Filtered Vector Search: State-of-the-art and Research Opportunities

Yannis Chronis
ETH Zurich
chronis@ethz.ch

Helena Caminal
Systems Research Group, Google
hcaminal@google.com

Yannis Papakonstantinou
Google Cloud
yannispap@google.com

Fatma Özcan
Systems Research Group, Google
fozcan@google.com

Anastasia Ailamaki
EPFL
anastasia.ailamaki@epfl.ch

ABSTRACT

This tutorial provides a comprehensive overview of filtered vector search (fvs). Fvs queries combine vector search with relational operators. The tutorial explores the challenges of integrating vector search into database engines and emphasizes the need for new optimization techniques. It explains the three primary filtered search methods for fvs queries over generic tree-based and graph-based indices and examines the factors influencing the selection of the most efficient method. A key objective is to highlight the importance of achieving stable recall, ideally in a declarative manner, ensuring consistent recall across queries. The tutorial then discusses recent filter-optimized vector indices and concludes by identifying open research challenges in the field of fvs, aiming to inspire further research and development.

PVLDB Reference Format:
Yannis Chronis, Helena Caminal, Yannis Papakonstantinou, Fatma Özcan, and Anastasia Ailamaki. Filtered Vector Search: State-of-the-art and Research Opportunities. PVLDB, 18(12): 5488 - 5492, 2025.
doi:10.14778/3750601.3750700

1 INTRODUCTION

The advancements of embedding models [10, 21, 24], the power of semantic search [11, 16], and the importance of RAG for improving and grounding LLM responses [1, 15] drive the integration of vector search into database engines. Storing and searching vectors in databases enables the combined querying of unstructured data (represented as vector embeddings) and the structured data that are already stored in databases. New optimization techniques, execution methods and data structures are needed to achieve high performance with high accuracy. Today, numerous commercial database systems have added support for vector search [6, 13, 29, 35, 40, 43], specialized vector databases have been introduced [4, 8, 38], and there is also significant interest from the academic and open source communities [2, 7, 20, 23, 25, 27, 31, 36, 39].

This tutorial elaborates on filtered vector search (fvs), the building block of queries that combine vector search with relational operators [20, 31, 40]. In filtered vector search, relational operators (or filters) on structured data restrict the database rows considered by the vector search. For example, an e-commerce search may allow filtering by the brand or price range while searching for similar products to a user provided description.

Vector search is commonly accelerated by vector indices [5, 25, 27, 36, 37] that perform approximate near neighbor search (ANN). ANN search trades-off latency for recall by reducing the number of vectors that are visited during each index traversal. The alternative is to calculate the similarity of every vector with the query vector and sort them (i.e. K nearest neighbor search or KNN). Vector indices can achieve substantial latency savings while achieving high recall but require careful tuning of the index creation and search [27, 37].

A vector search execution method tuned for unfiltered queries will fail to achieve high recall when filters are added [38, 40, 44]. Filters change the size and distribution of qualified data by eliminating vectors and, in certain cases, introducing correlation between the filter predicates and the vector space [31]. Filtered vector search algorithms should offer stable recall, namely achieve the same recall for all queries regardless of the filter predicates. Ideally, from the user's perspective the tuning should be declarative. In other words, the user specifies the target recall and not how to tune the execution to achieve it (i.e. how many leaves of a tree vector index to search [37]). Naturally these goals should be achieved with performance (latency and resource usage are low). To this end, innovation is needed in two areas: a) query optimization and b) filter-optimized vector indices.

Filter-optimized vector indices aim to approximate an index built exclusively from the vectors that satisfy the filters [20, 39–41]. An index built only for the vectors that satisfy each filter would minimize the search effort and avoid low recall results [30, 31], but is infeasible to build in practice, unless the queries always feature a very special filter type. For example, if all queries feature an equality condition on a categorical attribute, then the optimum strategy is to have a partitioned index, i.e., one index for each value of the categorical attribute. Proposed solutions trade-off latency, space or generality to improve the latency to recall balance they achieve. There are three components that can be changed to optimize an index for filters: the search algorithm [31], the construction [20, 40] or the similarity metric [39, 41]. The requirements of database workloads and information retrieval use cases are not the same. We categorize the proposed indices, which are motivated by information retrieval use cases, based on the types of filter predicates, the size of the filter results, and the number of structured attributes

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750700

they support [20, 31]. More recent approaches achieve competitive performance while being filter agnostic [31].

This tutorial covers filtered vector search, discusses the use cases that make a timely research area, provides a classification of existing approaches and discusses their trade-offs, and sets the goals for future solutions.

Tutorial Overview This tutorial is organized in five parts and the intended length is 1.5 hours.

- (a) **Brief Background: Why and how do we search vectors?**
  - (i) Essentials of vector approximate nearest neighbor search
  - (ii) How do Vector Indices accelerate vector search?
    - (1) Classification (Tree, Graph, Hashing)
  - (iii) Quantization
- (b) **Introduction to Filtered Vector Search**
  - (i) Searching structured data + vectors
  - (ii) Categorization of approaches
- (c) **Filtered Vector Search using Conventional Indices**
  - (i) Execution methods
  - (ii) Recall Challenge
    - (1) When to stop searching to achieve a target recall?
    - (2) Selectivity, Correlation: How do they affect vector search?
  - (iii) Performance Challenge
  - (iv) Navigability Challenge in graphs indices
- (d) **Filter Vector Search using Specialized Indices**
  - (i) Specialized Index Structures
    - (1) Predicate agnostic
    - (2) Predicate aware
  - (ii) Special Distance Functions
- (e) **Stable/Declarative Recall, Ease of Use**
- (f) **Query Optimization and Planning**
  - (1) Vector Search stopping conditions
  - (2) Choice of vector search execution and filter evaluation method
- (g) **Research Challenges**

Target Audience, Assumed Background, Related Tutorials This tutorial is intended a) for database researchers and practitioners interested in understanding and advancing the state-of-art techniques for filtered vector search and b) for PhD students who are seeking a high-impact research topic in this area.

There are no prerequisites beyond a basic understanding of database concepts, embeddings and brute-force vector search (KNN). The goal of this tutorial is threefold: (i) First, to introduce the different challenges of achieving high recall with good performance for database filtered vector search queries (ii) Present and classify the state-of-the-art in filtered vector search indices and execution methods (iii) Present open research opportunities.

There is active interest in the research community for vector search [12, 19, 34, 42, 45]. In the past five years, there have been tutorials on similarity search techniques [17, 18, 32], vector search for LLM RAG [9], and vector DB management techniques and systems [30]. Our tutorial aims to complement these tutorials by focusing on filtered vector search queries and inspire the design of filter-optimized vector search methods indinces.

2 THE TUTORIAL

2.1 Vector Search Background

The first part of the tutorial covers relevant background: a) definition of vector search, b) presentation of motivating use cases (RAG [1, 15], Semantic Search [11, 16], Recommendation Systems [16]), b) exact and approximate algorithms (KNN and ANN) [30]), c) classification and comparison of vector index structures (Tree-based indices [14, 22, 33], Graph-based Indices [27, 36, 39]), Hash-based indices [25] d) quantization techniques [28].

Vector search ranks vectors based on their distance from a query vector (most often returns the top-k vectors)[1]. The result can be exact or approximate, depending on whether a K nearest neighbor (KNN) or an approximate nearest neighbor (ANN) search is used [30]. Vector indices are used to perform ANN search and trade-off latency for recall by reducing the number of vectors considered during each search. This trade-off is controlled directly or indirectly by a set of index specific tuning knobs [20, 22, 27, 31].

2.2 Filtered Vector Search

Filtered vector search (fvs) queries restrict the database rows that are considered by the vector search. In the simplest case, the filters are atomic. In the richer case, they may also be nested subqueries. Expanding to joins with conditions on "dimension" tables fundamentally reduces to the nested subquery case.

Execution methods There are three main execution methods for executing filtered vector search queries depending on the order of operations (filter and vector search) and the vector search type. Prefiltering, where data is filtered and then KNN search is performed on the result of the filter. This method is preferred when the filter result is small [30]. The remaining two execution methods use a vector index to perform ANN vector search. Post-filtering, where the filters are evaluated on the vector index search result [40]. For inline-filtering, searching and filtering are combined, and the filters can be evaluated before the vector search starts (and stored in a bitmap) or evaluated during the vector index search.

Unsurprisingly, the three methods differ in terms of performance, and in practice also in terms of recall. Tuning the vector index search to achieve the same recall for the same query across all methods is challenging. For instance, post-filtering, in simplified approaches to filtered search, requires the approximate nearest neighbor (ANN) search to yield a multiple of K results to ensure at least K vectors remain after filtering [40], which complicates achieving high recall. This introduces an extra tuning parameter compared to inline filtering

Impact of Filters Filters modify data size and distribution by removing vectors and potentially introducing correlation between the filter and the vectors [31]. This impacts the number of vectors needed for a vector index search to reach a specific recall level compared to an unfiltered search. For instance, with a positive correlation, nearby vectors are more likely to pass the filter, reducing the number of vectors that need to be visited by a vector index

[1]A threshold on the maximum similarity distance can be additionally set.

search for high recall. Conversely, a negative correlation increases this number. Selective filters also increase the search effort [20, 39]. The optimal execution method depends on the relative costs of filtering, ANN vector index search, and KNN search. These costs are influenced by data and query characteristics such as selectivity, correlation, k, and data distribution [31, 40].

Stable and Declarative Recall Performance stability and a declarative interface have been at the core of database design. Stable and declarative recall, extend these concepts to vector search.

In the absence of filters, basic heuristics have been employed to tune vector index searching [37]. Since filters make the optimization of an execution method more complex, designing systems and algorithms that ensure stable recall is crucial. Stable recall: achieve consistent recall across queries, regardless of filter conditions or the execution method. The challenge lies in the fact that filter selectivity and correlation are notoriously difficult to predict during query optimization, yet these factors ultimately determine the necessary search effort [26]. Recently PostgreSQL proposed an adaptive approach to decide at runtime when the vector search should stop[2]. AlloyDB [6] also follows an adaptive approach.

Ideally, a user should simply declare their desired target recall, and let the database configure all the parameters to achieve it [3]. This approach eliminates the need for users to understand the intricacies of the data and queries, and the complexities of vector index tuning.

2.3 Filter-Optimized Vector Search Indices

The efficacy of vector indices depends on their ability to minimize the number of vectors that need to be searched to achieve a specific recall while applying the filters. The ideal index is one that is built only for the database rows that satisfy a set of filters. Building indices for all possible attribute values is infeasible, and existing approaches try to approximate this goal by creating value-induced neighborhoods, where vectors with the same (or similar) structured attribute values are connected[3]. Thus, during a vector search, the effort to find all vectors that satisfy a filter is small and the probability of low recall because of not reaching enough number vectors that satisfy the filters is smal. FilteredDiskANN [20] creates multiple graphs and merges them significantly modifying the index build step. NHQ incorporates the structured attributes in the vector similarity and requires less changes to the build or search algorithms [39, 41]. On the other hand, ACORN is designed to be filter-agnostic where it physically or virtually augments the HSNW index construction to guarantee navigability when filters eliminate graph nodes [31].

Proposed approaches can also be classified by their filter support. Some support only equality predicates [20, 39, 41] while [46] supports range predicates. Further, some approaches impose limits on the number of structured attributes each datapoint can have [39, 41] or the filter result cardinality [20]. A number of existing approaches get inspiration from from the information retrieval space, where the number of structured attributes and the number of unique values is limited. Notably, to the best of our knowledge, all fvs targeted optimizations that change the structure of an index for a set of filter attribute values are for graph indices.

[2]https://github.com/pgvector/pgvector/issues/678

[3]Orthogonally, one may create a partitioned index if a categorical filter is known in advance.

2.4 Research Opportunities

In addition to the stable and declarative recall challenges discussed in 2, there are many open challenges at the intersection of structured data, vector search and, more broadly, search. We highlight here a subset of them.

Benchmarks Pure vector search has benefited greatly from well-adopted vector search benchmarks. It is imperative that respective benchmarks emerge for filtered vector search.

Hybrid Search Semantic search is often combined with classic full text search. The combination remains relevant when semantic search is filtered vector search. Then, interesting opportunities emerge by exploiting the full text search for providing a “soft" alternative to filtering for certain subsets of filter conditions. For example, the full text search index is often expanded to include structured data, allowing filtered searches to be reduced to “boosted" text searches or even proper filtered searches, thus providing an alternate path towards specialized indexing.

Auto Configuration Many of the options we presented in this tutorial will eventually be incorporated in databases featuring vector search, as well as in purpose-built vector databases. This, in turn, will create a usability problem: What index should be used and how to tune its parameters? What execution method is best? The database community, with its long tradition of automated optimization, can create important innovations.

3 PRESENTERS

Yannis Chronis is a Researcher at the Systems Research@Google. He currently works on efficient filtered vector search architectures for Google’s database products. Yannis will be joining ETH Zurich in the summer of 2025 as an assistant professor; he received his PhD from the University of Wisconsin-Madison.

Helena Caminal is a Researcher at the Systems Research@Google. She is interested in problems at the intersection of database, ML, and hardware. Helena has worked on leveraging Associative Processing from the 1970s to design SRAM-based parallel processors used to accelerate database analytics. Helena received a PhD from Cornell University in 2022.

Anastasia Ailamaki is a Professor of Computer and Communication Sciences at EPFL, and a visiting researcher at Google. A recipient of the 2019 ACM SIGMOD Edgar F. Codd Innovations Award and the 2020 VLDB Women in Database Research Award, she earned her Ph.D. in Computer Science from the University of Wisconsin-Madison in 2000. She is an ACM fellow, an IEEE fellow, a member of the Academia Europaea, and an elected member of several National Research Councils.

Fatma Özcan is a Principal Engineer at Systems Research@Google. Her current research focuses on LLMs and ML for databases, text2SQL and conversational interfaces to data, platforms and infrastructure for large-scale data analysis. Dr Özcan got her PhD from University of Maryland, College Park and has over 23 years of experience in industrial research. She received the VLDB Women in Database Research Award in 2022. She is an ACM Fellow, and the vice chair of ACM SIGMOD.

Yannis Papakonstantinou is a Distinguished Engineer, working on Query Processing and GenAI, at Google Cloud. He is also an Adjunct Professor of Computer Science and Engineering at the University of California, San Diego, following many years of having been a UCSD regular faculty member. Previously he was an architect in query processing & ETL at Databricks. Earlier, he was a Senior Principal Scientist at Amazon Web Services from 2018-2021 and was a consultant for AWS since 2016. He has published over one hundred twenty research articles that have received over 21,000 citations. Yannis holds a Diploma of Electrical Engineering from the National Technical University of Athens, MS and Ph.D. in Computer Science from Stanford University (1997).

REFERENCES

[1] 2023. Brie Wolfson. Building chat langchain. https://blog.langchain.dev/buildingchat-langchain-2/)).

[2] 2025. Facebook FAISS. https://github.com/facebookresearch/faiss.

[3] 2025. Oracle Vector Search Manual,. https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/ai-vector-search-users-guide.pdf.

[4] 2025. Pinecone. https://www.pinecone.io/.

[5] 2025. ScaNN. github.com/google-research/google-research/tree/master/scann.

[6] 2025. ScaNN for AlloyDB,. https://services.google.com/fh/files/misc/scann_for_alloydb_whitepaper.pdf.

[7] 2025. SPTAG: A Library for Fast Approximate Nearest Neighbor Search. https://github.com/Microsoft/SPTAG.

[8] 2025. Weviate. https://weaviate.io/.

[9] Akari Asai, Sewon Min, Zexuan Zhong, and Danqi Chen. 2023. Retrieval-based language models and applications. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 6: Tutorial Abstracts).* 41–46.

[10] Oren Barkan and Noam Koenigstein. 2016. Item2vec: neural item embedding for collaborative filtering. In *2016 IEEE 26th international workshop on machine learning for signal processing (MLSP).* IEEE, 1–6.

[11] Fedor Borisyuk, Siddarth Malreddy, Jun Mei, Yiqun Liu, Xiaoyi Liu, Piyush Maheshwari, Anthony Bell, and Kaushik Rangadurai. 2021. VisRel: Media search at scale. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining.* 2584–2592.

[12] Cheng Chen, Chenzhe Jin, Yunan Zhang, Sasha Podolsky, Chun Wu, Szu-Po Wang, Eric Hanson, Zhou Sun, Robert Walzer, and Jianguo Wang. 2024. SingleStore-V: An Integrated Vector Database System in SingleStore. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 3772–3785. https://doi.org/10.14778/3685800.3685805

[13] James C. Corbett and et. al. 2013. Spanner: Google's Globally Distributed Database. *ACM Trans. Comput. Syst.* 31, 3, Article 8 (Aug. 2013), 22 pages. https://doi.org/10.1145/2491245

[14] Sanjoy Dasgupta and Yoav Freund. 2008. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing.* 537–546.

[15] Xin Luna Dong. 2024. The Journey to a Knowledgeable Assistant with Retrieval-Augmented Generation (RAG) *(SIGMOD/PODS '24).* Association for Computing Machinery, New York, NY, USA, 3. https://doi.org/10.1145/3626246.3655999

[16] Ming Du, Arnau Ramisa, Amit Kumar KC, Sampath Chanda, Mengjiao Wang, Neelakandan Rajesh, Shasha Li, Yingchuan Hu, Tao Zhou, Nagashri Lakshminarayana, et al. 2022. Amazon shop the look: A visual search system for fashion and home. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining.* 2822–2830.

[17] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. New trends in high-D vector similarity search: al-driven, progressive, and distributed. *Proc. VLDB Endow.* 14, 12 (July 2021), 3198–3201. https://doi.org/10.14778/3476311.3476407

[18] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. New trends in high-d vector similarity search: al-driven, progressive, and distributed. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3198–3201.

[19] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2024. Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. arXiv:2409.09913 [cs.DB] https://arxiv.org/abs/2409.09913

[20] Siddharth Gollapudi and et. al. 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. In *WWW '23* (Austin, TX, USA). Association for Computing Machinery, New York, NY, USA, 3406–3416. https://doi.org/10.1145/3543507.3583552

[21] Martin Grohe. 2020. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In *proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems.* 1–16.

[22] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *International Conference on Machine Learning.* https://arxiv.org/abs/1908.10396

[23] Jiawei Han, Xifeng Yan, and Philip S. Yu. 2006. Mining, Indexing, and Similarity Search in Graphs and Complex Structures. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE '06).* IEEE Computer Society, USA, 106. https://doi.org/10.1109/ICDE.2006.99

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 770–778.

[25] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[26] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proc. VLDB Endow.* 9, 3 (Nov. 2015), 204–215. https://doi.org/10.14778/2850583.2850594

[27] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (April 2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[28] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin'ichi Satoh. 2018. A survey of product quantization. *ITE Transactions on Media Technology and Applications* 6, 1 (2018), 2–10.

[29] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, Hossein Ahmadi, Dan Delorey, Slava Min, Mosha Pasumansky, and Jeff Shute. 2020. Dremel: a decade of interactive SQL analysis at web scale. *Proc. VLDB Endow.* 13, 12 (Aug. 2020), 3461–3472. https://doi.org/10.14778/3415478.3415568

[30] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of vector database management systems. *The VLDB Journal* 33, 5 (July 2024), 1591–1615. https://doi.org/10.1007/s00778-024-00864-x

[31] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proc. ACM Manag. Data* 2, 3, Article 120 (May 2024), 27 pages. https://doi.org/10.1145/3654923

[32] Jianbin Qin, Wei Wang, Chuan Xiao, Ying Zhang, and Yaoshu Wang. 2021. High-dimensional similarity query processing for data science. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining.* 4062–4063.

[33] Parikshit Ram and Kaushik Sinha. 2019. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining.* 1378–1388.

[34] Patrick Schäfer, Jakob Brand, Ulf Leser, Botao Peng, and Themis Palpanas. 2024. Fast and Exact Similarity Search in less than a Blink of an Eye. *arXiv preprint arXiv:2411.17483* (2024).

[35] Michael Stonebraker and Greg Kemnitz. 1991. The POSTGRES next generation database management system. *Commun. ACM* 34, 10 (Oct. 1991), 78–92. https://doi.org/10.1145/125223.125262

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

[37] Philip Sun, David Simcha, Dave Dopson, Ruiqi Guo, and Sanjiv Kumar. 2023. SOAR: Improved Indexing for Approximate Nearest Neighbor Search. In *Neural Information Processing Systems.* https://arxiv.org/abs/2404.00774

[38] Jianguo Wang and et. al. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21).* Association for Computing Machinery, New York, NY, USA, 2614–2627. https://doi.org/10.1145/3448016.3457550

[39] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2023. An efficient and robust framework for approximate nearest neighbor search with attribute constraint. In *NIPS '23* (New Orleans, LA, USA). Curran Associates Inc., Red Hook, NY, USA, Article 692, 14 pages.

[40] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3152–3165.

[41] Wei Wu, Junlin He, Yu Qiao, Guoheng Fu, Li Liu, and Jin Yu. 2022. HQANN: Efficient and robust similarity search for hybrid queries with structured and unstructured constraints. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management.* 4580–4584.

[42] Qian Xu, Juan Yang, Feng Zhang, Junda Pan, Kang Chen, Youren Shen, Amelie Chi Zhou, and Xiaoyong Du. 2025. Tribase: A Vector Data Query Engine for Reliable and Lossless Pruning Compression using Triangle Inequalities. *Proc. ACM Manag. Data* 3, 1, Article 82 (Feb. 2025), 28 pages. https://doi.org/10.1145/3709743

[43] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. Pase: Postgresql ultra-high-dimensional approximate nearest neighbor search extension. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data.* 2241–2253.

[44] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, et al. 2023. {VBASE}: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23).* 377–395.

[45] Zeqi Zhu, Zeheng Fan, Yuxiang Zeng, Yexuan Shi, Yi Xu, Mengmeng Zhou, and Jin Dong. 2024. FedSQ: A Secure System for Federated Vector Similarity Queries. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4441–4444. https://doi.org/10.14778/3685800.3685895

[46] Chaoji Zuo, Miao Qiao, Wenchao Zhou, Feifei Li, and Dong Deng. 2024. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 1, Article 69 (March 2024), 26 pages.