy effects for HNSW on the LAION-10M dataset for different ef_{higher} at k=1. We find that an ef_{higher} up to 10 increases the recall further, and although the QPS decreases slightly correspondingly, the search performance is better than for ef_{higher}=1, which is the original default choice of the HNSW. However, for larger ef_{bottom}, the additional gains fade out, while for k=10 and k=100 they are smaller and almost negligible for all tested ef_{bottom}. This suggest that, besides for k=1 and smaller ef_{bottom}, it is generally better to increase ef_{bottom} as a search parameter for better performance than increasing ef_{higher} beyond 1, although it potentially allows for a more fine-grained tuning of the target recall range.

We find that for k=1 and relatively small ef_{bottom}, the relative hierarchy effects are the strongest while disappearing for larger k and larger ef_{bottom} on the LAION-10M dataset. However, the hierarchy effect for HNSW on the other datasets is not showing that strongly. For example, on the SIFT dataset, we only observe positive gains in terms of QPS but not recall. On the DEEP-1M and GLOVE-100 datasets, we find no positive hierarchy effects. Interestingly, though, on the GLOVE-100 dataset, the second level shows slightly worse in terms of recall than the other subgraphs. We expect this to be because of worse connectivity for that particular level. See Figure 3 in the appendix for the other datasets.
Given the already high recall ranges that HNSW reaches for these datasets, we expect at

ef_{bottom}	HNSW - LAION-3M				HNSW - LAION-10M			
	Recall		QPS		Recall		QPS	
	Gain	Bottom-Level	Gain	Bottom-Level	Gain	Bottom-Level	Gain	Bottom-Level
10	+6.67%	78.63%	+26.99%	2046.22	+7.37%	75.69%	+35.62%	1696.29
30	+1.46%	91.94%	+18.57%	1267.35	+1.15%	90.28%	+24.2%	1036.84
50	+0.8%	94.82%	+13.46%	968.9	+0.54%	93.57%	+19.35%	795.49
100	+0.34%	97.1%	+9.55%	640.54	+0.22%	96.06%	+11.84%	532.3
200	+0.09%	98.39%	+5.68%	405.68	+0.04%	97.34%	+8.73%	334.87

Table 5.1: Hierarchy effects for different HNSW graphs on the LAION 3M and 10M dataset for k=10 and ef_{higher}=1 for different ef_{bottom}. The gains are of the top level's recall/QPS over the bottom level's recall/QPS for the respective ef_{bottom}. The level sizes for the HNSW on the LAION-3M dataset are: [3.000.000, 99.912, 3257, 128, 2] and for the HNSW on the LAION-10M dataset are: [10.120.191, 337.332, 11.256, 353, 16].

best only parameter configurations for HNSW that result in lower recall ranges to show a (stronger) hierarchy effect. It might be more likely that we observe stronger hierarchy effects for smaller k, given our findings for the LAION-10 datasets. We briefly tested this by evaluating HNSW on the DEEP-1M dataset for k=10 and k=1 but only find similar effects, which fade out for larger ef_{bottom} (see Figure 4 in the appendix; note that the recall in these plots starts from a smaller range).

Additionally, we apply the same procedure to three variants of HNSW, differentiated by their configured out-degrees and $ef_{construction}$ into a low, a medium, and a high-performance tuned variant, on the LAION-3M subset. Similarly to the LAION-10 dataset, we observe that the hierarchy effect shows stronger for small ranges of ef_{bottom}. Furthermore, we observe no major impact of a smaller or higher configured degree for HNSW on the hierarchy effects itself, but naturally see an improvement in its overall performance (see Figure 5 in the appendix). In Table 5.1, the relative gains for the medium-tuned HNSW graph on the LAION-3M dataset and an HNSW graph on the LAION-10M dataset for k=10 are shown. Additional Tables for k=1 and k=100 can be found in the appendix (see Table 1 and Table 3)

To conclude, we empirically confirm the positive effect of the hierarchy architecture alongside the corresponding redundancy of the HNSW graph on the search performance in terms of recall and QPS. Additionally, we show that the effect is more strongly observed with the QPS, and decreases in high parameter configurations of the graph and the search function. We observe that an ef_{higher}=1 is most likely the optimal choice when searching the HNSW graph, although increasing it allows for a more fine-grained tuning of the search performance. However, we also observe that the HNSW sees no positive hierarchy effects on its search performance for certain datasets, like the GLOVE-100, which has a comparatively large LID.

5.4 HSGF

The evaluation of the HSGF follows a similar procedure to that for the HNSW by comparing the level-wise search performance for different HSGF graphs. We test a range of different HSGF graph configurations by combining the DEG, EFANNA, NSSG, and RNN graphs as level-graphs with the random or flooding subset selection, as well as more experimental

selection heuristics such as hub-nodes or flooding with additionally selected points for the levels of the HSGF. The significantly longer construction time for the EFANNA graph, and therefore also the NSSG graph, makes them unattractive choices for our evaluation setup, so we end up evaluating them in less depth and in fewer HSGF configurations compared to the DEG and RNN graphs.
The primary datasets for this evaluation are also here: the LAION-3M and LAION-10M datasets. Next to evaluating the hierarchy effect for different HSGF graphs, we specifically want to evaluate the influence of the point selection heuristics and the level graph on that effect. Furthermore, our second evaluation objective focuses on comparing the HSGF to the HNSW graph in terms of search performance and construction time.

We start our evaluation by using the same level graph on each level with the same parameters and the same level selection on the LAION-3M dataset by combining each of the four graphs with the random and flooding selection, respectively. In the next step, we compose more specific HSGF configurations based on these results. We primarily configure the flooding distance, level subset sizes, and the individual graph's parameters in terms of their out-degree. Finally, our most promising configurations are evaluated additionally on the LAION-10M dataset. To differentiate the different configurations, we introduce the following notation: HSGF-(<Level-Graph>-<Level-Selection>)∗-<Dataset>, which for one of our starting configurations would be for example *HSGF-DEG-Flooding=1-3M* (we name the graph builder and selection strategy only once if it is the same on all levels).

For each starting configuration, we use similar graph parameters for the level graphs to the ones we used for the base graphs' evaluation on the LAION-3M dataset, and either a flooding distance of one or, for the random selection, determine the level subset sizes similarly to the HNSW algorithm. The search is, like the evaluation of HNSW, executed for a range of configurations of k, ef_{bottom}, and ef_{higher}.

We find for each of these configurations positive hierarchy effects in terms of both recall and QPS, albeit with strong differences between the various HSGF-∗ graphs. A selection of graphs can be seen in Figure 5.5. Similar to our findings for the HNSW graph, we find that the hierarchy effect for the HSGF configurations is stronger for small ef_{bottom} and fades out for larger ef_{bottom}, which naturally also here corresponds to high recall ranges. In contrast to the findings for HNSW, we find that the HSGF does not see further improvement to its hierarchy effects from a $ef_{higher} > 1$. Figure 5.8 shows for a HSGF-DEG-Flooding=2-Flooding=1 on the LAION-10M dataset that the recall increases with a bigger ef_{higher} but is below the performance curve for ef_{higher}=1. This indicates that increasing ef_{higher} is a suboptimal choice for tuning the search performance of an HSGF graph than ef_{bottom}.

The differences in the hierarchy effect are both in terms of recall and QPS. Specifically, looking at the configuration of k=100 and ef_{higher}=1, we find significant hierarchy effects primarily for the HSGF-EFANNA graph, for which the gains for the random selection are larger than for the flooding approach. More and stronger positive hierarchy effects for the other HSGF configurations can be seen for smaller k. Focusing momentarily on k=10 and ef_{higher}=1, we find the HSGF-DEG graphs to show the strongest gains in recall and QPS for both the random and flooding selection, although with larger recall gains for the flooding selection and larger QPS gains for the random selection, respectively. The HSGF-RNN shows larger QPS gains for the random approach than the flooding selection, but

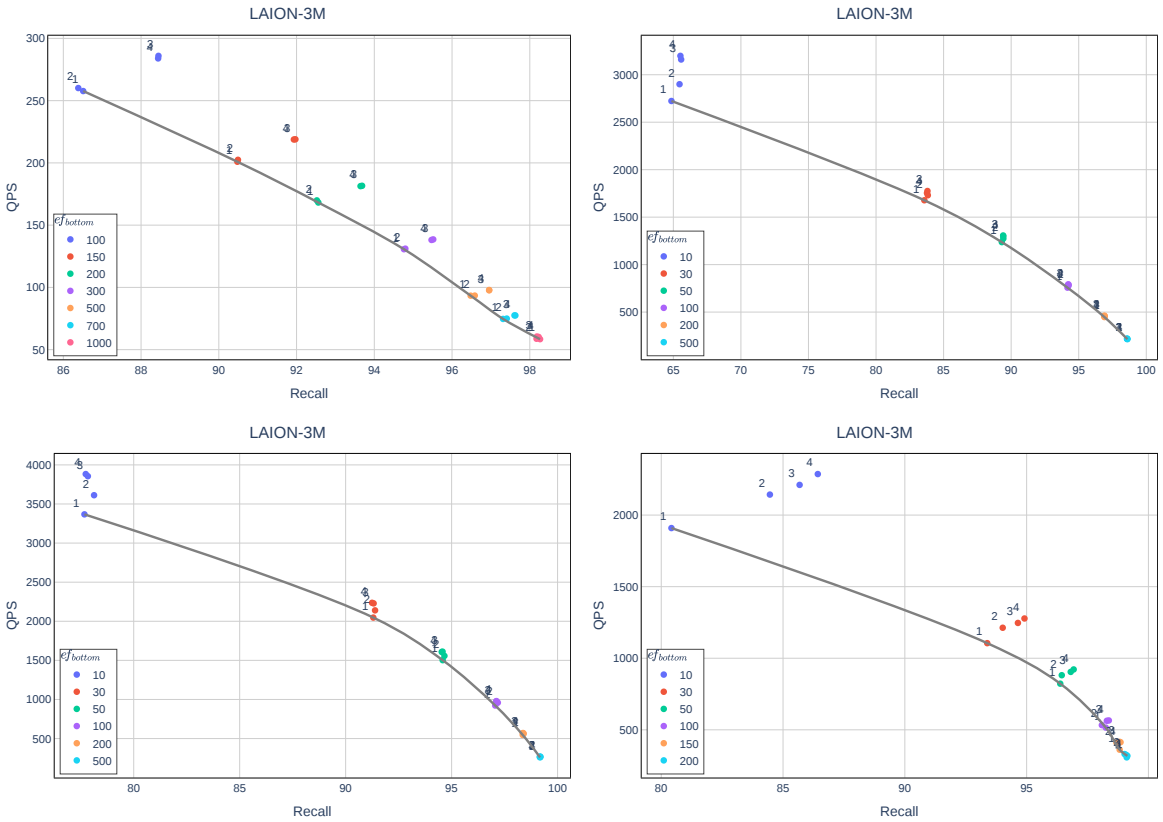

Figure 5.5: QPS over Recall for different HSGF configurations on the LAION-3M dataset for k=100 (HSGF-EFANNA) and otherwise k=10 for different ef_{bottom}. From top-left clockwise: HSGF-EFANNA-Random, HSGF-NSSG-Random, HSGF-DEG-Flooding=2-Flooding=1, HSGF-RNN-Random. Note the different ranges on the axes for both the recall and QPS in each plot.

otherwise shows only marginal to slightly negative effects on the recall. A similar situation can be observed for the NSSG graph. Furthermore, we find no benefit from the hub node selection approach or other modified versions of the random or flooding approach, and therefore drop these from further evaluation.

Focusing on the HSGF-DEG

Based on these results, we focus the remaining evaluation on different configurations of the HSGF-DEG and HSGF-RNN, meaning that at minimum, a DEG or RNN graph is on the bottom level. We test different combinations of different higher-level graphs, including EFANNA and NSSG, and observe negative effects for the HSGF-RNN-$*$ but no significant difference for HSGF-DEG-$*$ configurations, besides a negative impact when combined with the RNN (see Figure 8 in the appendix).
First, we observe that using the same graph type on all levels in an HSGF seems to work better, or at least more consistently, than combining different level graphs. Second, we make the observation that the hierarchy effect is more strongly appearing for graphs with a

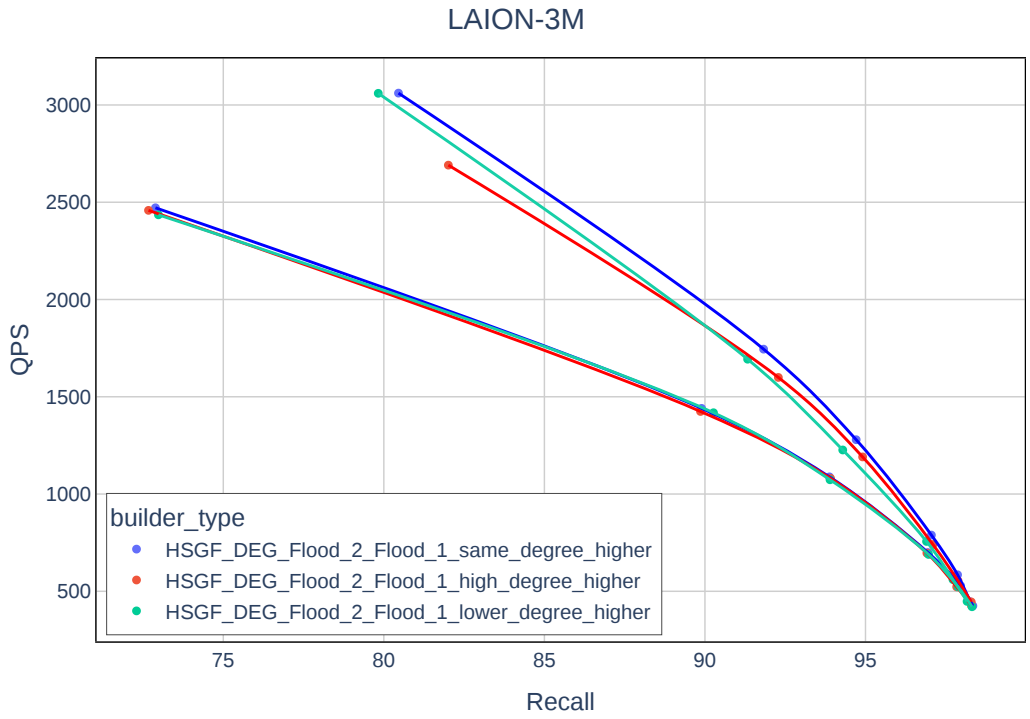

Figure 5.6: QPS over Recall for different HSGF-DEG-Flooding=2-Flooding=1 graphs differentiated only by their out-degree on the levels above the bottom level on the LAION-3M dataset for k=1, ef_{higher}=1 and for ef_{bottom}=[10, 30, 50, 100, 150, 200].

constant and relatively large average out-degree (>30), given the findings for (HSGF-)RNN and (HSGF-)NSSG, which are both graphs without a constant out-degree. Consequently, we focus primarily on configurations of the HSGF with a DEG graph on all levels. We therefore test HSGF-DEG graphs in more depth by configuring HSGF-DEG-$*$ graphs with different level subset sizes, flooding distances greater than 1, and differently configured parameters of the DEG graphs themselves. We focus next primarily on the LAION-3M dataset and a search configuration with k=10 or k=1 and ef_{higher}=1 because the hierarchy effects are showing in these search configurations the strongest.

On the LAION-3M dataset, the flooding selection with a flooding distance of *one* results for a DEG graph with an out-degree of 60 in roughly the following level subset size: 190,000, 12,000, 800, and 50, whereas with a flooding distance of two it results roughly in: 8000 and <30. The HSGF-DEG with the flooding distance of *two*, however, shows only limited gains over its two or maximum three levels, depending on the cutoff criteria for the minimum level size, compared to the HSGF-DEG-Flooding=1. We combine both approaches to find a suitable middle ground by selecting the first subset with a flooding distance of *two* and the remaining subsets on the higher levels with a flooding distance of *one*. We find similar recall for this approach compared to the HSGF-DEG-Flooding=1, while measuring slightly higher QPS. Matching the hierarchy effects for significantly smaller subset sizes, consequently, leads to shorter construction times and additionally results in a smaller memory footprint.

Given our previous results, which showed mostly small differences between a random and flooding selection approach for the HSGF-DEG, we compare the two-stage flooding approach to a random selection. For the random selection, we specify the level subset sizes to roughly match those of the two-stage flooding approach. Interestingly, we find basically no differences in the performance gains between the two selection approaches for the LAION-3M dataset. Our impression during testing, however, is that the flooding approach generally seems to be more stable, but we are unable to empirically confirm this.

Additionally, we compare different numbers of levels and therefore also different level sizes as well as different parameter configurations for the DEG graphs themselves, for example, using a graph with a smaller or larger degree and smaller or larger $ef_{construction}$.
In Figure 5.6, three different HSGF-DEG-Flooding=2-Flooding=1 graphs are compared based on their respective out-degree for the levels above the bottom level. We find that the HSGF graph with the same degree on all levels, and thus the same graph parameters on all levels, performs the best. Although a larger out-degree on higher levels increases the recall further, it also decreases the QPS such that the performance is below the curve for the same degree graph. It is worth mentioning that the construction differences between these three HSGF graphs are marginal (< 2%) because of the small subset sizes on the higher levels. The flooding selection naturally selects fewer (or in other words deselects more) points for a graph with a higher degree, which additionally reduces the subset sizes.

We observe in a few cases that the hierarchy effects for an HSGF graph turn negative, primarily in terms of the QPS, for large values of ef_{bottom} (e.g., ef_{bottom}>100 at k=10), depending on the base graph's search performance, which in our setup corresponds to recall ranges above 97%. Furthermore, we find that the subgraphs between the bottom and top levels deliver mixed results, such that the curve for the subgraphs' performance is not monotonically increasing. See, for example, in Figure 5.7 for the second sub-plot, the performance for the second level. That is, even though the complete HSGF shows positive hierarchy effects. This could be an unintended side effect of our evaluation setup, by starting the search always with the same entry points.
The experiments for the different number of levels and the corresponding level subset sizes indicate that there seems to be a somewhat optimal spot for both parameters, depending on the dataset and configuration of the out-degree for the level graphs, which is similar to the findings in [35]. For too large a number of levels, there are only gains in recall but not in QPS. Because of our level-wise measurements of the search performance, we can see that too small a number of levels would stop short of additional, possible performance gains. Across all datasets, including the larger LAION-10M, we find the best performance generally for four or five levels.

Independent of the selection approach, we sometimes measure slightly worse performance for the last level over the second last level, which seems to be related to the size of the last level. Our observations suggest a threshold of around 150 points for the last level to mitigate the risk of a performance drop, but we also find such performance drops for larger last level sizes, which suggests that the selection or level graph itself in those cases is not ideal. Interestingly, for the HNSW, we still find additional hierarchy gains when the last level consists of less than ~50 points, which correspondingly allows the HNSW to have a slightly larger level count.

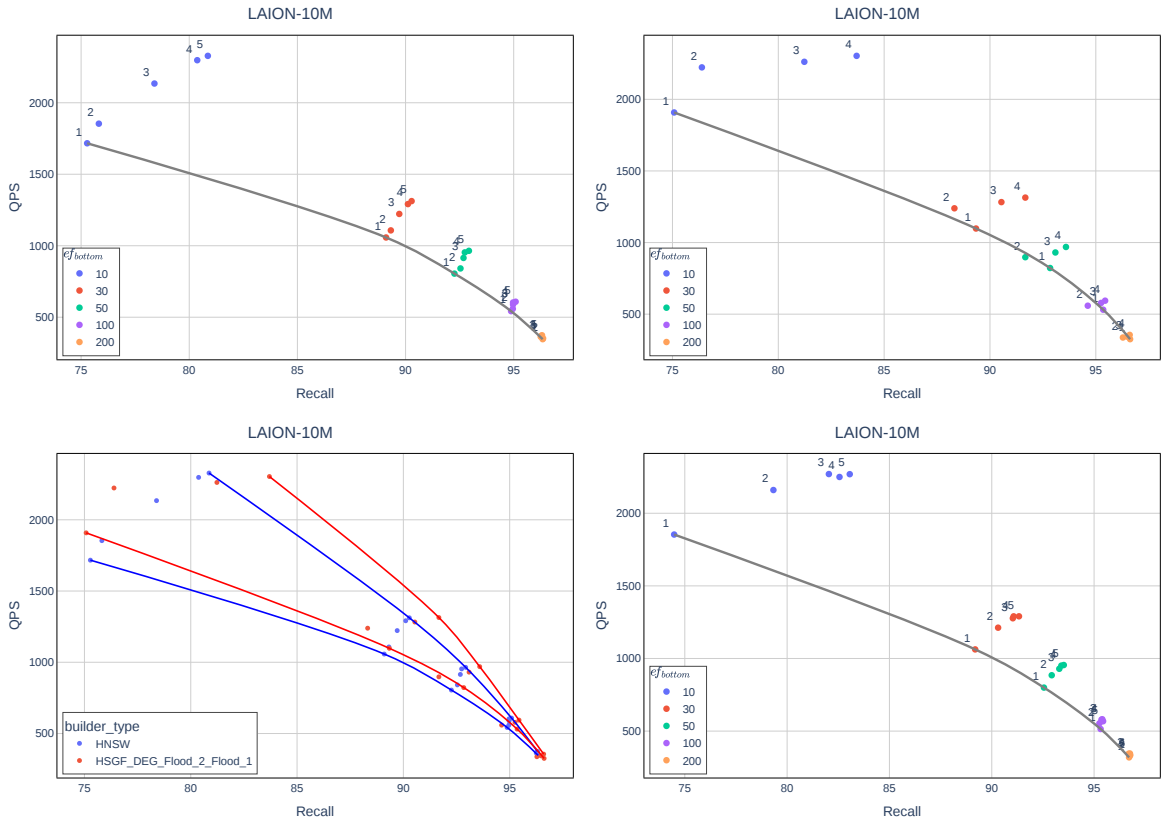

Figure 5.7: QPS over Recall for HNSW, HSGF-DEG-Flooding=2-Flooding=1 and HSGF-DEG-Random (clockwise from top-left) on the LAION-10M dataset for k=1, various ef_{higher} and for different ef_{bottom}. The plot in the bottom left is a combination of the first two and shows the HNSW vs the HSGF-DEG-Flooding=2-Flooding=1. For each, the lower curve shows the search started from the bottom level, and the upper curve shows the search started from the top level.

Comparison to HNSW

Finally, we evaluate the HSGF-DEG-Flooding=2-Flooding=1 and HSGF-DEG-Random on the LAION-10M dataset because its larger size creates a more relevant setup. Note, because of the large memory consumption of RNN during construction, we cannot evaluate HSGF-RNN graphs on the LAION-10M dataset, for which stronger hierarchy effects might have shown.

Our findings for the LAION-10M dataset are in line with our previous findings for the smaller LAION-3M, which is why we almost exclusively only present plots for the larger LAION-10M dataset. In Figure 5.7 and Table 5.3 the level wise performance for the HSGF-DEG-Random and HSGF-DEG-Flooding=2-Flooding=1 are compared to the HNSW for k=1, while two additional Tables for k=10 and k=100 can be found in the appendix (Table 4 and Table 5). Table 5.2 lists information about the time needed to construct each level graph and select each higher level's subset.

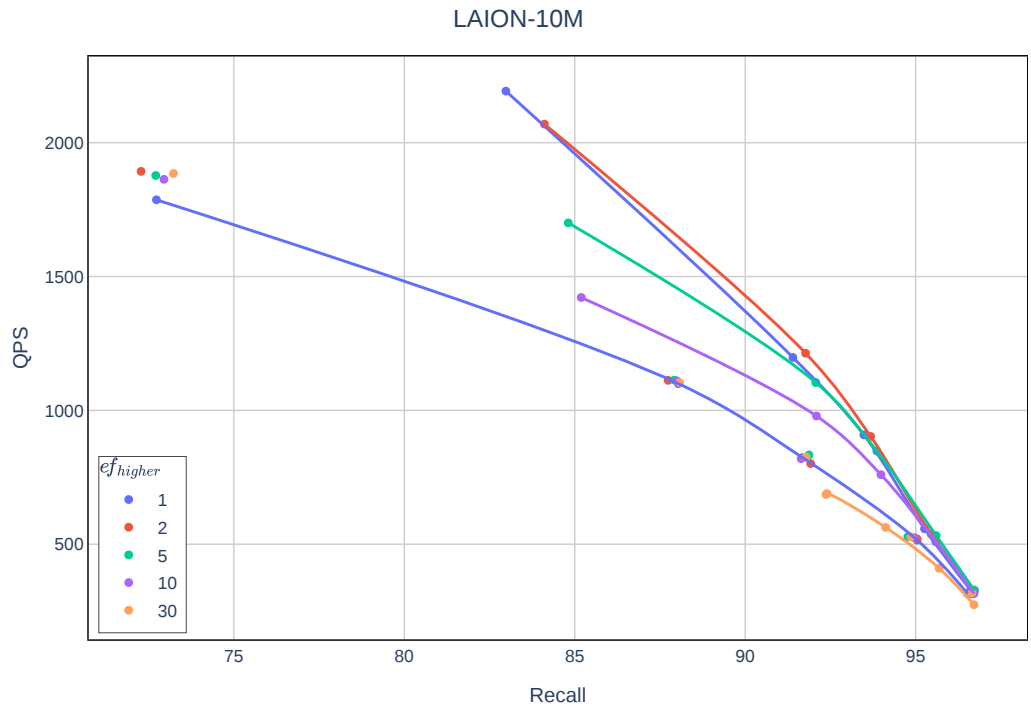

Figure 5.8: QPS over Recall for HSGF-DEG-Flooding=2-Flooding=1 on the LAION-10M dataset with k=1, for ef_{higher}=[1, 2, 5, 30] and multiple ef_{bottom}=[10, 30, 50, 100, 200]. Note, the clustered points around the (blue) bottom level line are the result of minor deviations in the (computer-)system's performance, and therefore primarily impact the QPS but not the recall. Therefore, we only plot the bottom level line for ef_{higher}=1. However, compared to the HNSW results, we are surprised by the relatively large gaps in the deviations.

The random selection for the first subset on the LAION-10M dataset takes less than 2ms in our setup for a subset size of 35,000. The flooding selection with a flooding distance of two takes on an HSGF-DEG graph with a bottom level out-degree of 50 around 71ms (selecting around 34,000 points). The flooding approach is, therefore, as expected, significantly slower than a random selection. But given that the construction of the corresponding HSGF-DEG graph takes more than 1700s for the LAION-10M dataset, we expected the time differences in the selection strategy to only play a minor role on any dataset size. We therefore decide not to analyze the time consumption of the selection approaches in more depth.

As shown in Table 5.2, HNSW has a more than 3x longer construction time compared to the HSGF-DEG graphs. Due to our finding that the HSGF-DEG graph shows strong hierarchy effects, we can construct the HSGF-DEG graph on the LAION-10M so that it results in the first subset size of around 34,000 points, which is less than 0.5% of the original size. This significant size reduction means that the upper levels, including all further subset selections, of an HSGF-DEG graph can be constructed in a fraction of the time it takes to construct the bottom level. Further, this means that the construction time of an HSGF graph is almost exclusively determined by the construction time needed for the graph on the bottom level.

Level	HSGF-DEG-Flooding=2-Flooding=1	HSGF-DEG-Random	HNSW
Graph 0	2624.48 s	2665.97 s	/
Subset 1	70.62 ms	1.01 ms	/
Graph 1	3.74 s	4.08 s	/
Subset 2	1.40 ms	59.77 µs	/
Graph 2	93.86 ms	92.52 ms	/
Subset 3	239.58 µs	7.94 µs	/
Graph 3	5.14 ms	5.25 ms	/
Subset 4	90.41 µs	2.26 µs	/
Graph 4	/	218.53 µs	/
Overall Time	2724 s	2777 s	10834 s
Subset Sizes	[10120191, 33811, 2540, 191]	[10120191, 35000, 2500, 200, 30]	[10120191, 337332, 11256, 353, 16]

Table 5.2: HSGF-DEG build information on the LAION-10M dataset (which consists of 10,120,191 vectors). Both graphs have been built with the same level graph parameters of the DEG with an out-degree=50 and $ef_{construction}$=100. Note the different time scales and that the level starts at index zero; consequently, the first subset is for level one.

ef_b	**HNSW**				**HSGF-DEG-Flooding=2-Flooding=1**				**HSGF-DEG-Rand**			
	Recall		**QPS**		**Recall**		**QPS**		**Recall**		**QPS**	
	$ef_{higher} = 1$											
	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.
10	+7.41%	75.28%	+35.62%	1716.59	+11.48%	75.08%	+20.72%	1908.29	+11.52%	74.48%	+22.42%	1853.49
30	+1.32%	89.1%	+24.06%	1057.67	+2.61%	89.34%	+19.67%	1097.98	+2.4%	89.19%	+21.45%	1062.53
50	+0.73%	92.26%	+19.81%	804.88	+0.81%	92.84%	+17.87%	822.21	+1.05%	92.55%	+19.44%	799.58
100	+0.22%	94.88%	+12.16%	542.43	+0.09%	95.35%	+11.88%	531.02	+0.13%	95.32%	+10.43%	514.39
200	-0.03%	96.34%	+7.84%	346.45	-0.02%	96.62%	+9.21%	325.36	+0.05%	96.71%	+5.79%	319.12
ef_b	$ef_{higher} = 2$				$ef_{higher} = 3$				$ef_{higher} = 3$			
	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.
10	+9.55%	75.28%	+31.66%	1734.34	+12.76%	75.08%	+7.25%	1892.3	+13.52%	74.48%	+5.2%	1867.48
30	+1.87%	89.1%	+25.07%	1046.95	+2.94%	89.34%	+13.59%	1099.17	+2.94%	89.19%	+13.29%	1068.24
50	+0.93%	92.26%	+19.69%	809.81	+1.04%	92.84%	+13.9%	821.75	+1.27%	92.55%	+13.89%	798.42
100	+0.3%	94.88%	+12.55%	545.47	+0.19%	95.35%	+9.34%	529.99	+0.23%	95.32%	+7.49%	516.45
200	-0.04%	96.34%	+7.7%	347.68	-0.06%	96.62%	+4.87%	325.15	-0.04%	96.71%	+3.85%	319.27

Table 5.3: Hierarchy effects for different hierarchy graphs on the LAION 10M dataset for $k = 1$ and different ef_{higher} for different ef_{bottom} (shortened to ef_b). The gains are of the top level's recall/QPS over the bottom level's (B.L.) recall/QPS for the respective ef_{bottom}.

Other datasets

For completeness, we additionally evaluate the HSGF-DEG-Flooding=2-Flooding=1 on the other dataset used in this evaluation, for which the result can be seen in the Appendix in Figure 6. In contrast to our results for the LAION-10M dataset, but similar to our results of HNSW, we find less strong hierarchy effects for the different datasets and almost exclusively gains in terms of QPS. Moreover, we find for some datasets (e.g., GLOVE-100) negative effects on the recall, which is interesting in the sense that the greedy search found entry points for which it potentially got stuck in local minima. A possible explanation for the negative effects for the GLOVE-100 dataset could be its large LID. Furthermore, Figure 9 shows a comparison of the top-level performance over its bottom level for an HNSW graph and a comparable HSGF-DEG-Flooding=2-Flooding=1 graph on the other datasets used in this work.

In a small additional study, we experiment with an HSGF graph that uses a brute-force

computed KNN graph on its level on the LAION-300K dataset (an even smaller subset of the LAION-5B). We observe that the random selection leads to more stable hierarchy effects compared to the flooding selection, and expect this to be the result of the intrinsic problem of the KNN graph for small datasets, for which it might not be connected. For the KNN, the flooding selection just reduces the number of neighbors for each point, as each subset cannot contain any closer neighbors, which therefore intensifies the local minima problem. This effect is less strong for a (fully) random selection, which is likelier to create longer and more diverse connections compared to the flooding selection.

6 Discussion

Through our evaluation setup of querying each level-stacked subgraph of an HNSW graph, we empirically confirm that the hierarchy layers and corresponding redundancy have a positive effect on the search performance of an HNSW graph in terms of both recall and QPS. Evaluating the effects of the hierarchy architecture has, to our knowledge, not been done in such a setup. We find the strongest (positive) hierarchy effects for small ef_{bottom} compared to the respective k and small search result sizes of $k <= 100$. For larger ef_{bottom}, we also find positive effects, although they fade out for increasing ef_{bottom} and show in relative terms stronger gains for the QPS. Our experiments for different ef_{higher} confirms the original work's selection of $ef_{higher} = 1$, although we observe minimal additional gains in the recall with a corresponding drop in QPS for $ef_{higher} > 1$.

The bottom level is the determining factor for the overall search performance of an HNSW graph, given that only the bottom level contains all points of the training data. This confirms our assumption that the higher levels serve as an efficient strategy to select "good" entry points to the bottom level, or, in the form of an analogy, serve as an extension of the search, albeit as a shortcut.

The HSGF allows the construction of ANNS graphs with a hierarchy architecture and a corresponding redundancy of the training data similar to the HNSW graph. The levels in the HSGF graph can be constructed by using any established ANNS graph algorithm combined with different heuristics for the selection of the subset for the next level. By those means, the HSGF adds a range of additional parameters that can and need to be configured when constructing an HSGF graph. The HSGF is generally built level by level from the bottom up, but for a random subset selection, all levels could, in theory, be built at once in parallel as well. However, we hypothesize that the gains from this are marginal as the bottom level is by far the dominating factor for the construction time.

We have shown that for specific configurations of the HSGF (e.g., HSGF-DEG-Flooding=2-Flooding=1), the HSGF significantly improves the search performance in terms of both recall and QPS compared to the graph on the bottom level. Therefore, we confirm that we can replicate the hierarchy effects seen for the HNSW graph for a generalized form of a hierarchy ANNS graph. As such, the HSGF can be seen as an extension to any established graph construction algorithm, as the established graph's architecture is not changed by adding hierarchy layers on top of it, which can improve the search performance of the respective ANNS graph.

Like our findings for the hierarchy effects on the search performance for the HNSW graph, HSGF graphs also show stronger effects for small ef_{bottom} compared to the respective k and search parameter configurations with $k <= 100$. Similarly, the hierarchy effects on the HSGF graphs fade out for larger ef_{bottom}. We therefore argue that larger ef_{bottom} start to

cancel out potential positive hierarchy effects as the search gets increasingly more exhaustive, rendering the selected entry points' influence minimal to none.
However, we find positive hierarchy effects only for certain HSGF configurations and negative effects for others. We observe a correlation to the out-degree of the respective level graph builder, such that DEG and EFANNA, which have a constant out-degree, show (stronger) hierarchy effects than NSSG or RNN, which are pruned graphs with a non-constant and low average out-degree.

Furthermore, the HSGF can be configured with different ANNS graphs on its levels; however, during evaluation, we observe that the best-performing HSGF graphs are the ones with the same level builder on all their levels. This suggests that, even though the levels of the HSGF are built independently of each other, there still seems to be some form of "dependencies" between the layers when searching an HSGF graph. Coming back to our original analogy of viewing the (greedy) search process as two stages of finding the right neighborhood and then the best neighbors, we showed that the hierarchy layers serve as a more efficient strategy to get closer to those right neighborhoods. Following this logic, using the same graph, and therefore the same edge selection and routing characteristics, already on the higher levels, is the best means to that end.

As in the name of the HSGF, it is designed as a framework and works independently of the level graph builders, which therefore adds a variety of additional parameters to create ANNS graphs. Consequently, determining the best parameter configuration becomes more complex as it is generally a resource-intensive and not straightforward process. However, in our evaluation, we showed that, for example, HSGF-DEG graphs do not need large subset sizes for the higher levels to benefit from the hierarchy design. Additionally, we observed that the exact parameter configuration of the level graphs on higher levels in the HSGF has only a limited effect, and using the same configuration for the higher-level graphs as for the bottom level already performs well. Furthermore, we show that the flooding approach presented in this work can be configured to select level subset sizes that result in strong hierarchy effects. The flooding approach works somewhat independently of the dataset, as it works on the level graph itself. We therefore argue that, even though the HSGF has a lot of additional parameters, selecting an HSGF configuration that benefits from the hierarchy design is not more complex than it is for just the bottom-level graph on its own.
Nonetheless, the resource-intensive step of tuning parameters for the bottom level remains, which highlights the importance of a (more) automated parameter tuning. We conjecture that it is likely that, in practice, most applications utilizing ANNS graphs use sub-optimal parameter configurations.

We showed that a flooding distance of two, as part of our presented two-stage flooding approach, selects a subset size of less than 1% of the training data. As a result, all higher levels in an HSGF graph can be constructed in a fraction of the time needed for the bottom level, which means that the HSGF graph needs only a relatively small amount of additional time for its construction compared to the bottom level. While we show that the random selection is magnitudes faster than the flooding approach, it still relies on other strategies to select optimal subset sizes, and that the time difference is negligible for the overall HSGF construction time. Additionally, because of the significantly smaller subset sizes, the impact of better performing parameter configurations for the level graphs on the

higher levels on the respective construction time is also comparatively small. Although we find that for better performing parameter configurations of the level graphs, the hierarchy effect pivots towards higher recall and smaller QPS.

Against our expectations, we find for certain HSGF configurations (e.g., HSGF-RNN) negative effects on both the recall and QPS. This is especially surprising for the recall when using the flooding selection, as it means that the search selects worse entry points to the bottom level, such that finding the correct neighbors becomes more "difficult" for the greedy search. However, the respective graph's architecture might be the cause of this, as we see the effect get smaller for larger ef.

Given that on some datasets the hierarchy effect for different HSGF graphs only shows for the QPS, and not the recall, this suggests that the greedy search already finds the best possible points on that particular graph, regardless of the hierarchy layers (for a respective ef_{bottom}). The hierarchy layers still help to find better entry points to shorten the paths towards the final points, and only a larger ef_{bottom} allows the greedy search to step over "larger local minima" and improve the recall. However, this suggests that larger level subset sizes increase the likelihood of stepping over such minima already on the higher levels, but we are unable to empirically confirm this. Furthermore, we see this as support for our hypothesis that low-average out-degree graphs are less positively affected by the hierarchy design.

We argue that there exists an optimal selection for the subset sizes in an HSGF graph and that the presented flooding approach, specifically in the two-stage procedure, performs as a good strategy towards that optimal selection. However, we are unable to consistently find the right size for the last level, as we sometimes observe better performance for the second last level. This additionally stands in contrast to the HNSW graph, which seems to benefit even from last-level sizes of less than ~30 points.

We show that we can use the HSGF to construct a competitive hierarchy graph in less time than an HNSW graph. However, we want to stress that *only* the gains in recall and QPS are the result of the HSGF's hierarchy design, but that the faster construction time, because of HSGF's black box approach, is the result of the construction performance of the respective graph (here DEG).

Moreover, by using a parallel setting for constructing the graphs in our evaluation, we offer an alternative and more practical view, compared to the current literature, on the construction times of different ANNS graphs. This highlights the significant differences in the speed-ups from a parallel setting for different graph algorithms, which in a single-threaded evaluation setup go unnoticed.

6.1 Limitations

Naturally, this work's research is subject to certain limitations. We reimplement three different, established ANNS graph construction algorithms by *translating* them based on their publicly available code base into our framework of the GraphIndexAPI. This opens our implementations up for small errors by potentially misunderstanding the original code.

We use our base graphs evaluation to check that our implementations reach the originally reported performance levels. Although we cannot directly replicate the exact performance numbers, we expect the differences to be largely hardware and framework related, as the relative comparison to our implementations and between the different graphs hold up. At the same time, our approach is advantageous as we control for fairer comparisons in our evaluation by using the same framework for the graphs as well as the same setup for the search. Specifically, for the DEG graph, we deviate from the original implementation by replacing the range search with the greedy search, which might result in slightly different neighbors for each point and therefore different graphs. However, we confirm the search performance with our base graph evaluation and therefore keep the faster greedy search (implementation).

We are unable to run experiments for larger datasets than the LAION-10M due to limitations of the hardware available to us. We did not have 100% exclusive access to the machine we used for our evaluation, although all results presented in this work should have been run with practically exclusive usage of the machine. For the HNSW and the HSGF, we confirm and show positive hierarchy effects on the search performance already for a dataset size of 10 million vectors, and do not expect the hierarchy effect to fade out for larger datasets. We confirm the hierarchy effects primarily on the subsets of the LAION-5B dataset and find less strong to mixed results for both HNSW and different HSGF graphs for the other presented datasets. The results on the datasets, other than the LAION dataset, however, do not invalidate our finding that it is possible to improve the search performance of an established ANNS graph by adding hierarchy layers on top of it.
Furthermore, we run all our experiments in a parallel setting compared to a single-threaded setting, which is more common in the literature. This impacts the construction times, but we expect it to have only a limited effect on the graph's search performance, which we measure in a single-threaded setting, and does not change the relative comparisons we draw.

Our approach to create a fairer search setup by using the same point (*point-zero*) as the first entry point for the search function on every level of an HSGF graph might have created, in certain cases, undesired side effects in the measured search performance if the point-zero ends up being *badly* connected in the corresponding level graph. However, we expect these cases and their impact to be limited.
Otherwise, our search setup controls for potential deviations by running the search in a similar configuration multiple times and over multiple subgraphs. Additionally, when comparing two HSGF graphs based on their subset selection strategy, we use the same parameter configuration, and even try to tune the level subset sizes to similar values. Furthermore, we compare the search performance of the bottom-level graph between different experiments, as the performance of that graph is independent of the HSGF structure. Our evaluation setup, by querying each hierarchy-subgraph, therefore automatically controls for deviations in the search performance.

We do not empirically evaluate the flooding approach in more detail for a range of different ANNS graphs, as it is highly dependent on the configuration of the individual graph as well as the dataset. Therefore, we do not see the benefit of evaluating the resulting level sizes for a selected number of specific graph and dataset configurations, beyond what we indirectly

did through our evaluation. Given our focus on the HSGF-DEG, using different flooding distances for non-constant out-degree graphs like RNN might perform better, which we did not study extensively in this work.

An inherent problem of accurately evaluating ANNS graphs is the wide range of possible parameter configurations that are dependent on the respective dataset. This issue is even stronger for the HSGF, which, next to the choice for the level selection heuristic, adds virtually any ANNS graph and its respective parameters as parameters. A comparison of all possible configurations is naturally not feasible. However, as already discussed, we find that while the selection of the ANNS graph plays a significant role, its parameters play only a secondary role.

We started this work with the hypothesis that the positive hierarchy effects on the search performance, as seen with the HNSW graph, can be replicated for other ANNS graphs with a hierarchy framework. Naturally, we are therefore prone to be biased towards positive findings, as such confirming our hypothesis alongside shorter construction times for different HSGF configurations compared to HNSW graphs. Firstly, the comparatively longer construction time of HNSW is already established in the literature. Secondly, we control for overfitting to our hypothesis by evaluating our presented HSGF structure for a wide range of parameter configurations and carefully tuning both the HSGF and HNSW to a similar recall range, starting from the lower end of the recall range. Moreover, we store all parameter configurations of our experiments in a structured format alongside the results of each experiment and publish these as CSV files as part of our artifact package, which adds transparency and supports reproducibility.
We ran some additional experiments about our limitations after we finished our main evaluation, which can be found at the beginning of the Appendix 6.3.

6.2 Future Research

Our work opens different venues for future research. The partly mixed results for different datasets and specific HSGF configurations highlight the need for further research on hierarchy ANNS graphs. The Hierarchical Search Graph Framework is designed and implemented with future extensibility in mind. By implementing additional established ANNS graph construction algorithms in our framework, a wider set of HSGF configurations can be evaluated. Evaluating the hierarchy effect on datasets that are larger than the LAION-10M dataset and exploring the performance for unindexed queries should hold important insights for real-world applications. Furthermore, exploring the reasons why graphs like RNN or NSSG do not seem to benefit from hierarchy layers, such that it is difficult to select better entry points, could offer valuable insights and potential pathways towards improving the search performance via a hierarchy architecture, also for these types of graphs.
Evaluating additional subset selection heuristics might hold further improvements to the search performance of our HSGF graphs.

Exploring different strategies to refine the edges of the level graphs in an HSGF graph might hold additional search performance gains. We measure the average recall for our search performance evaluation, but looking into the measured performance for each query

individually might hold insights into the fundamental characteristics of each ANNS graph and could open up new pathways for designing the architecture of future ANNS graphs. Lastly, given the empirical nature of our work, a better theoretical understanding of hierarchy graphs and ANNS graphs in general should be a priority.

6.3 Summary

In this master's thesis, we present the Hierarchical Search Graph Framework to construct hierarchy graphs on top of established ANNS graphs from the bottom level up, inspired by the HNSW graph. We show that the hierarchy design and its corresponding redundancy improve the search performance in terms of both recall and QPS for a range of HSGF configurations and datasets by working as an efficient strategy to select entry points to the bottom-level graph. The HSGF can therefore be seen as an extension of existing non-hierarchical ANNS graphs. Specifically, we find the strongest hierarchy effects for smaller result sizes of $k<100$ and relatively small values of ef_{bottom}. For other HSGF graphs, however, we find mixed and, in a few cases, even negative effects from our hierarchy design, which we expect to be dependent on the characteristics of the respective level graph. Our best-performing HSGF configuration uses a DEG graph on all its levels and uses a two-stage flooding selection heuristic to select the subsets for the higher levels, although we also show similar search performance for a random selection approach. On the LAION-10M dataset for $k = 1$, these HSGF-DEG configurations see search performance gains from the hierarchy design between 5-10% in larger QPS for large ef_{bottom} ranges, and 2-11% gains in recall for small ef_{bottom} ranges. In either case, we show that the performance gains of the hierarchy design are stronger than the potential gains of just increasing ef_{bottom} itself. We present a novel approach to empirically evaluate the search performance of hierarchy graphs by querying each level-stacked subgraph individually, which enables us to confirm the positive effects of hierarchy design for the HNSW graph. Although, also for the HNSW, we find similar limitations to the hierarchy effect as found for the HSGF. Furthermore, we show that the HSGF can be efficiently and quickly constructed as the subset sizes on the higher levels are significantly smaller than the complete training data used for the bottom level. Consequently, the additional overhead of the HSGF compared to its *plain* bottom-level graph is marginal. Therefore, we can tune a DEG graph through our hierarchy framework to similar recall and QPS levels as a comparable HNSW graph, although with a significantly shorter construction time.

Our work opens new venues for future research into hierarchical approximate nearest neighbor graphs. We publish all our code and results as part of this work's artifact package.

Bibliography

[1] Alexandr Andoni and Piotr Indyk. "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. ISSN: 0272-5428. Oct. 2006, pp. 459–468. DOI: 10.1109/FOCS.2006.49. URL: https://ieeexplore.ieee.org/document/4031381 (visited on 12/24/2024).

[2] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. "Practical and optimal LSH for angular distance". In: *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*. Vol. 1. NIPS'15. Cambridge, MA, USA: MIT Press, Dec. 2015, pp. 1225–1233. (Visited on 02/16/2025).

[3] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. "Accelerated Nearest Neighbor Search with Quick ADC". In: *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. ICMR '17. New York, NY, USA: Association for Computing Machinery, June 2017, pp. 159–166. ISBN: 9781450347013. DOI: 10.1145/3078971.3078992. URL: https://dl.acm.org/doi/10.1145/3078971.3078992 (visited on 07/14/2025).

[4] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. "HD-index: pushing the scalability-accuracy boundary for approximate kNN search in high-dimensional spaces". In: *Proc. VLDB Endow.* 11.8 (Apr. 2018), pp. 906–919. ISSN: 2150-8097. DOI: 10.14778/3204028.3204034. URL: https://dl.acm.org/doi/10.14778/3204028.3204034 (visited on 02/16/2025).

[5] Sunil Arya and David M. Mount. "Approximate nearest neighbor queries in fixed dimensions". In: *Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms*. SODA '93. USA: Society for Industrial and Applied Mathematics, Jan. 1993, pp. 271–280. ISBN: 9780898713138. (Visited on 02/20/2025).

[6] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. *ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms*. arXiv:1807.05614 [cs]. July 2018. DOI: 10.48550/arXiv.1807.05614. URL: http://arxiv.org/abs/1807.05614 (visited on 12/31/2024).

[7] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. "ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms". In: *Information Systems* 87 (Jan. 2020), p. 101374. ISSN: 0306-4379. DOI: 10.1016/j.is.2019.02.006. URL: https://www.sciencedirect.com/science/article/pii/S0306437918303685 (visited on 02/22/2025).

[8] Artem Babenko and Victor Lempitsky. "The Inverted Multi-Index". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.6 (June 2015), pp. 1247–1260. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2014.2361319. URL: https://ieeexplore.ieee.org/document/6915715 (visited on 02/22/2025).

[9] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. "When Is "Nearest Neighbor" Meaningful?" en. In: *Database Theory — ICDT'99*. Ed. by Catriel Beeri and Peter Buneman. Berlin, Heidelberg: Springer, 1999, pp. 217–235. ISBN: 9783540492573. DOI: 10.1007/3-540-49257-7_15.

[10] Sebastian Bruch. *Foundations of Vector Retrieval*. Springer Nature Switzerland, 2024. ISBN: 9783031551826.

[11] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. "Searching in metric spaces". In: *ACM Comput. Surv.* 33.3 (Sept. 2001), pp. 273–321. ISSN: 0360-0300. DOI: 10.1145/502807.502808. URL: https://dl.acm.org/doi/10.1145/502807.502808 (visited on 07/05/2025).

[12] Patrick Chen, Wei-Cheng Chang, Jyun-Yu Jiang, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. "FINGER: Fast Inference for Graph-based Approximate Nearest Neighbor Search". In: *Proceedings of the ACM Web Conference 2023*. WWW '23. New York, NY, USA: Association for Computing Machinery, Apr. 2023, pp. 3225–3235. ISBN: 9781450394161. DOI: 10.1145/3543507.3583318. URL: https://dl.acm.org/doi/10.1145/3543507.3583318 (visited on 02/20/2025).

[13] D.W. Dearholt, N. Gonzales, and G. Kurup. "Monotonic Search Networks For Computer Vision Databases". In: *Twenty-Second Asilomar Conference on Signals, Systems and Computers*. Vol. 2. ISSN: 1058-6393. Oct. 1988, pp. 548–553. DOI: 10.1109/ACSSC.1988.754602. URL: https://ieeexplore.ieee.org/document/754602 (visited on 01/27/2025).

[14] B. Delaunay. "Neue Darstellung der geometrischen Kristallographie". In: *Zeitschrift für Kristallographie - Crystalline Materials* 84.1-6 (1933), pp. 109–149. DOI: doi:10.1524/zkri.1933.84.1.109. URL: https://doi.org/10.1524/zkri.1933.84.1.109.

[15] Haya Diwan, Jinrui Gou, Cameron Musco, Christopher Musco, and Torsten Suel. "Navigable Graphs for High-Dimensional Nearest Neighbor Search: Constructions and Limits". In: *Advances in Neural Information Processing Systems*. Ed. by A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang. Vol. 37. Curran Associates, Inc., 2024, pp. 59513–59531. URL: https://proceedings.neurips.cc/paper_files/paper/2024/file/6dc63b4063c978cf195bc15178e8152a-Paper-Conference.pdf.

[16] Wei Dong, Charikar Moses, and Kai Li. "Efficient k-nearest neighbor graph construction for generic similarity measures". In: *Proceedings of the 20th international conference on World wide web*. WWW '11. New York, NY, USA: Association for Computing Machinery, Mar. 2011, pp. 577–586. ISBN: 9781450306324. DOI: 10.1145/1963405.1963487. URL: https://dl.acm.org/doi/10.1145/1963405.1963487 (visited on 12/13/2024).

[17] Cong Fu and Deng Cai. *EFANNA : An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph*. arXiv:1609.07228 [cs]. Dec. 2016. DOI: 10.48550/arXiv.1609.07228. URL: http://arxiv.org/abs/1609.07228 (visited on 02/20/2025).

[18] Cong Fu, Changxu Wang, and Deng Cai. "High Dimensional Similarity Search With Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.8 (Aug. 2022), pp. 4139–4150. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2021.3067706. URL: https://ieeexplore.ieee.org/abstract/document/9383170 (visited on 02/22/2025).

[19] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. "Fast approximate nearest neighbor search with the navigating spreading-out graph". In: *Proc. VLDB Endow.* 12.5 (Jan. 2019), pp. 461–474. ISSN: 2150-8097. DOI: 10.14778/3303753.3303754. URL: https://dl.acm.org/doi/10.14778/3303753.3303754 (visited on 02/14/2025).

[20] K. Ruben Gabriel and Robert R. Sokal. "A New Statistical Approach to Geographic Variation Analysis". In: *Systematic Biology* 18.3 (Sept. 1969), pp. 259–278. ISSN: 1063-5157. DOI: 10.2307/2412323. eprint: https://academic.oup.com/sysbio/article-pdf/18/3/259/4595606/18-3-259.pdf. URL: https://doi.org/10.2307/2412323.

[21] Jianyang Gao and Cheng Long. "High-Dimensional Approximate Nearest Neighbor Search: with Reliable and Efficient Distance Comparison Operations". In: *Proc. ACM Manag. Data* 1.2 (June 2023), 137:1–137:27. DOI: 10.1145/3589282. URL: https://dl.acm.org/doi/10.1145/3589282 (visited on 02/20/2025).

[22] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. "Optimized Product Quantization for Approximate Nearest Neighbor Search". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2013, pp. 2946–2953. DOI: 10.1109/CVPR.2013.379. URL: https://ieeexplore.ieee.org/document/6619223 (visited on 02/16/2025).

[23] Ben Harwood and Tom Drummond. "FANNG: Fast Approximate Nearest Neighbour Graphs". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 1063-6919. June 2016, pp. 5713–5722. DOI: 10.1109/CVPR.2016.616. URL: https://ieeexplore.ieee.org/document/7780985 (visited on 02/15/2025).

[24] Tadeo Hepperle. *Vantage-Point Tree to HNSW: Fast Index Construction for Approximate Nearest Neighbor Search in High-Dimensional Data*. URL: https://github.com/tadeohepperle/vecnn/blob/main/Tadeo%20Hepperle%20HNSW%20Master%20Thesis.pdf (visited on 07/24/2025).

[25] Nico Hezel, Kai Uwe Barthel, Konstantin Schall, and Klaus Jung. "An Exploration Graph with Continuous Refinement for Efficient Multimedia Retrieval". In: *Proceedings of the 2024 International Conference on Multimedia Retrieval*. ICMR '24. New York, NY, USA: Association for Computing Machinery, June 2024, pp. 657–665. DOI: 10.1145/3652583.3658117. URL: https://dl.acm.org/doi/10.1145/3652583.3658117 (visited on 02/14/2025).

[26] Nico Hezel, Kai Uwe Barthel, Konstantin Schall, and Klaus Jung. *Fast Approximate Nearest Neighbor Search with a Dynamic Exploration Graph using Continuous Refinement*. arXiv:2307.10479 [cs]. July 2023. DOI: 10.48550/arXiv.2307.10479. URL: http://arxiv.org/abs/2307.10479 (visited on 12/13/2024).

[27] Nico Hezel, Kai Uwe Barthel, Bruno Schilling, Konstantin Schall, and Klaus Jung. "Dynamic Exploration Graph: A Novel Approach for Efficient Nearest Neighbor Search in Evolving Multimedia Datasets". en. In: *MultiMedia Modeling*. Ed. by Ichiro Ide, Ioannis Kompatsiaris, Changsheng Xu, Keiji Yanai, Wei-Ta Chu, Naoko Nitta, Michael Riegler, and Toshihiko Yamasaki. Singapore: Springer Nature, 2025, pp. 333–347. DOI: 10.1007/978-981-96-2054-8_25.

[28] Elias Jääsaari, Ville Hyvönen, and Teemu Roos. "LoRANN: Low-Rank Matrix Factorization for Approximate Nearest Neighbor Search". In: *Advances in Neural Information Processing Systems*. Ed. by A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang. Vol. 37. Curran Associates, Inc., 2024, pp. 102121–102153. URL: https://proceedings.neurips.cc/paper_files/paper/2024/file/b939da3932e88ded5e9b08026e35069d-Paper-Conference.pdf.

[29] Herve Jégou, Matthijs Douze, and Cordelia Schmid. "Product Quantization for Nearest Neighbor Search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (Jan. 2011), pp. 117–128. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2010.57. URL: https://ieeexplore.ieee.org/document/5432202 (visited on 02/22/2025).

[30] Zhi Jing, Yongye Su, and Yikun Han. "When Large Language Models Meet Vector Databases: A Survey". In: Feb. 2025, pp. 7–13. DOI: 10.1109/AIxMM62960.2025.00008. URL: https://ieeexplore.ieee.org/abstract/document/11005010 (visited on 07/22/2025).

[31] Yannis Kalantidis and Yannis Avrithis. "Locally Optimized Product Quantization for Approximate Nearest Neighbor Search". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2014, pp. 2329–2336. DOI: 10.1109/CVPR.2014.298. URL: https://ieeexplore.ieee.org/document/6909695 (visited on 02/16/2025).

[32] D. T. Lee and B. J. Schachter. "Two algorithms for constructing a Delaunay triangulation". en. In: *International Journal of Computer & Information Sciences* 9.3 (June 1980), pp. 219–242. ISSN: 1573-7640. DOI: 10.1007/BF00977785. URL: https://doi.org/10.1007/BF00977785 (visited on 02/20/2025).

[33] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. "Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement". In: *IEEE Transactions on Knowledge and Data Engineering* 32.8 (Aug. 2020), pp. 1475–1488. ISSN: 1558-2191. DOI: 10.1109/TKDE.2019.2909204. URL: https://ieeexplore.ieee.org/abstract/document/8681160 (visited on 02/20/2025).

[34] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. "Multi-probe LSH: efficient indexing for high-dimensional similarity search". In: *Proceedings of the 33rd international conference on Very large data bases*. VLDB '07. Vienna, Aus-

tria: VLDB Endowment, Sept. 2007, pp. 950–961. ISBN: 9781595936493. (Visited on 07/14/2025).

[35] Yu A. Malkov and D. A. Yashunin. "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.4 (Apr. 2020), pp. 824–836. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2018.2889473. URL: https://ieeexplore.ieee.org/document/8594636 (visited on 02/14/2025).

[36] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. "Approximate nearest neighbor algorithm based on navigable small world graphs". In: *Information Systems* 45 (Sept. 2014), pp. 61–68. ISSN: 0306-4379. DOI: 10.1016/j.is.2013.10.006. URL: https://www.sciencedirect.com/science/article/pii/S0306437913001300 (visited on 02/14/2025).

[37] Einat Minkov, Ramnath Balasubramanyan, and William W. Cohen. "Activity-centred Search in Email". In: *CEAS 2008 - The Fifth Conference on Email and Anti-Spam, 21-22 August 2008, Mountain View, California, USA*. 2008. URL: http://www.ceas.cc/2008/papers/ceas2008-paper-54.pdf.

[38] Marius Muja and David G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration". In: *International Conference on Computer Vision Theory and Application VISSAPP'09)*. INSTICC Press, 2009, pp. 331–340.

[39] Yutaro Oguri and Yusuke Matsui. *Theoretical and Empirical Analysis of Adaptive Entry Point Selection for Graph-based Approximate Nearest Neighbor Search*. arXiv:2402.04713 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2402.04713. URL: http://arxiv.org/abs/2402.04713 (visited on 02/20/2025).

[40] Naoki Ono and Yusuke Matsui. *Relative NN-Descent: A Fast Index Construction for Graph-Based Approximate Nearest Neighbor Search*. arXiv:2310.20419 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2310.20419. URL: http://arxiv.org/abs/2310.20419 (visited on 12/23/2024).

[41] Naoki Ono and Yusuke Matsui. "Relative NN-Descent: A Fast Index Construction for Graph-Based Approximate Nearest Neighbor Search". In: *Proceedings of the 31st ACM International Conference on Multimedia*. MM '23. New York, NY, USA: Association for Computing Machinery, Oct. 2023, pp. 1659–1667. DOI: 10.1145/3581783.3612290. URL: https://dl.acm.org/doi/10.1145/3581783.3612290 (visited on 02/15/2025).

[42] 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)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162/.

[43] Alexander Ponomarenko, Nikita Avrelin, Bilegsaikhan Naidan, and Leonid Boytsov. "Comparative Analysis of Data Structures for Approximate Nearest Neighbor Search". In: Jan. 2014.

[44] William Pugh. "Skip lists: a probabilistic alternative to balanced trees". In: *Commun. ACM* 33.6 (June 1990), pp. 668–676. ISSN: 0001-0782. DOI: 10.1145/78973.78977. URL: https://dl.acm.org/doi/10.1145/78973.78977 (visited on 02/24/2025).

[45] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. "LAION-5B: an open large-scale dataset for training next generation image-text models". In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., Nov. 2022, pp. 25278–25294. ISBN: 9781713871088. (Visited on 07/14/2025).

[46] Larissa C. Shimomura, Rafael Seidi Oyamada, Marcos R. Vieira, and Daniel S. Kaster. "A survey on graph-based methods for similarity searches in metric spaces". In: *Information Systems* 95 (Jan. 2021), p. 101507. ISSN: 0306-4379. DOI: 10.1016/j.is.2020.101507. URL: https://www.sciencedirect.com/science/article/pii/S0306437920300181 (visited on 02/20/2025).

[47] Chanop Silpa-Anan and Richard Hartley. "Optimised KD-trees for fast image descriptor matching". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587638. URL: https://ieeexplore.ieee.org/document/4587638 (visited on 02/16/2025).

[48] *Sixty years of percolation*. arXiv:1712.04651 [math]. Dec. 2017. DOI: 10.48550/arXiv.1712.04651. URL: http://arxiv.org/abs/1712.04651 (visited on 02/23/2025).

[49] *The Faiss library*. arXiv:2401.08281 [cs]. Feb. 2025. DOI: 10.48550/arXiv.2401.08281. URL: http://arxiv.org/abs/2401.08281 (visited on 02/22/2025).

[50] Godfried T. Toussaint. "The relative neighbourhood graph of a finite planar set". In: *Pattern Recognition* 12.4 (1980), pp. 261–268. ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(80)90066-7. URL: https://www.sciencedirect.com/science/article/pii/0031320380900667.

[51] *Voyager: Spotify's nearest-neighbor search library for Python and Java*. URL: https://spotify.github.io/voyager/ (visited on 02/09/2025).

[52] Hui Wang, Wan-Lei Zhao, Xiangxiang Zeng, and Jianye Yang. "Fast k-NN Graph Construction by GPU based NN-Descent". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 1929–1938. ISBN: 9781450384469. DOI: 10.1145/3459637.3482344. URL: https://doi.org/10.1145/3459637.3482344 (visited on 12/23/2024).

[53] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. "A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search". In: *Proc. VLDB Endow.* 14.11 (July 2021), pp. 1964–1978. ISSN: 2150-8097. DOI: 10.14778/3476249.3476255. URL: https://dl.acm.org/doi/10.14778/3476249.3476255 (visited on 02/20/2025).

[54] Ming Yang, Yuzheng Cai, and Weiguo Zheng. "CSPG: Crossing Sparse Proximity Graphs for Approximate Nearest Neighbor Search". In: *Advances in Neural Information Processing Systems*. Ed. by A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang. Vol. 37. Curran Associates, Inc., 2024, pp. 103076–103100. URL: https://proceedings.neurips.cc/paper_files/paper/2024/file/bab1486cec466c980b40e7d633dd4bbc-Paper-Conference.pdf.

[55] Weijie Zhao, Shulong Tan, and Ping Li. "SONG: Approximate Nearest Neighbor Search on GPU". In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. ISSN: 2375-026X. Apr. 2020, pp. 1033–1044. DOI: 10.1109/ICDE48307.2020.00094. URL: https://ieeexplore.ieee.org/document/9101583 (visited on 12/13/2024).

List of Abbreviations

ANNS	Approximate (k) Nearest Neighbor Search
ANN	Approximate Nearest Neighbor
CSPG	Crossing Sparse Proximity Framework
DEG	Dynamic Exploration Graph
DG	Delaunay Graph
HNSW	Hierarchical Navigable Small World (Graph)
HSGF	Hierarchical Search Graph Framework
IVF	Inverted File Index
LID	Local Intrinsic Dimensionality
KNNS	k Nearest Neighbor Search
MRNG	Monotonic Relative Neighborhood Graph
MSNET	Monotonic Search Network
NSG	Navigating Spreading-out Graph
NSSG	Navigating Satellite System Graph
NSW	Navigable Small World (Graph)
QPS	Queries-Per-Second
RNG	Relative Neighborhood Graph
RNG*(S)	Randomized Neighborhood Graph
SSG	Satellite System Graph

List of Figures

List of Tables

List of Algorithms

Appendix

Consult the artifact package, which is described in a separate README file, to inspect the hierarchy effects data for other parameter configurations.

Additional Experiments

Following our main evaluation, we run a few additional experiments, but on a smaller scale, which in parts are related to our limitations.

The KNN graph is on a local level optimal for its k. We evaluate whether a KNN graph on higher levels improves the entry point selection and thus the search performance further over our existing approaches. Therefore, we compare an HSGF-DEG-Flooding=2--Flooding=1 and an HSGF-DEG-Flooding=2-k=60-NN-Rand graph, which use the same parameters for the bottom DEG graph, have the same out-degree specified for the higher levels, and are tuned to roughly the same level sizes. We find no significant difference in their level-wise search performance, but given the nature of the KNN, we measure a significantly longer construction time for the HSGF-KNN graph (see Figure 1).
Given the small subset sizes on higher levels, there might be a risk that the DEG approaches the edge selection of the KNN for the same degree. However, we did not investigate this further.

We noted the potential limitation of using the *point-zero* in our evaluation. We briefly re-run some experiments and potentially observe a more stable behavior for the middle levels of the HSGF graphs, but cannot empirically confirm this. Nonetheless, this does not impact the validity of our main results.

Given that our work is based on the hierarchy design from the HNSW, we evaluate the performance of a hierarchy graph, which is the result of swapping the bottom level of an HNSW graph with a different established graph. Therefore, we evaluate whether the hierarchy levels of the HNSW can be used to efficiently select entry points to another ANNS graph and increase its search performance. We find negative effects for all but the second level of a tested DEG merged with an HNSW. Therefore, we find worse performance than a comparable HSGF-DEG.

We briefly test the performance for an HNSW graph and an HSGF-DEG-Flooding=2--Flooding=1 graph on normally distributed data (n=1,000,000, d=30) with unindexed queries for $k = 1$, 10, and 100. We find that the hierarchy does not influence the recall and slightly negatively influences the QPS for both graphs for all three k. Given the higher LID of this normally distributed data, our findings are similar to our previous results for the GLOVE-100 dataset. Therefore, we believe a more extensive study on how to utilize the hierarchy

design on datasets with a high LID, and additionally with unindexed queries, could hold valuable insights.

Additional Plots and Tables

ef_{bottom}	HNSW - LAION-3M				HNSW - LAION-10M			
	Recall		QPS		Recall		QPS	
	Gain	Bottom-Level	Gain	Bottom-Level	Gain	Bottom-Level	Gain	Bottom-Level
10	+6.78%	79.06%	+26.38%	2074.28	+7.41%	75.28%	+35.62%	1716.59
30	+1.46%	91.85%	+19.19%	1286.51	+1.32%	89.1%	+24.06%	1057.67
50	+0.78%	94.52%	+14.77%	983.66	+0.73%	92.26%	+19.81%	804.88
100	+0.34%	96.86%	+9.27%	656.54	+0.22%	94.88%	+12.16%	542.43
200	+0.09%	98.04%	+5.94%	413.89	-0.03%	96.34%	+7.84%	346.45

Table 1: Hierarchy effects for different HNSW graphs on the LAION-3M and 10M dataset for k=1 and ef_{higher}=1 for different ef_{bottom}. The gains are of the top level's recall/QPS over the bottom level's recall/QPS for the respective ef_{bottom}. The level sizes for the HNSW on the LAION-3M dataset are: [3.000.000, 99.912, 3257, 128, 2] and for the HNSW on the LAION-10M dataset are: [10.120.191, 337.332, 11.256, 353, 16].

ef_{bottom}	HNSW - LAION-3M				HNSW - LAION-10M			
	Recall		QPS		Recall		QPS	
	Gain	Bottom-Level	Gain	Bottom-Level	Gain	Bottom-Level	Gain	Bottom-Level
10	+9.47%	79.06%	+11.26%	2068.55	+12.34%	75.28%	+20.04%	1732.91
30	+2.01%	91.85%	+10.2%	1285.06	+2.46%	89.1%	+16.65%	1063.91
50	+1.07%	94.52%	+8.64%	977.47	+1.24%	92.26%	+13.38%	815.8
100	+0.42%	96.86%	+4.72%	656.64	+0.34%	94.88%	+10.05%	540.53
200	+0.14%	98.04%	+2.67%	414.99	+0.0%	96.34%	+3.81%	353.81

Table 2: Hierarchy effects for different HNSW graphs on the LAION-3M and LAION-10M dataset for k=1 and ef_{higher}=5 for different ef_{bottom}. The gains are of the top level's recall/QPS over the bottom level's recall/QPS for the respective ef_{bottom}. The level sizes for the HNSW on the LAION-3M dataset are: [3.000.000, 99.912, 3257, 128, 2] and for the HNSW on the LAION-10M dataset are: [10.120.191, 337.332, 11.256, 353, 16].

ef_{bottom}	HNSW - LAION-3M				HNSW - LAION-10M			
	Recall		QPS		Recall		QPS	
	Gain	Bottom-Level	Gain	Bottom-Level	Gain	Bottom-Level	Gain	Bottom-Level
100	+0.31%	95.2%	+9.17%	639.43	+0.2%	95.36%	+11.49%	488.94
150	+0.13%	96.8%	+7.01%	491.48	+0.08%	96.78%	+9.13%	378.32
200	+0.08%	97.57%	+5.53%	404.11	+0.03%	97.41%	+7.66%	311.92
300	+0.04%	98.33%	+4.05%	302.95	+0.02%	98.04%	+5.01%	236.1
500	+0.02%	98.88%	+2.56%	207.69	+0.01%	98.5%	+3.48%	162.04
700	-0.0%	99.11%	+1.8%	160.76	-0.0%	98.7%	+2.54%	125.33
1000	-0.0%	99.28%	+1.31%	121.69	-0.0%	98.82%	+1.72%	95.18

Table 3: Hierarchy effects for different HNSW graphs on the LAION-3M and LAION-10M dataset for k=100 and ef_{higher}=1 for different ef_{bottom}. The gains are of the top level's recall/QPS over the bottom level's recall/QPS for the respective ef_{bottom}. The level sizes for the HNSW on the LAION-3M dataset are: [3.000.000, 99.912, 3257, 128, 2] and for the HNSW on the LAION-10M dataset are: [10.120.191, 337.332, 11.256, 353, 16].

ef_{bottom}	HNSW				HSGF-DEG-Flooding=2-Flooding=1				HSGF-DEG-Rand			
	Recall		QPS		Recall		QPS		Recall		QPS	
	$ef_{higher} = 1$											
	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.
10	+7.37%	75.69%	+35.62%	1696.29	+11.86%	73.69%	+20.49%	2008.58	+12.2%	72.91%	+25.6%	1859.6
30	+1.15%	90.28%	+24.2%	1036.84	+2.42%	89.67%	+26.41%	1103.51	+2.59%	89.25%	+23.87%	1074.09
50	+0.54%	93.57%	+19.35%	795.49	+0.78%	93.47%	+21.61%	818.76	+1.23%	93.01%	+19.9%	797.98
100	+0.22%	96.06%	+11.84%	532.3	+0.17%	96.16%	+14.87%	531.0	+0.21%	96.05%	+10.84%	515.47
200	+0.04%	97.34%	+8.73%	334.87	+0.07%	97.46%	+11.4%	324.64	-0.04%	97.5%	+8.38%	317.19
ef_{bottom}	$ef_{higher} = 2$				$ef_{higher} = 3$				$ef_{higher} = 3$			
	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.
10	+9.33%	75.69%	+33.27%	1625.88	+13.47%	73.69%	+9.16%	1902.09	+14.16%	72.91%	+8.66%	1855.67
30	+1.74%	90.28%	+22.49%	1011.47	+2.89%	89.67%	+15.77%	1103.08	+3.19%	89.25%	+15.28%	1070.85
50	+0.77%	93.57%	+19.17%	769.9	+1.01%	93.47%	+16.02%	819.71	+1.47%	93.01%	+15.81%	798.05
100	+0.25%	96.06%	+11.36%	517.57	+0.23%	96.16%	+11.36%	531.24	+0.35%	96.05%	+8.99%	515.34
200	+0.05%	97.34%	+6.55%	333.51	+0.03%	97.46%	+8.29%	324.56	-0.02%	97.5%	+6.58%	317.16

Table 4: Hierarchy effects for different hierarchy graphs on the LAION-10M dataset for $k = 10$ and different ef_{higher} for different ef_{bottom}. The gains are of the top level's recall/QPS over the bottom level's (B.L.) recall/QPS for the respective ef_{bottom}.

ef_{bottom}	HNSW				HSGF-DEG-Flooding=2-Flooding=1				HSGF-DEG-Rand			
	Recall		QPS		Recall		QPS		Recall		QPS	
	$ef_{higher} = 1$											
	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.
100	+0.2%	95.36%	+11.49%	488.94	+0.12%	94.52%	+7.68%	564.96	+0.17%	94.4%	+15.54%	514.59
150	+0.08%	96.78%	+9.13%	378.32	+0.0%	96.33%	+13.95%	403.79	-0.01%	96.3%	+6.89%	414.65
200	+0.03%	97.41%	+7.66%	311.92	+0.04%	97.11%	+12.5%	328.0	-0.06%	97.15%	+5.57%	337.41
300	+0.02%	98.04%	+5.01%	236.1	+0.03%	97.89%	+10.19%	240.44	-0.02%	97.9%	+4.74%	246.35
500	+0.01%	98.5%	+3.48%	162.04	+0.01%	98.44%	+8.49%	160.79	-0.01%	98.44%	+9.24%	155.9
700	-0.0%	98.7%	+2.54%	125.33	+0.01%	98.66%	+7.08%	122.13	-0.01%	98.65%	+8.59%	117.77
1000	-0.0%	98.82%	+1.72%	95.18	+0.01%	98.82%	+5.78%	91.13	-0.01%	98.82%	+7.14%	87.95
ef_{bottom}	$ef_{higher} = 2$				$ef_{higher} = 3$				$ef_{higher} = 3$			
	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.	Gain	B.-L.
100	+0.26%	95.36%	+12.21%	492.01	+0.22%	94.52%	+12.78%	534.11	+0.35%	94.4%	+6.48%	546.67
150	+0.1%	96.78%	+8.96%	381.64	+0.05%	96.33%	+11.52%	403.05	+0.08%	96.3%	+11.3%	389.75
200	+0.04%	97.41%	+9.02%	310.52	+0.04%	97.11%	+9.95%	328.47	-0.01%	97.15%	+3.85%	336.8
300	+0.01%	98.04%	+5.35%	237.39	-0.02%	97.89%	+9.87%	239.08	-0.01%	97.9%	+2.97%	247.12
500	+0.0%	98.5%	+3.85%	162.86	-0.01%	98.44%	+7.39%	160.82	-0.01%	98.44%	+1.66%	165.92
700	-0.01%	98.7%	+2.22%	125.07	-0.01%	98.66%	+7.75%	120.46	-0.02%	98.65%	+7.45%	118.26
1000	-0.0%	98.82%	+0.36%	96.71	-0.02%	98.82%	+4.76%	91.38	-0.01%	98.82%	+6.78%	87.88

Table 5: Hierarchy effects for different hierarchy graphs on the LAION 10M dataset for $k = 100$ and different ef_{higher} for different ef_{bottom}. The gains are of the top level's recall/QPS over the bottom level's (B.L.) recall/QPS for the respective ef_{bottom}.

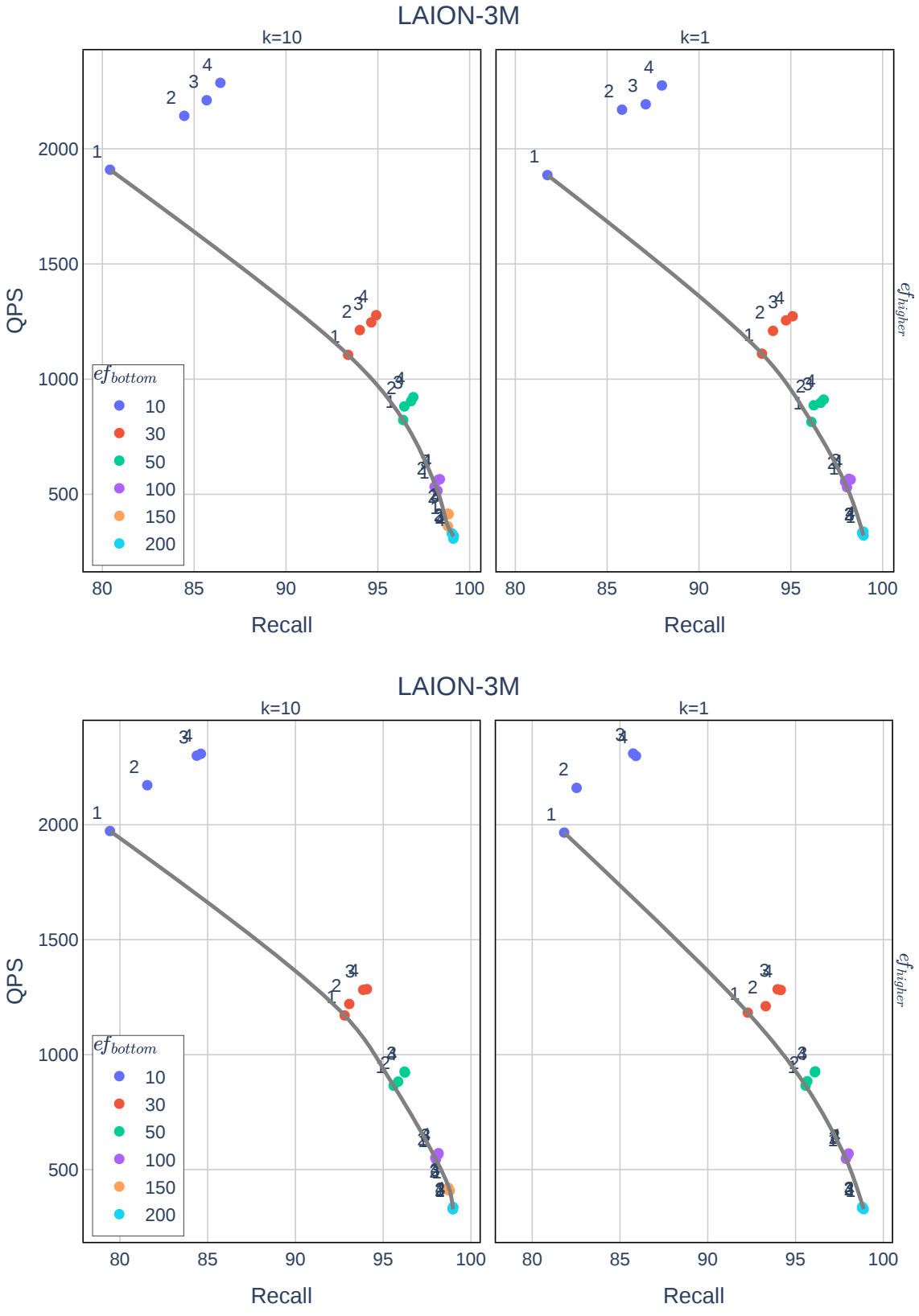

Figure 1: Queries-per-second over recall for HSGF-DEG-Flooding=2-Flooding=1 (top) and HSGF-DEG-Flooding=2-k=60-NN-Rand (bottom) on the LAION-3M datasets for $k = 10$ and $k = 1$ and different *ef*. The small deviations in the performance for the bottom level graph (the points annotated with "1") are primarily the consequence of the inherent randomness in the DEG's construction process.

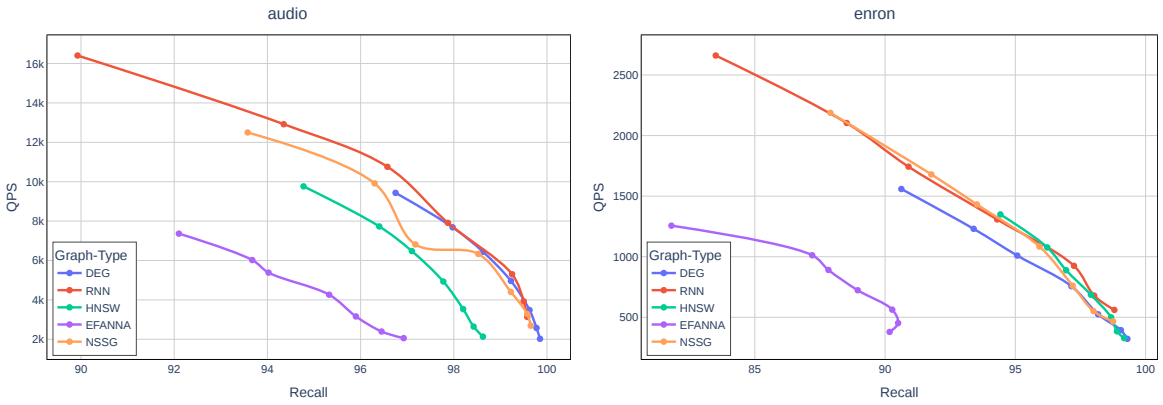

Figure 2: Queries-per-second over recall for the Audio and Enron dataset for different ef. The search configuration for both datasets is the following: k=20 and ef_{bottom}=[20, 30, 40, 60, 100, 150, 200]. We could not find a direct cause for the atypical curves for NSSG and EFANNA on the AUDIO and ENRON datasets, respectively, but we hypothesize that the greedy search might have gotten stuck in local minima.

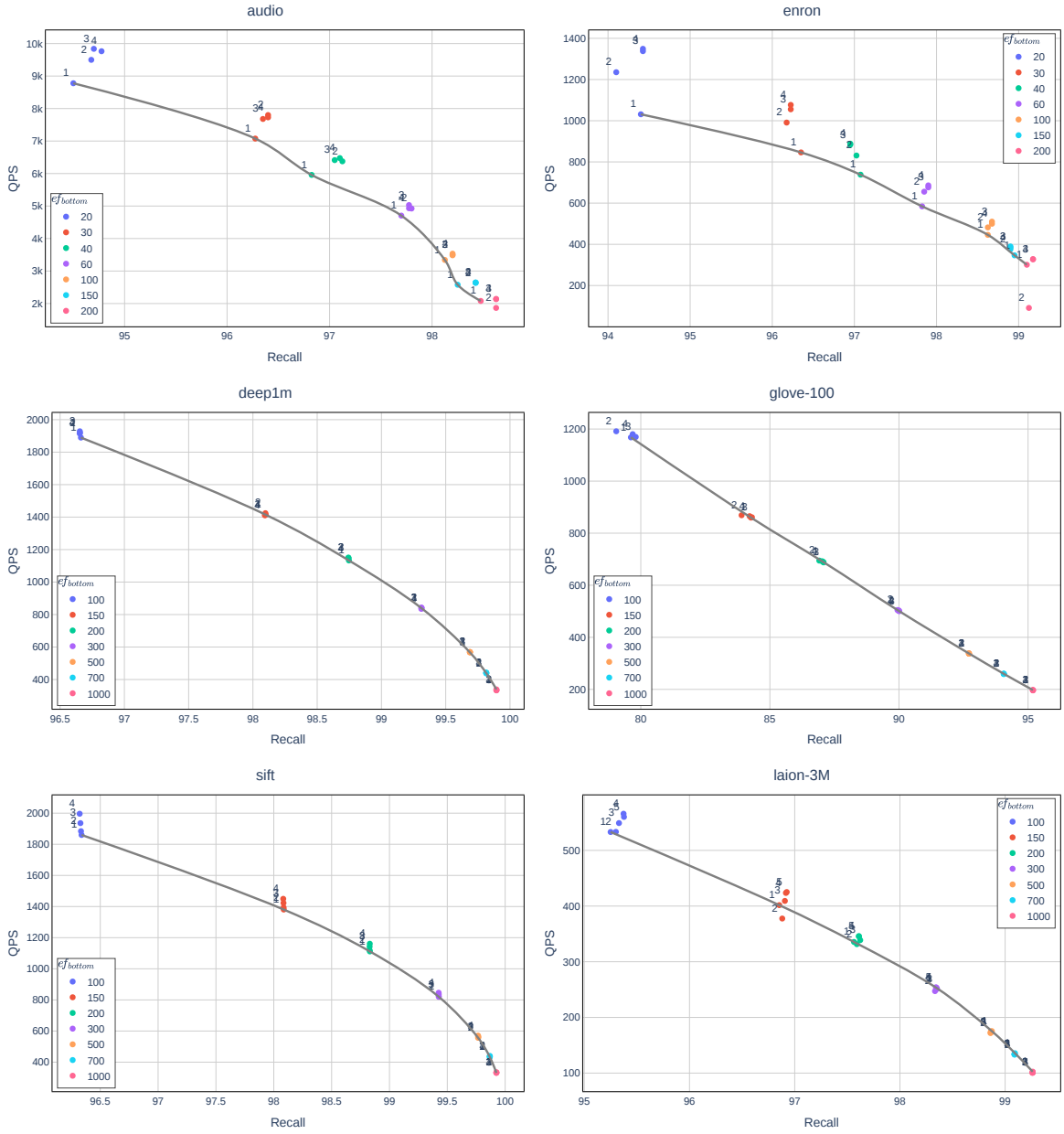

Figure 3: Level-wise QPS over Recall for HNSW on different datasets for different ef_{bottom}.

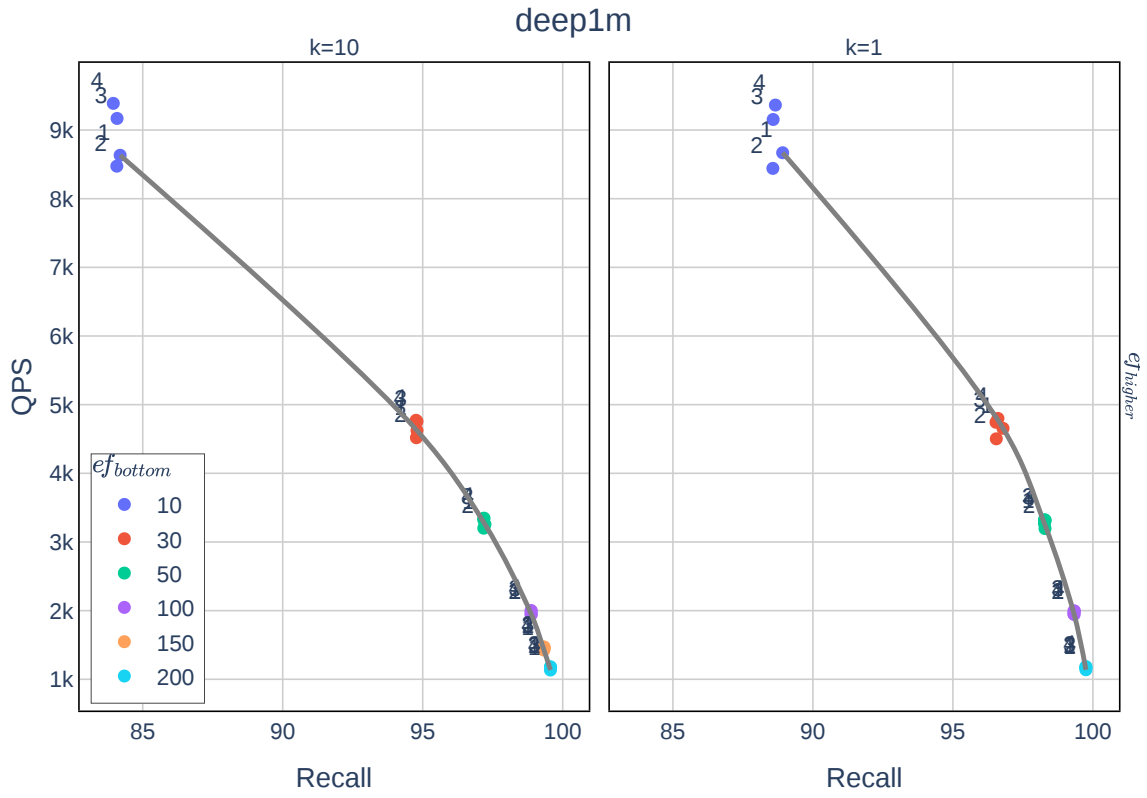

Figure 4: Level-wise QPS over Recall for HNSW on the DEEP-1M dataset with k=10 and k=1 and $ef_{higher} = 1$ for multiple ef_{bottom}. The number next to each point refers to the size of the level-stacked subgraph.

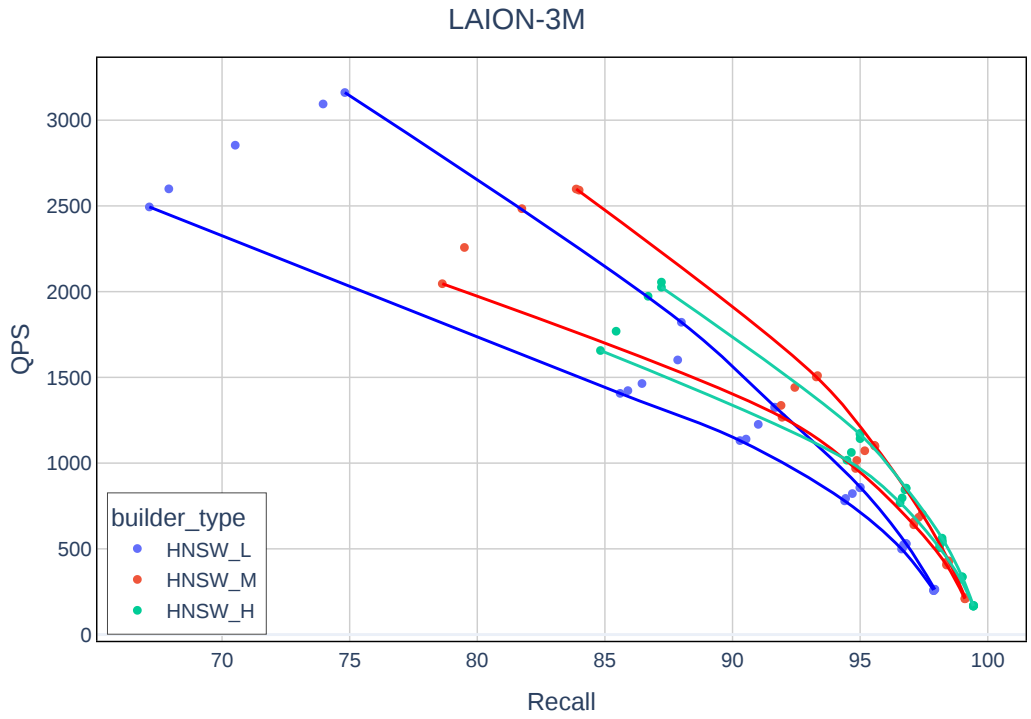

Figure 5: Level-wise QPS over Recall for three different HNSW graphs on the LAION-3M dataset for k=10 and $ef_{higher} = 1$ for multiple ef_{bottom}. The three graphs are differentiated in their parameters into low (L), medium (M), and large (L) degree tuned graphs. The curves span the respective graph's bottom and top-level performance, showing the closing gap for higher recall ranges.

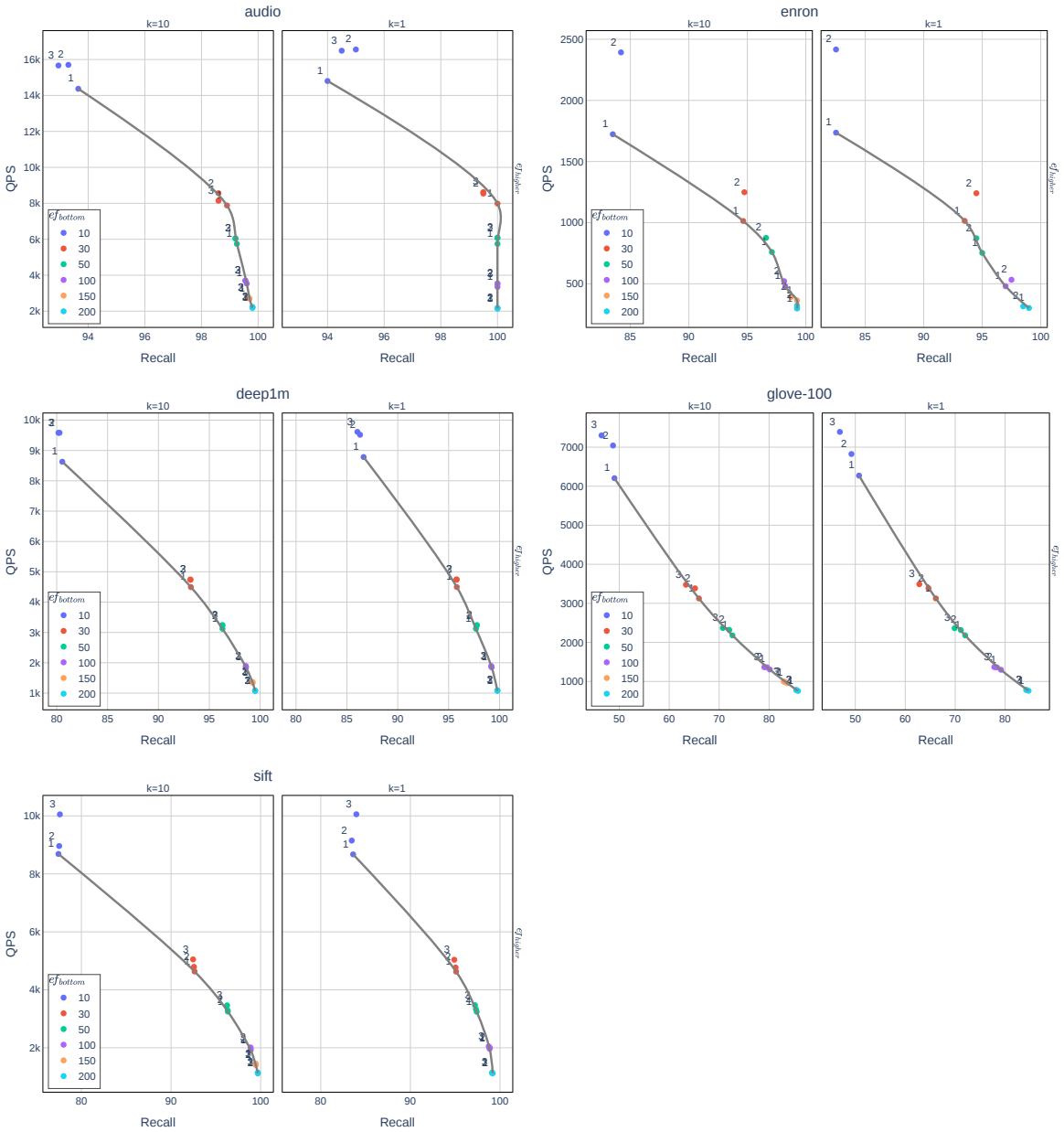

Figure 6: Level-wise QPS over Recall for HSGF-DEG-Flooding=2-Flooding=1 on different datasets for different ef_{bottom}.

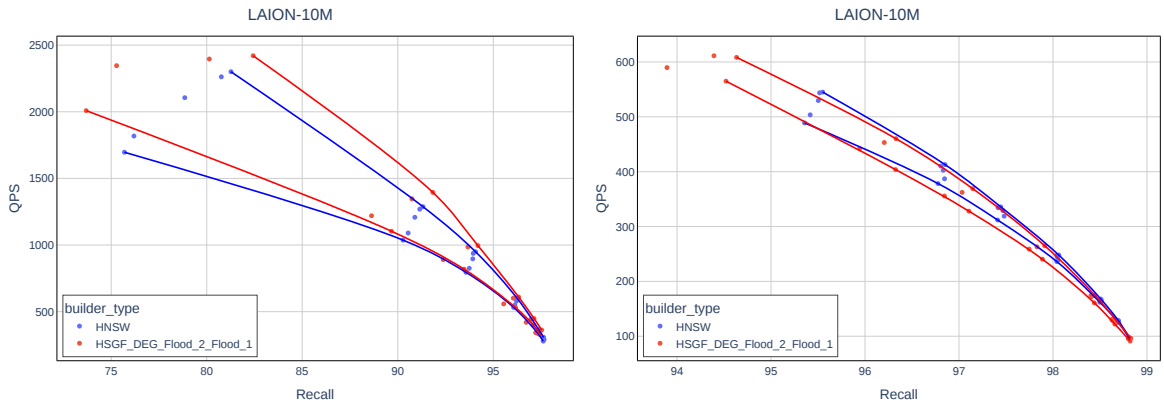

Figure 7: QPS over Recall for the HNSW vs the HSGF-DEG-Flooding=2-Flooding=1 on the LAION-10M, for each, the lower curve shows the search started from the bottom level, and the upper curve shows the search started from the top level. For k=10 (left) and k=100 (right).

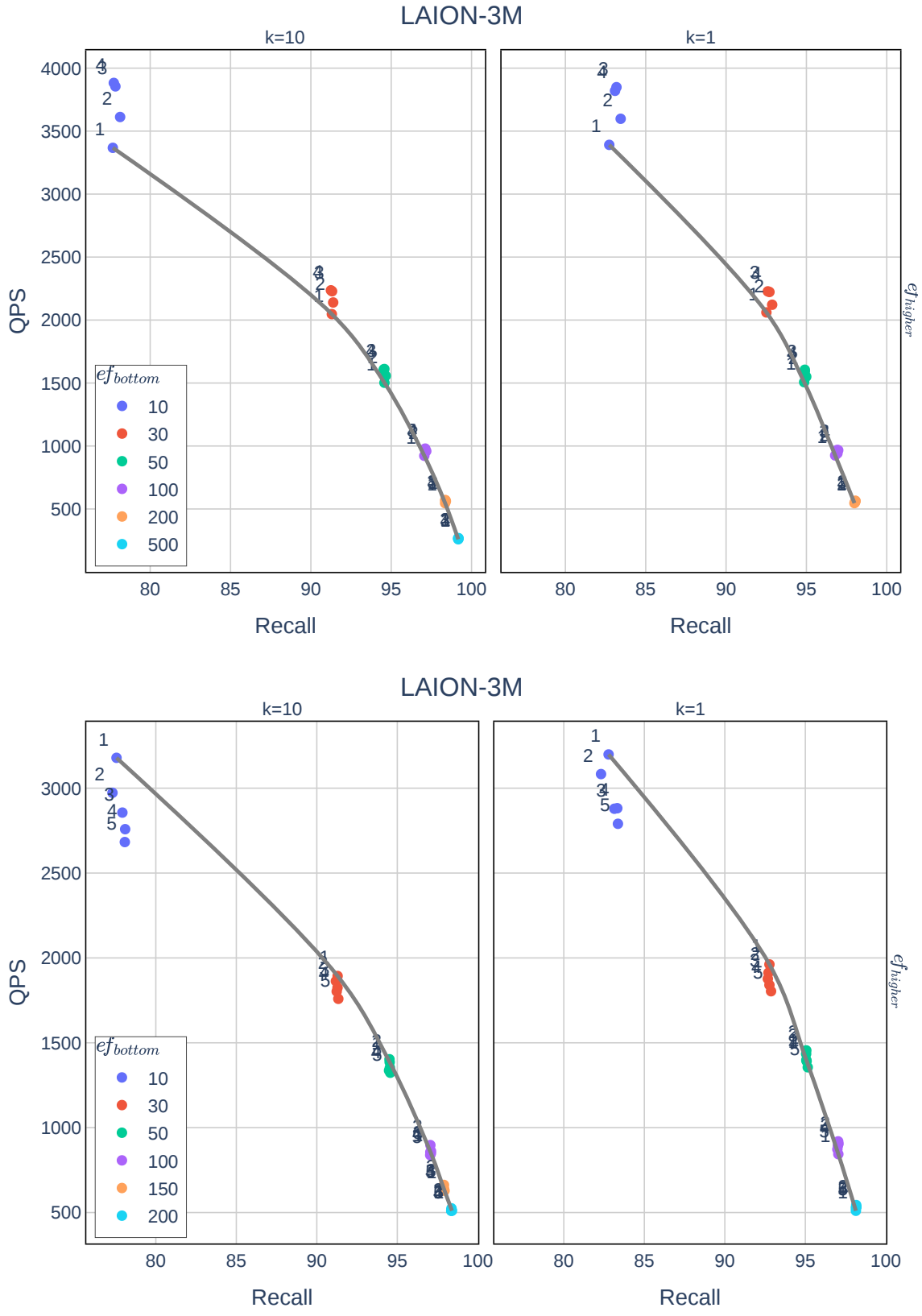

Figure 8: QPS over Recall comparing the HSGF-RNN-Rand (top) and HSGF-RNN-Rand-DEG-Rand (bottom) on the LAION-3M dataset for $k = 1$ and $k = 10$ for multiple ef_{bottom}.

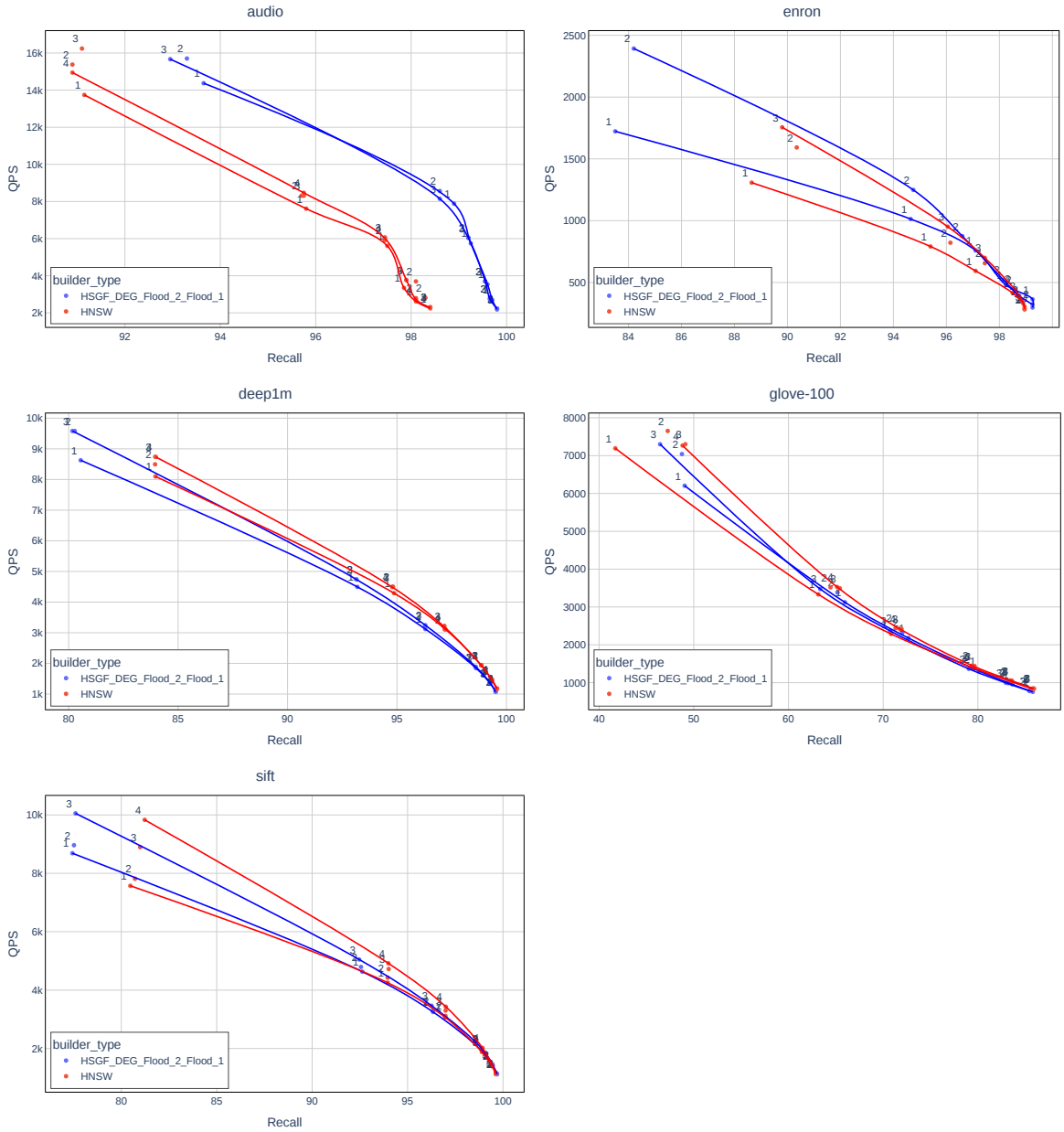

Figure 9: QPS over Recall comparing HSGF-DEG-Flooding=2-Flooding=1 and HNSW on multiple datasets for $k = 10$ and for multiple ef_{bottom}.

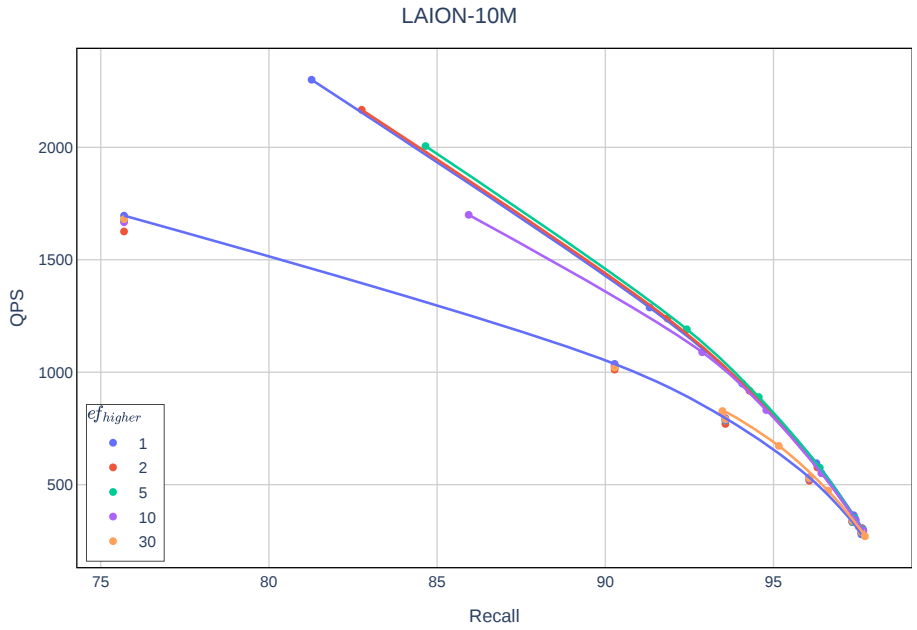

Figure 10: QPS over Recall for HNSW on the LAION-10M dataset with k=10, for ef_{higher}=[1, 2, 5, 30] and multiple ef_{bottom}=[10, 30, 50, 100, 200, 250]. Note, the vertically clustered points on the (blue) bottom level line are the result of minor deviations in the (computer-)system's performance, and therefore only impact the QPS but not the recall.

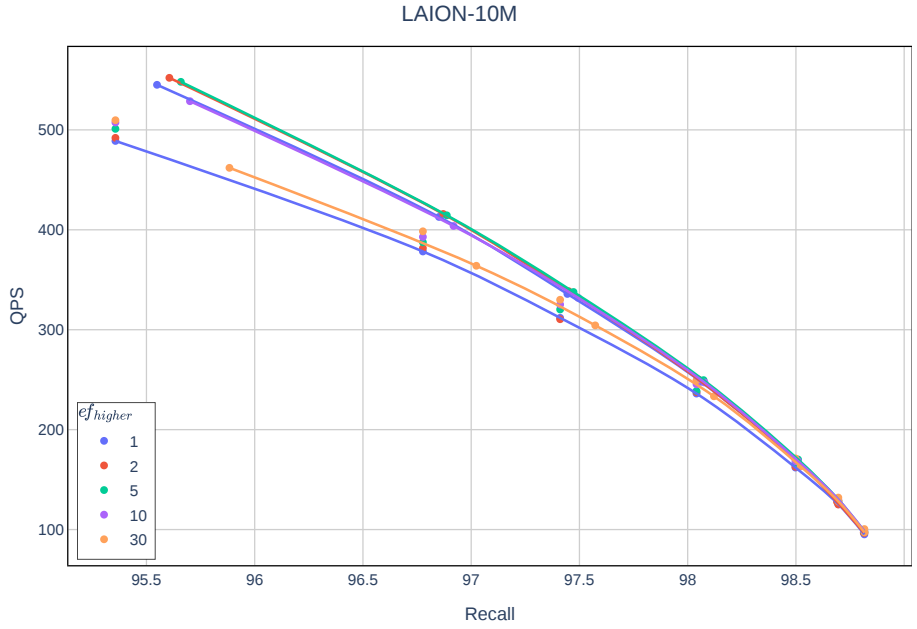

Figure 11: QPS over Recall for HNSW on the LAION-10M dataset with k=100, for ef_{higher}=[1, 2, 5, 30] and multiple ef_{bottom}=[100, 150, 200, 300, 500, 700, 1000]. Note, the vertically clustered points on the (blue) bottom level line are the result of minor deviations in the (computer-)system's performance, and therefore only impact the QPS but not the recall.

Verwendete Hilfsmittel

Folgende Hilfsmittel wurden beim Erstellen der Arbeit eingesetzt:

- Neben den klassischen, automatischen Tools zur Rechtschreibprüfung, wurde explizit Grammarly[1] verwendet. Dabei wurde Grammarly, aber erst zum Ende hin verwendet und nicht während des Verfassens, sodass keine KI-generativen Funktionen von Grammarly genutzt wurden.
- ChatGPT wurde nur über den Universitätszugang, und nur in äußerst selten Fällen genutzt, aber nie, um explizit Code oder Text zu generieren. Bspw. wurden Anfragen gestellt, um Plotly-Parameter herauszufinden oder um eine Erklärung für einige Zeilen C++ Code zu erhalten.

[1] https://www.grammarly.com/

Eidesstattliche Versicherung

(Affidavit)

Kolbe, Colin | 198201

Name, Vorname
(surname, first name)

Matrikelnummer
(student ID number)

☐ Bachelorarbeit
(Bachelor's thesis)

■ Masterarbeit
(Master's thesis)

Titel
(Title)

Bottom-Up Hierarchical Search Graph Generation

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, 01.08.2025

Ort, Datum
(place, date)

C. Kolbe

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, 01.08.2025

Ort, Datum
(place, date)

C. Kolbe

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**