Accurate and Efficient Metadata Filtering in Pinecone's Serverless Vector Database

Amir Ingber [1] **Edo Liberty** [1]

Abstract

Vector databases enable semantic search over large, dynamic datasets, supporting complex queries that combine vector similarity with metadata constraints. They are increasingly used in retrieval-augmented generation (RAG) systems, where accurate filtering over metadata – such as document type, user context, or recency – is essential to response quality. In serverless settings, where compute and storage are fully decoupled, this becomes especially challenging: data is continuously inserted and deleted, and metadata may be updated independently of the vector index, yet filters must be applied accurately and efficiently at query time.

This paper presents the design of metadata filtering in Pinecone's serverless vector database, which achieves high accuracy by integrating filtering into the vector retrieval path. Our architecture leverages immutable vector slabs organized in an LSM-tree structure in object storage, with stateless, on-demand executors that require novel coordination mechanisms to maintain correctness without tight coupling. We formalize accuracy through exact filter recall metrics and analyze two fundamental filter interaction paradigms: ad-hoc application versus pre-computed filter representations. We present results of filtered ANN search over a public filtered-search dataset (YFCC), as well as data from a production customer with categorical and numeric fields, demonstrating scalable performance while maintaining exact filtering accuracy.

[1]Pinecone, New York, NY, USA. Correspondence to: Amir Ingber <ingber@pinecone.io>.

Proceedings of the 1st Workshop on Vector Databases at International Conference on Machine Learning, 2025.

1. Introduction

Vector databases have emerged as critical infrastructure for modern AI applications, enabling semantic search over large-scale datasets through approximate nearest neighbor (ANN) algorithms. These systems support complex queries that combine vector similarity with metadata constraints, powering applications from recommendation systems to retrieval-augmented generation (RAG) where contextual filtering is essential for response quality. The vector database landscape is broadly divided between open-source solutions and managed services, each with distinct advantages. Open-source ANN packages, with FAISS (Douze et al., 2024; Johnson et al., 2019) being a primary example, offer flexibility and customization, allowing developers to tailor the system to specific use cases and maintain full control over which algorithm is used as well as its hyperparameters. However, they require significant operational overhead, from managing cluster scaling and data durability to optimizing performance across diverse workloads. Managed vector database services, in contrast, abstract away operational complexity while providing enterprise-grade reliability, security, and performance guarantees. Pinecone (Pinecone Systems, Inc., 2024a) represents this service-oriented approach, focusing on delivering high accuracy and reliability across thousands of simultaneous use cases with varying data characteristics, query patterns, and scale requirements. This multi-tenancy constraint introduces unique challenges: the system must maintain consistent performance and accuracy across diverse workloads while providing strong isolation and reliability guarantees. In this paper, we present Pinecone's approach to metadata filtering in serverless vector databases, where the challenge of maintaining high accuracy becomes particularly acute. In serverless architectures, compute and storage are fully decoupled, data undergoes continuous mutations, and metadata may be updated independently of vector indices. Yet applications demand high filtered-ANN accuracy, as the filters are many times crucial to the business logic and/or functionality of the downstream application. We focus on the interaction patterns between filters and ANN algorithms, and identify key challenges for future research in this rapidly evolving field.

2. Pinecone's Serverless Vector Database

Let $X \subseteq \mathbf{R}^d$ be a set of n vectors with a fixed and finite dimension d. In the classical problem of vector search, we are interested in finding the closest k vectors to a given query vector $q \in \mathbf{R}^d$. The closeness between two vectors is chosen according to the application, where cosine similarity and Euclidean distance are popular choices.

Pinecone's new serverless architecture (Pinecone Systems, Inc., 2024b) manages thousands of production-grade ANN indexes[1]. It is based on several key components:

Index partitioned into slabs. Every set of vectors is partitioned into non-overlapping parts called slabs. Each slab contains, in addition to the vectors, an internal *index*, which helps answering queries fast, as well as a *metadata index*, which takes care of processing the metadata filters. When a search query comes in, it is sent to all slabs, and results are merged before returned to the user. A slab is illustrated in Figure 1.

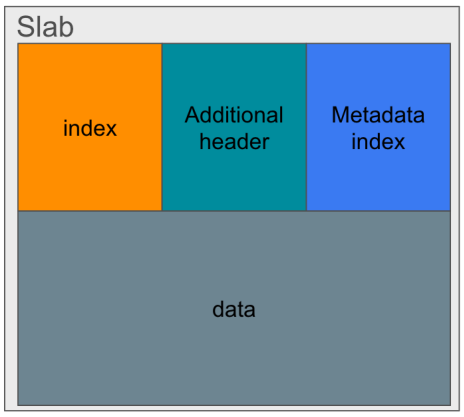


Figure 1. A single slab, containing the data (raw vectors), index (e.g. PQ codes and IVF centroids), and a metadata index for handling metadata filters. A slab also contains additional data, e.g. for mapping between global ids and slab-local vector ids, stored in an additional header.

Slabs are immutable. Slabs live in blob storage, and are assigned into levels. When new writes arrive, new slabs are generated and added to level 0. A *compaction* process runs in the background and merges smaller slabs into larger ones (e.g. slabs from level 0 into a larger slab in level 1 etc), in order to keep the overall number of slabs small. Deletions and updates are managed via a tombstones mechanism, which has its own compaction process, also running in the background. This is an adaptation of the classical LSM structure (O'Neil et al., 1996) for the vector search problem. The slab structure is illustrated in Figure 2.

Slabs may have different ANN algorithms. Smaller slabs are written and re-written multiple times by the compaction process, so their index is built fast using a random projection algorithm(Ailon & Chazelle, 2009). Larger slabs use more sophisticated indexing algorithms, such as InVerted Files (IVF), product quantization (PQ) (Jégou et al., 2011), as well as graph algorithms such as HNSW (Malkov & Yashunin, 2020). These algorithms are slower to build compared to the random projection algorithm, but allow much faster searches, especially for larger datasets.

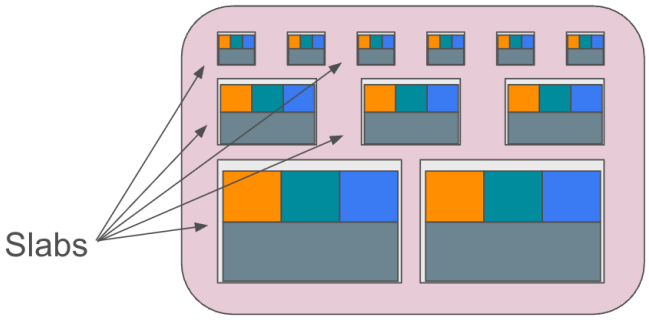


Figure 2. Slab structure in Pinecone's serverless vector DB. New vectors form new small slabs, which are merged in the background to larger slabs in lower levels.

[1] In pinecone terminology, an index is a collection of vectors, logically partitioned into *namespaces*. Vector insertions / deletions, as well as search queries are always addressed at a specific namespace. In this paper we will simplify matters and assume only a single namespace.

3. Metadata Filtering

3.1. Filtering model

The vector database contains, in addition to each vector, metadata, in the form of (key,value) pairs. The values can be (string) categories, or floating point numbers. For example, in an e-commerce application, where vectors represent items in a catalog for similarity search, the metadata may contain items such as price and product category. For example:

```
{"price": 10.0, "category": "toys"}.
```

Queries are comprised of the query vector as before, in addition to a *filter expression* that restricts the set of vectors in the index. For example:

```
{"price": {"$lte": 20.0},
"category": {"$in": ["toys", "electronics"]}}
```

represents a filter that restricts the price of the product to be ≤ 20, and the category to be either `toys` or `electronics`. In general, filters may be more complex, with `and`,`or` operators. The full specification of the filtering language pinecone supports is available online[2].

3.2. Measuring search accuracy

When searching for the k-closest matches without filters, ANN accuracy is typically measured using Recall@k, defined as the fraction of matches returned by the search index, out of the ground truth set: the top-k true closest matches (according to the exact similarity metric). In the filtered setting, the definition is similar, where now the ground truth is a set of closest vectors in the set, subject to the filter constraint. Note that the ground truth set may now contain fewer than k matches, in cases where there are not enough vectors in the set that match the filter.

3.3. Filtering architecture

During the ANN search process, there are mainly two ways to interact with the filter bitmap:

Predicate access. Here, given a vector and its metadata, return `true,false`. The predicate output is then used in the search process. This straightforward approach is simple to implement, but in many cases turns out to be slower than the next approach.

Precompute a match list. Given the filter expression, compute the list of all matching vectors. When possible, this is the preferred approach[3]. Moreover, even if the search algorithm only uses predicate access to the filter, pre-computing the list of matches often makes repeated predicate computations faster.

In the pinecone architecture, we precompute the list of matches for the filter. As mentioned in the previous section, each slab has data structure called a *metadata index*, which converts the metadata filter from a filter expression into an explicit representation of the vector ids that can be considered in this slab. That list of vectors is represented in a tightly compressed form – a bitmap. The bitmap allows not only fast predicate evaluations (e.g. "does vector id 123 match the filter"), but also direct iterator access to all the vector ids that match the filter.

As mentioned above, every slab (subset of an index) may use a different indexing algorithm. Next, we survey the main approaches we use for the different algorithms.

3.4. Pre-filtering for small slabs.

For smaller slabs, we use compressed vector approaches (e.g. PQ or random projections), where for every vector we store a compressed representation. The compressed representations are typically stored in the memory of the nodes that serve the queries, and allow fast point-reads into them. So in query time, we iterate over the ids of the matching vectors, and only scan their compressed representations in order to compute the approximate distance to the query. In order to maintain high search recall, we typically keep more than the required k vectors in a heap-like structure, and re-rank these based on a more accurate vector representation, which typically resides on disk.

3.5. Filters and IVF indexes

For larger slabs, we may use an indexing algorithm such as IVF, where the vectors in the slab are partitioned into clusters based on cetroids (e.g. using the well-known k-means algorithm for finding the centroids). In query time, we typically find the closest `nprobe` clusters, and scan ("probe") only the vectors only in these clusters. The process is illustrated in Fig. 3. Within the selected clusters, we scan the compressed representation (e.g. PQ) for all the vectors. Tuning `nprobe` in order to control the tradeoff between latency and recall is typically done by hand, with representing datasets.

Simple application of filtering in IVF usually does not work. When filters become more strict, leaving fewer matching vectors, the accuracy generally drops. This is illustrated in Fig. 4a, where 50% of the vectors are filtered out, and the recall drops. When 90% of the vectors are filtered out (see Fig. 4b), fewer vectors from the ground truth set remain in the set of scanned vectors, and the recall becomes even lower, and the search results are unusable.

[2] `https://docs.pinecone.io/guides/search/filter-by-metadata`

[3] In more complex filtering models, e.g. regex matches on text, this is harder to achieve.

In order to make IVF-type indexes compatible with filtering across all selectivity ranges, we make several key modifications to the IVF query process, based on the filter.

IVF bypass. If the filter is very restrictive, matching less than some prescribed threshold of vectors, we skip the IVF process altogether, and scan all the matching vectors. The latency is controlled by the threshold, and the recall remains high, by design.

Global scan fraction. Instead of choosing a number of clusters to scan in advance (a.k.a. `nprobe`), we choose a scan fraction (e.g. 0.1), and keep scanning partitions until their total size covers at least this amount of vectors from the slab. The reason is that the clusters in IVF are usually not very balanced (there are ways to encourage this but at the expense of accuracy). We want that if, for some query, the first 5 clusters turn out to be small, then keep scanning more clusters. And on the other side of the spectrum, if for a query the first few clusters are already much larger than average, then the recall improvements from scanning more clusters is minimal.

Adaptive scan fraction. We adapt the scan fraction according to the filter selectiveness. For instance, if we see that 50% of the vectors are filtered out, then we increase the scan fraction for that query. This process is illustrated in Figures 5a and 5b. The increase mechanism is designed empirically. We have found the that the following form for the scan fraction is effective:

$$\texttt{scan\_fraction} = f_0 \cdot \sigma^{-\alpha},$$

where σ is the filter selectivity, and f_0 is the scan fraction that applies when there is no filter. The positive parameter α is chosen empirically.

Minimal matches. To account for edge cases, we also impose a minimal number of matching vectors that must be scanned.

3.6. Filters and graph indexes

Graph-based ANN algorithms have become increasingly popular, including HNSW (Malkov & Yashunin, 2020), DiskANN (Subramanya et al., 2019) and other variants. Integrating filters with a graph index has always been a challenge, since applying the filter during navigation typically harms the connectivity of the search graph, so it needs to be carefully tuned. Some authors attempted to incorporate the metadata into the graph build process, e.g. Filtered-DiskANN (Gollapudi et al., 2023). However, this approach does not extend to general filter models, where the filters are more complex and/or cannot be characterized in advance.

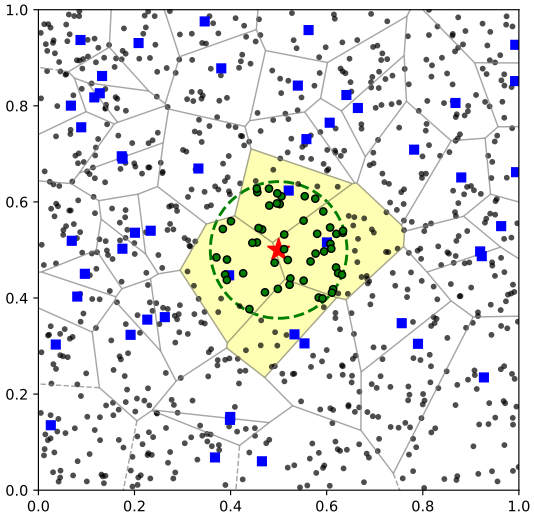


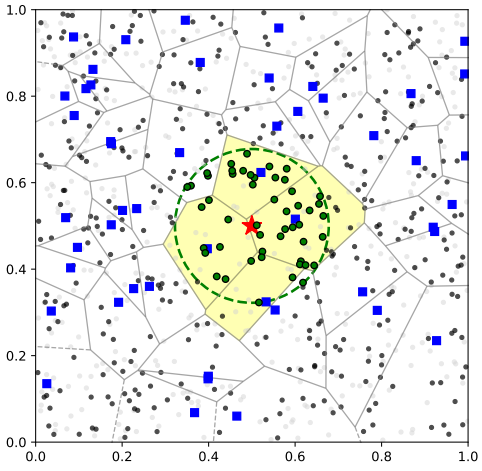
Figure 3. Standard IVF ANN search without filters. Red star: query point. Squares: centroids of each cell. In the search process, we scan the nprobe=4 closest clusters. In this example, the top-k points (marked in green) are indeed within the searched clusters, resulting in good recall.

Recently, methods that are so-called predicate-agnostic have been proposed (Patel et al., 2024), where the graph is built by artificially increasing its density using different methods. In this method, the filter is applied many times during the navigation process.

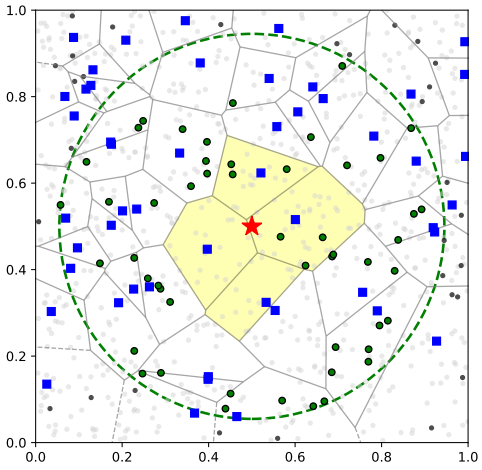
One approach that we have found to work robustly enough is to not use the filters in the initial navigation process: first, search for the closest point, and from that point, continue scanning until enough points that match the filter have been accumulated. This is the approach used, for instance, in the `p2` type pod[4]. As a reminder, the pod-based vector database is different than the serverless one, where the graph is always kept in memory. This makes the graph algorithm a good choice only for some use cases, typically high QPS where a server is always kept running, and more care is needed when planning the size of the server against the expected load.

Improving filters for graph indexes in a robust manner for a serverless architecture is a topic of active research in the community, as well as within Pinecone. We will elaborate on the specific research questions at the end of the paper.

[4] `https://www.pinecone.io/blog/hnsw-not-enough/`

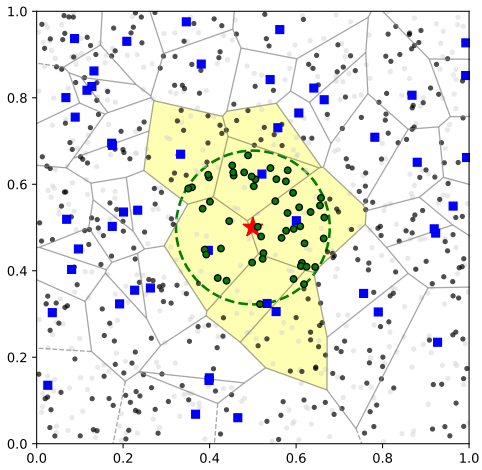


(a) 50% of the vectors are filtered out. Mild drop in recall, as some of the true nearest neighbors (green points) are outside the scanned area.

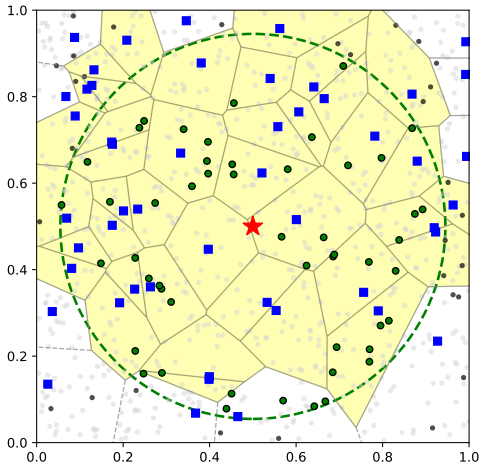


(b) 90% of the vectors are filtered out. Large drop in recall, as many of the true nearest neighbors fall outside the scanned area.

Figure 4. Naive IVF with filtered ANN. Recall drops as the filters become more selective.



(a) 50% of the vectors are filtered out. nprobe increased from 4 to 7. Recall stays high.



(b) 90% of the vectors are filtered out. nprobe increased from 4 to 45. Recall stays high.

Figure 5. Adaptive IVF with filtered ANN. No recall drops, regardless of the filter.

4. Experimental results

We present detailed recall results of two indexes. In this benchmark, we are mostly interested in the recall for a broad range of selectivity values.

4.1. Datasets

We benchmark two different datasets. The first one, denoted YFCC, is the formal dataset for the filter track in the 2023 BigANN competition (Simhadri et al., 2024). It comprises of 10M 192-dimensional vectors, representing images embedded with a variant of the CLIP model. Every vector is attached to several *tags*, representing words in the description of the image, as well as other image categories, such as camera model and the country. The queries represent image embeddings (for image similarity search), and the metadata filter is one or two tags that must appear in the vector tags. The filter selectivity is spread uniformly (in log scale) between 10 vectors all the way to around 2M vectors.

The second dataset is based on data from a pinecone customer, comprising of 35M vectors in a search application. The metadata includes both categorical values, as well as numerical values. The query set is based on actual production search queries. The filter uses a combination of `$in` operation for a categorical field, as well as a range filter on a numeric field. The query selectivity in this representative query set is dominated by a single type of filter ($> 60\%$), with selectivity of around 18%. The dataset also contains more restrictive queries. See Table 1 for more details about both datasets.

4.2. Benchmark setting

For both datasets, we uploaded them into a pinecone serverless index in the eu-west AWS region. We used the bulk import capability[5], which allows uploading large datasets from blob storage. Queries were sent sequentially using the python SDK.

The pinecone indexes are tested with their default settings, without any internal or external modification.

4.3. Results

For YFCC, the mean recall@10 was 0.989. Figure 6 shows that the recall remains high for all ranges of selectivity, without a drop in either extremes of the selectivity spectrum. The internal query latency (excluding round-trip time to the client) was around 20ms. The latency was measured on a second of two subsequent runs, to make sure that slabs are loaded to the executor (a.k.a. warm slabs).

[5] `https://docs.pinecone.io/guides/index-data/import-data`

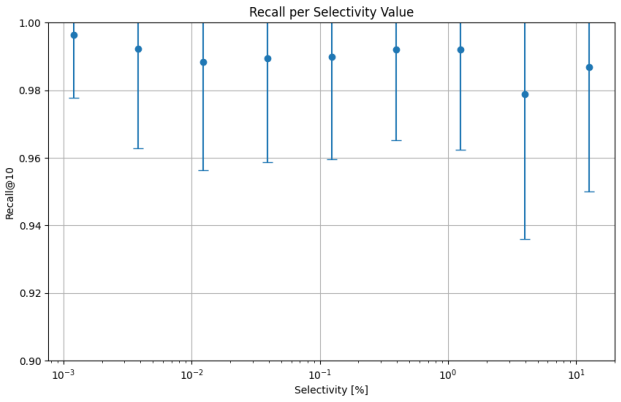


Figure 6. YFCC dataset: Recall@10 vs. query selectivity. For each selectivity range, we show the mean and variance of the recall per query.

For the customer dataset, mean recall@100 was also high: 0.986. Figure 7 shows recall@100 across selectivity ranges for the customer dataset. For the dominant selectivity value, the mean recall was the lowest, around 0.981 with small variance. Overall, the recall remained extremely high. The internal search latency was around 75ms. The reasons for the increased latency are (1) the bigger size of the dataset, and (2) the more complex filter (range filter).

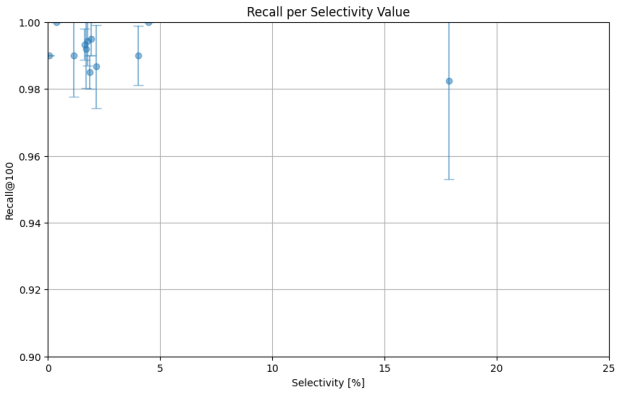


Figure 7. Production dataset: Recall@100 vs. query selectivity. For each selectivity value in the query set, we show the mean and variance of the recall per query. A large number of queries have the same filter, with around 18% selectivity.

5. Summary and Future Work

We reviewed Pinecone's latest serverless architecture, focusing its metadata filtering capabilities. We described two benchmarks of datasets in the 10M vector scale, with different filtering characteristics. For all cases, the search accuracy, measured in recall, remained extremely high (98%). For future research, we would point out the topic of robust designs of graph ANN algorithms that support filters, both

Dataset	Dimension	Similarity	Top k	Vectors	Queries	Query Selectivity[%]				
						min	p10	p50	p90	max
YFCC	192	Euclidean	10	10M	1000	0.0006	0.002	0.14	7	19
Customer data	768	Cosine	100	35M	100	0.07	1.7	17.8	17.8	17.8

Table 1. Dataset and query statistics

metadata aware and agnostic. For example, given a simple filter scenario of m categories, known in advance, how can we design a graph algorithm to answer effectively queries with filters on the *intersection* of categories?

References

Ailon, N. and Chazelle, B. The fast johnson–lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on Computing*, 39(1):302–322, 2009. doi: 10.1137/060673096. URL https://doi.org/10.1137/060673096.

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., and Jégou, H. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

Gollapudi, S., Karia, N., Sivashankar, V., Krishnaswamy, R., Begwani, N., Raz, S., Lin, Y., Zhang, Y., Mahapatro, N., Srinivasan, P., Singh, A., and Simhadri, H. V. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference 2023*, pp. 3406–3416. ACM, 2023. doi: 10.1145/3543507.3583552.

Jégou, H., Douze, M., and Schmid, C. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7 (3):535–547, 2019.

Malkov, Y. A. and Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

O'Neil, P., Cheng, E., Gawlick, D., and O'Neil, E. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.

Patel, L., Kraft, P., Guestrin, C., and Zaharia, M. Acorn: Performant and predicate-agnostic search over vector embeddings and structured data. In *Proceedings of the 2024 International Conference on Management of Data*, pp. 2741–2756. ACM, 2024.

Pinecone Systems, Inc. Pinecone vector database, 2024a. URL https://www.pinecone.io/.

Pinecone Systems, Inc. Pinecone serverless architecture. https://docs.pinecone.io/reference/architecture/serverless-architecture, 2024b. Accessed: 2025-05-30.

Simhadri, H. V., Aumüller, M., Ingber, A., Douze, M., Williams, G., Manohar, M. D., Baranchuk, D., Liberty, E., Liu, F., Landrum, B., Karjikar, M., Dhulipala, L., Chen, M., Chen, Y., Ma, R., Zhang, K., Cai, Y., Shi, J., Chen, Y., Zheng, W., Wan, Z., Yin, J., and Huang, B. Results of the big ann: Neurips'23 competition. *CoRR*, abs/2409.17424, 2024. URL https://arxiv.org/abs/2409.17424.

Subramanya, S. J., Devvrit, F., Simhadri, H. V., Krishnawamy, R., and Kadekodi, R. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *Advances in Neural Information Processing Systems*, volume 32, pp. 13748–13758, 2019.