BigVectorBench: Heterogeneous Data Embedding and Compound Queries are Essential in Evaluating Vector Databases

Guoxin Kang
Institute of Computing Technology, Chinese Academy of Sciences
kangguoxin@ict.ac.cn

Zhongxin Ge
Institute of Computing Technology, Chinese Academy of Sciences
gezhongxin22s@ict.ac.cn

Jingpei Hu
Institute of Computing Technology, Chinese Academy of Sciences
hujingpei22s@ict.ac.cn

Xueya Zhang
University of Chinese Academy of Sciences
zhangxueya21@mails.ucas.ac.cn

Lei Wang
Institute of Computing Technology, Chinese Academy of Sciences
wanglei_2011@ict.ac.cn

Jianfeng Zhan
Institute of Computing Technology, Chinese Academy of Sciences
zhanjianfeng@ict.ac.cn

ABSTRACT

Vector databases are designed to effectively store, organize, and retrieve high-dimensional vectors, enabling faster and more accurate querying and analysis. This study highlights that the performance of cutting-edge vector databases hinges on their proficiency in managing heterogeneous data embedding and handling compound queries. The former task revolves around converting varied data types into a cohesive vector format, while the latter involves processing multimodal or single-modal queries with precise constraints. The paper advocates for evaluating these dual tasks within an integrated benchmark framework. However, state-of-the-art vector database benchmarks overlook heterogeneous data embedding and compound queries, creating a gap in evaluating vector database performance.

To address this gap, we introduce BigVectorBench, a benchmark suite designed to evaluate vector database performance. BigVectorBench contributes by defining and evaluating the embedding performance of heterogeneous data. Additionally, it abstracts compound queries, which are increasingly used in real-world applications, replacing unimodal vector searches. Our rigorous evaluations validate the two design decisions of BigVectorBench and identify performance bottlenecks of mainstream vector databases. Its source code and user manual are available from https://github.com/BenchCouncil/BigVectorBench.

PVLDB Reference Format:
Guoxin Kang, Zhongxin Ge, Jingpei Hu, Xueya Zhang, Lei Wang, and Jianfeng Zhan. BigVectorBench: Heterogeneous Data Embedding and Compound Queries are Essential in Evaluating Vector Databases. PVLDB, 18(5): 1536 - 1550, 2025.
doi:10.14778/3718057.3718078

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/BenchCouncil/BigVectorBench.

Proceedings of the VLDB Endowment, Vol. 18, No. 5 ISSN 2150-8097.
doi:10.14778/3718057.3718078

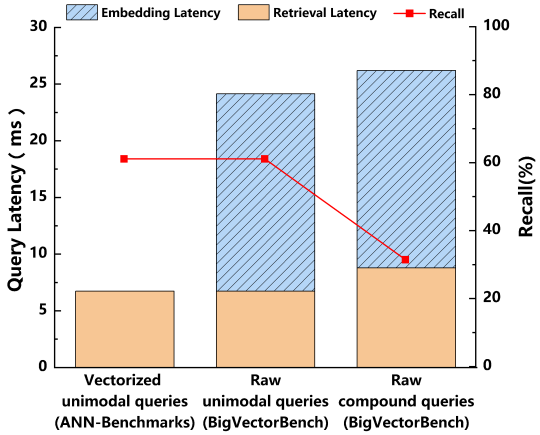


Figure 1: Benchmarking a state-of-the-art vector database: Milvus using ANN-Benchmarks and our BigVectorBench, respectively. Our experiments show data embedding latency has significant impacts on user query response time; Compared to vectorized unimodal queries, vectorized compound queries significantly decrease recall by 48.54% and reduce throughput by 23.38%. Here, 'recall' refers to the fraction of the true top-k nearest neighbors retrieved in top-k nearest neighbor searches. This substantial performance gap highlights the critical importance of heterogeneous data embedding and compound queries in benchmarking vector databases.

1 INTRODUCTION

High-dimensional vector data is the cornerstone for various techniques and applications, including Retrieval-Augmented Generation (RAG) in Large Language Models (LLMs) [32, 48, 54, 100, 104], recommendation systems [98, 99], and Bioinformatics [18, 63, 87]. Vector databases are designed to effectively store, organize, and retrieve high-dimensional vectors, enabling faster and more accurate querying and analysis.

State-of-the-art vector databases [39, 72, 73] like Milvus, Weaviate, and Qdrant necessitate direct user engagement and facilitate the handling of raw compound queries. Raw queries undergo two

essential processes: data embedding and retrieval. This necessity arises from their pivotal role in transforming a variety of user queries - whether text, images, or other data types - into vectors for efficient retrieval. These databases boast two key features that distinguish them from traditional databases. Firstly, they excel in effectively managing diverse data types. Secondly, vector databases showcase remarkable proficiency in executing rapid Approximate Nearest Neighbor (ANN) searches under various constraints. Leveraging these advantages, vector databases have become foundational infrastructure in modern artificial intelligence (AI) applications [53, 71, 90].

State-of-the-art vector database benchmarks primarily focus on the performance of Approximate Nearest Neighbor (ANN) algorithms for unimodal queries [11, 12, 55, 84] while overlooking the increasing importance of data embedding [16, 22, 53], leading to a focus solely on optimizing vector indexing in current vector database enhancements. Meanwhile, ANN-Benchmarks only model unimodal queries while overlooking compound queries. As depicted in Figure 1, these omissions result in a notable disparity when assessing the performance of vector databases across various evaluation conditions that are of interest to stakeholders.

To address these issues, this paper proposes BigVectorBench, a novel benchmark suite specifically designed to evaluate vector databases. The main contributions are summarized as follows:

(1) We have discovered that the quality and latency of embedding heterogeneous data have substantial effects on user query response times and the recall of approximate nearest neighbor searches. Therefore, we contend that assessing these two tasks within an integrated benchmark framework is crucial. A thorough examination of both heterogeneous data embedding and approximate nearest neighbor searches is necessary to precisely identify system bottlenecks.

(2) We propose a methodology for defining evaluation conditions for vector databases based on real-world application requirements. We have meticulously designed and developed an extensible benchmark framework named BigVectorBench that is tailored for vector databases. It comprises five datasets, ten workloads, four fundamental compound query types, and various variations.

(3) We conduct hundreds of experiments on the leading-edge vector databases, including Milvus [93], Weaviate [3], and Qdrant [5]. Predominantly, embedding latency, especially in multimodal queries, significantly affects response times, often exceeding hundreds of milliseconds. Current optimizations in vector databases focus mainly on improving indexing for faster retrieval in large datasets, neglecting embedding performance enhancements. This oversight is a significant gap in making vector databases more efficient. Additionally, vectorized compound queries significantly reduce retrieval throughput and recall, with recall nearly halving compared to unimodal queries.

2 BACKGROUND AND RELATED WORK

2.1 The background of vector databases

Vector databases are designed to efficiently handle and search through high-dimensional vector data, streamlining the process of managing complex inputs like images, text, or multimodal data. As illustrated in Figure 2, the retrieval process in vector databases begins with the receipt of raw user queries, which predominantly involves two key modules: heterogeneous data embedding and vectorized query retrieval.

Nowadays, embedding techniques have emerged as pivotal tools for the nuanced understanding and processing of text, images, and multimodal data. These techniques, tailored to the specifics of the original data types and their intended applications, exhibit a remarkable specialization [66]. Text embedding efficiency inversely correlates with text length, decreasing from words to documents [52, 64, 65, 75]. Unlike word embedding models, sentence and paragraph models capture context well within a 512-token limit. Meanwhile, document models manage longer texts up to 8191 tokens, capturing extensive context [7, 17, 41, 57, 70, 95–97]. Embedding techniques for image, audio, and multimodal data include models like ResNet [45], VGG [85], DeepSpeech [43], Wavenet [89], and wav2vec [14, 81]. The above models often work in isolation and cannot align heterogeneous data types efficiently. To address this, models like CLIP [76], ImageBind [35], ViLBERT [59], LXMERT [86], OSCAR [56], and UNITER [20] are designed to embed heterogeneous data into a unified embedding space.

Upon translating original data into vectors, vector databases build specific indexes to optimize data retrieval. These indexes are categorized into tree, hash, quantization, and graph-based types [55, 73]. While tree-based indexes [2, 23, 24, 67, 68, 80, 83, 101] excel in low-dimensional vector range searches, hash-based indexes [8, 9, 19, 25, 47] are adept at managing high-dimensional data by grouping similar items into hash buckets. Specifically, HD-Index [10] is constructed upon the RDB-tree (Reference Distance B+-trees), facilitating Approximate Nearest Neighbor searches within high-dimensional datasets. Quantization indexes [13, 33, 34, 37, 38, 40, 50, 58], on the other hand, economize on storage at the expense of feature loss and recall. Graph-based indexes [27, 31, 44, 46, 49, 61, 62, 92] stand out for their efficiency in pinpointing nearest neighbors in large datasets and their ability to add new data points, albeit at the cost of increased index construction time and computational resources.

2.2 State-of-the-art and state-of-the-practice benchmarks

While heterogeneous data embedding and compound queries are widely used in the industry, it has come to our attention that these critical features have yet to be integrated into any existing benchmarks. Currently, there is a notable absence of a specialized benchmark tailored for vector databases. In this context, Big-ANN-Benchmarks [84] and ANN-Benchmarks [11, 12] stand out as the most relevant benchmarks available and are designed to evaluate the performance of Approximate Nearest Neighbor (ANN) algorithms. Lin et al. [55] contribute extensively by providing numerous datasets and conducting a comprehensive experimental evaluation of 16 ANN algorithms under various settings.

The primary goal of ANN-Benchmarks is to evaluate how well approximate nearest neighbor algorithms work since this is the most crucial function in vector databases. It contains synthetic and realistic datasets and is compatible with the majority of ANN libraries, including FAISS [30, 51], Annoy [2], NMSLIB [61], and HNSWlib [62]. A distinctive feature of ANN-Benchmarks is its

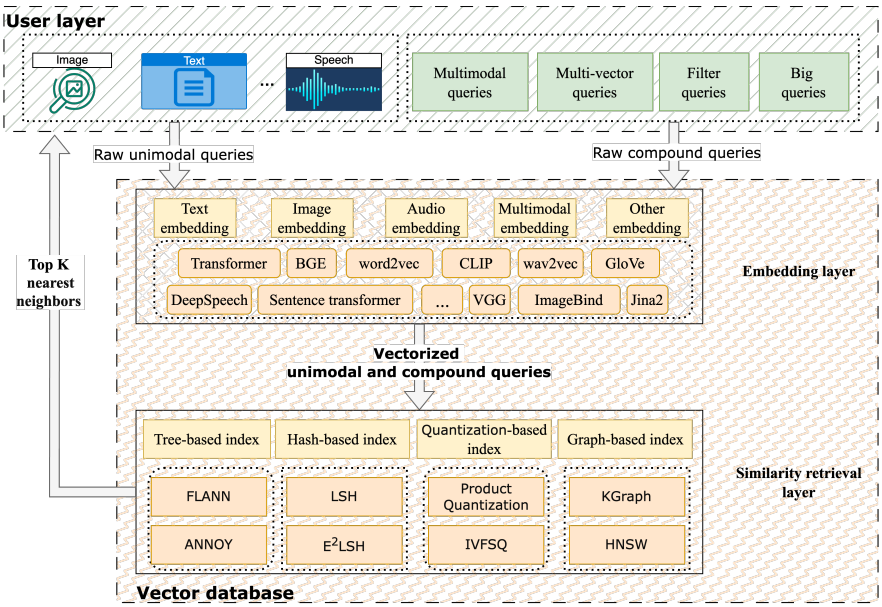


Figure 2: The vector database retrieval workflow.

Table 1: Comparing BigVectorBench against leading-edge benchmarks.

Benchmarks	Embedding				Unimodal queries	Compound queries				Large-scale datasets
	Text	Vision	Audio	Multimodality		Multi-vector queries	Multi-modal queries	Filter queries	Big queries	
ANN-Benchmarks	×	×	×	×	√	×	×	×	×	×
Big-ANN-Benchmarks	×	×	×	×	√	×	×	√	×	√
BigVectorBench	√	√	√	√	√	√	√	√	√	√

flexibility, allowing users to tailor vector indexes with customized parameters. Ultimately, ANN-Benchmarks deliver comprehensive performance metrics, including search time and recall metrics. Depending on the evaluation results, it automatically generates either scatter plots or interactive visualizations, offering intuitive insights into algorithm performance. Big-ANN-Benchmarks are a more advanced version of ANN-Benchmarks, specifically engineered to tackle the complexities of large-scale datasets emblematic of real-world scenarios, where data volumes can soar into the billions.

We compare BigVectorBench with ANN-Benchmarks and Big-ANN-Benchmarks in Table 1. Existing benchmarks focus solely on evaluating the performance of ANN algorithms through unimodal queries, neglecting the crucial aspects of embedding performance and the compound queries existing in modern vector databases. Additionally, existing benchmarks fail to demonstrate how the quality of heterogeneous data embedding influences approximate nearest neighbor search performance.

Embeddings are essential for translating data and capturing its essential structure, and the quality of heterogeneous data embeddings substantially affects the recall of compound queries. Our experimental results show the embedding latency for an 8192-length text spirals to a staggering 800 milliseconds. This duration even exceeds the latency incurred during a top-k approximate nearest neighbors search. Intuitively, we can truncate a long text to a shorter length to achieve reduced embedding latency. For instance, processing a 128-length text merely requires 52 milliseconds. Unfortunately, we found that the trade-off for reducing embedding latency comes at the cost of sacrificing context richness, leading to poor embedding efficiency. As a result, this translates into a less than satisfactory outcome of identifying fewer than 12 accurate nearest neighbors in a top 100 nearest neighbors search.

Moreover, we quantify the performance gap between unimodal queries and compound queries, such as vector query within restrictive conditions, which significantly impact the retrieval performance of vector databases. Figure 1 illustrates that on a CPU platform, using the IVF_PQ index, there is a significant decrease in both throughput and recall for Milvus, one of the state-of-the-art vector databases, by 23.38% and 48.54%, respectively. The detailed differences between unimodal and compound queries are presented in Section 4.3.

3 MOTIVATION

3.1 Evaluating heterogeneous data embedding latency and efficiency is essential for vector database evaluation

The comprehensive benchmarking of vector databases necessitates the inclusion of embedding of original heterogeneous data, a critical

execution path in the retrieval process in vector databases. Raw user queries, whether text, images, audio, or multimodal, are initially processed through the data embedding module. This crucial step translates high-dimensional data into a more manageable, lower-dimensional, dense, and continuous vector space, setting the stage for subsequent operations. However, prevailing benchmarks tend to focus narrowly on the performance of the top-k approximate nearest neighbors search within vector databases, inadvertently overlooking the significance of heterogeneous data embedding in the total execution path of the retrieval process. This oversight is nontrivial as the efficiency and latency of heterogeneous data embedding are instrumental in determining the end-to-end performance of retrieving vector databases.

Each user query response fundamentally depends upon two key execution paths: the embedding latency for raw user queries and the retrieval latency from the vector database. Embedding latency thus plays a critical role in influencing user response times. Additionally, the efficiency of the embedding process is essential for conducting effective top-k approximate nearest neighbor searches. Striking a balance between embedding latency and efficiency is crucial for optimizing user experience. In the realm of embedding models, the richness of the original data correlates directly with the quality of the resulting embedding vectors. However, this richness comes at a cost, introducing challenges that cannot be overlooked. For example, longer texts, brimming with contextual information, demand significantly more computational power, memory, and time for processing, thus heightening the embedding latency.

Embeddings are pivotal for translating data and capturing its essential structure. Our experimental results underscore the need for a trade-off between embedding latency and efficiency to optimize vector database response time. This balance ensures rapid and precise vector retrieval, highlighting the necessity of considering both factors in vector database evaluations.

3.2 Exploring the core competency of vector databases: the critical importance of handling compound queries

Traditional vector databases are designed for simple unimodal queries, which search for the top-k approximate nearest neighbors from the large-scale data set. These databases aim to establish more efficient indexes for quickly retrieval the approximate nearest neighbors, such as tree-based, graph-based, hash-based, quantization-based, and hybrid indexes. However, the workload patterns processed in vector databases are changing from simple unimodal queries to compound queries because of the complex application scenarios. The natural productive environment leads to similarity queries that accompany many compound restrictive conditions, such as date, space, and topics.

ANN-Benchmarks stand as the leading benchmark for evaluating the approximate nearest neighbor (ANN) capabilities of existing vector databases. It posits that superior vector databases excel in processing a higher number of queries per second and achieving high recall on large-scale data sets. This benchmark has guided developers to primarily focus on enhancing the ANN capabilities of vector databases. However, it inadvertently overlooks the critical need for handling compound queries, which are prevalent in real-world production environments.

For instance, in content-based recommendation scenarios, users often seek the top-k news items related to the specific news within restrictive conditions, such as a given time frame, rather than sifting through the entire data set. This requirement emphasizes the necessity for vector databases to efficiently execute vector queries within restrictive conditions on partial data sets, as opposed to merely performing simple unimodal queries on extensive data collections. In the realm of vector databases, the shift towards executing within restrictive conditions on partial datasets, as opposed to conducting simple similarity queries on extensive data collections, necessitates a comprehensive reassessment of system architecture and optimization strategies. This paradigm shift has profound implications for indexing strategies, where a transition to more dynamic and fine-grained indexing is imperative to facilitate efficient data retrieval from subsets. Instead, BigVectorBench underlines the importance of developing and optimizing vector databases to support more complex, real-world workload patterns.

4 BENCHMARK DESIGN AND IMPLEMENTATION

In this section, we present BigVectorBench, a benchmark suite designed to evaluate vector databases effectively.

4.1 Methodology

Our aim with BigVectorBench is to conduct a meticulous evaluation of leading-edge vector databases [3–5]. Inspired by the methodology in [102], we have conducted a thorough survey of use cases in real-world applications of vector databases, including cross-modal retrieval, image-text matching, and video recommendation. This comprehensive analysis has provided a solid foundation for establishing evaluation conditions.

Derived from the requirements of real-world applications, we distill two essential tasks for evaluating vector databases: heterogeneous data embedding and compound query execution. Then, we establish evaluation conditions by varying different and representative task instances, algorithms, algorithm instances, processors, and systems. We use an example to illustrate how we developed the evaluation conditions for evaluating vector databases.

For example, in vector database evaluations, both users and vector database developers are concerned with the retrieval performance of compound queries, where filter queries serve as a key task instance. Filter queries are commonly found in e-commerce applications, such as in the recommendation system of Amazon book [1]. Our evaluation objectives are multiple ANN algorithms to handle filter queries in vector databases, covering various vector indexing algorithms, including tree-based, hash-based, quantization-based, and graph-based indexes, with specific implementations such as the ivfpq index in Milvus.

Given the extensive complexity and size of the datasets and workloads, a full-scale evaluation of vector databases could be prohibitively expensive and time-consuming. To ensure a focused and representative evaluation, we carefully select relevant datasets

[1] https://www.amazon.com/books-used-books-textbooks

and workloads that best capture the essence of these tasks. Use cases and benchmark design will be elaborated in Sections 4.2 and 4.3.

4.2 Real-world use case studies

We have summarized corresponding use cases for data embedding and compound queries, including E-commerce recommendations, keyframe recommendations, cross-modal text-to-image searches, and long text searches. Due to space limits, this subsection will mainly introduce the cross-modal text-to-image search use cases. Additional details are available at our GitHub repository: https://github.com/BenchCouncil/BigVectorBench.

Cross-modal text-to-image search represents an emerging use case of vector retrieval, where both text and image are converted into vectors using multi-modal embedding models like CLIP. This use case is increasingly employed in popular social networking platforms such as TikTok [2] and REDnote [3]. Users can input textual descriptions — for instance, detailing a historical event or artwork — and the application then retrieves and displays images that are visually similar to the described content. This approach not only makes the information more accessible but also enhances its visual appeal, facilitating a more intuitive understanding of the search results.

4.3 Benchmark design

4.3.1 Basic types of compound query. We abstract four distinct instances of compound queries: multi-vector query, multi-modal query, vector query with filters, and big query, delineated as follows:

- **Multi-vector query:** It consists of multiple vectors of the same modality, each describing aspects of a single object or feature. For instance, in video recommendation systems, we employ a segmentation and equal-spacing sampling strategy to extract four keyframes from each short video [15], following prominent works such as FiT [15], TSN [94], and GST [60]. These keyframes are then used to construct a multi-vector query to retrieve videos with similar content. Additionally, another variant of the multi-vector query integrates both dense and sparse vectors. Currently, BigVectorBench only supports multi-vector workloads using vectors of equal dimensionality that utilize Euclidean distance.
- **Multi-modal query:** It integrates heterogeneous modal data, such as textual and visual data, into a single query. A common application is an image-text retrieval task, especially in the context of large language model retrieval systems, where the query involves processing and understanding both textual and visual information.
- **Vector query with filters:** It employs specific filters as criteria or labels to search for similar vectors. This type of query is particularly common in business-critical applications where users need to find products or services that meet certain constraints, like price ranges, sales volumes, or delivery timelines. A common filtering criterion is the timestamp. For instance, in the cc_news dataset [42], each record consists of a timestamp paired with corresponding news text. By converting timestamps into Unix time, it becomes feasible to conduct efficient similarity searches for news articles within specified time intervals.
- **Big query:** It encompasses significantly more information than an unimodal query, potentially including extensive text documents or lengthy audio recordings. These queries are typically represented as high-dimensional vectors, often exceeding 1,000 dimensions, with examples featuring dimensions of 1,536 and 2,072. Abstracted from real production environments, big queries incorporate extensive and complex information, making them exceptionally suitable for data-intensive applications.

We have abstracted and defined four fundamental types of compound queries as above. In practice, a compound query manifests as a combination of one or more basic types, reflecting the complex and diverse needs of users in a production environment.

4.3.2 Workloads design principles. The workloads in BigVectorBench should represent and amplify these four fundamental aspects as follows:

(1) Data points in the same workload should be in the same Euclidean space and directly available for use in calculations, including but not limited to addition, subtraction, scalar multiplication, and dot product. The selection of distance metrics aligns with the implementation of vector databases. Euclidean distance is the only metric universally supported by Milvus, Weaviate, and Qdrant, emphasizing its importance for comparable performance evaluation across various vector databases. Similarity calculations are fundamental to many nearest-neighbor search algorithms.
(2) Although data points within the same workload occupy the same dimensional space, the dimensionality can vary significantly across different workloads. This diversity in dimensions leads to a cluttered and unorganized appearance of the data on a larger scale. The challenge in managing this clutter lies in designing vector databases that are flexible enough to handle data points across varying dimensions rather than being optimized for a specific dimensionality or data structure.
(3) BigVectorBench should include large-scale workloads to mimic the challenges of processing and analyzing vector data in real-world scenarios. The workload must be representative of the colossal nature of vector data in production environments, encompassing tens of millions to even hundreds of millions of data points. Large-scale workloads help in testing the scalability and performance of vector databases, ensuring that they can handle the voluminous and complex nature of real-life vector data efficiently.
(4) The workloads in BigVectorBench should be continuous, reflecting the ongoing nature of data generation in real-world applications. Continuous vector data are produced from various devices and applications, such as sensors in IoT devices, user interactions in web applications, and real-time monitoring systems. Consequently, BigVectorBench should include workloads that simulate or originate from continuous data sources, facilitating the development and

[2] https://www.tiktok.com/about
[3] https://rednote.in/

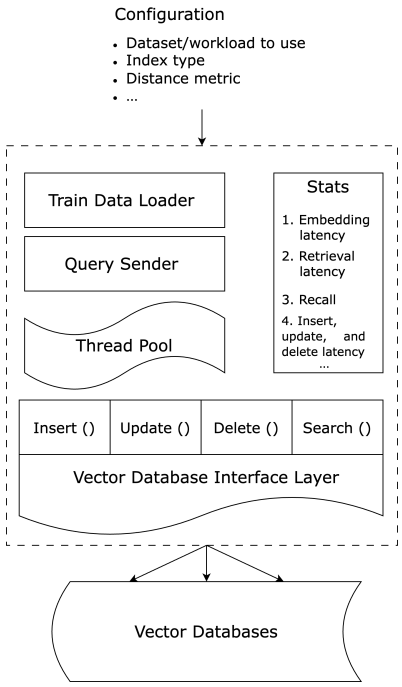


Figure 3: BigVectorBench architecture.

evaluation of vector databases that can embed and retrieve data in a real-time or near-real-time context.

In essence, to thoroughly evaluate vector databases, the workloads employed must obey the above characteristics. This comprehensive design ensures that the workloads not only simulate real-world complexities but also serve to evaluate the ability of existing vector databases to handle complex vector data in a production environment.

4.3.3 Dataset and workload details. We have meticulously chosen five datasets and ten workloads, all commonly used in real-world applications, to thoroughly evaluate the embedding and retrieval performance of vector databases.

Dataset sources. Aligned with our motivation, we meticulously selected 5 datasets and 10 workloads from typical real-world scenarios to achieve two main goals. The first goal is to evaluate the latency and efficiency of embedding models, while the second goal focuses on evaluating the search performance in vector databases.

We conducted a comprehensive evaluation of various embedding models, including text, image, audio, video, and multi-modal datasets. The text datasets were categorized based on the number of tokens in a single entry, leading to the formation of BigVectorBench, which encompasses sentence-level, paragraph-level, and short document-level datasets. Specifically, the maximum token count in the paragraph-level dataset is eightfold that in the sentence-level, and similarly, the short document-level dataset contains up to eight times the tokens of the paragraph-level dataset.

To gauge the latency and efficiency of embedding models, we introduced text and image classification datasets. In practical settings, vector databases often process complex inputs like PDF screenshots and user voice recordings, necessitating Optical Character Recognition (OCR) and Automatic Speech Recognition (ASR) for initial data conversion into text. The precision of this conversion is crucial for embedding efficiency, while the conversion speed significantly impacts user response times. Our datasets also aim to evaluate the performance and latency of OCR and ASR tools.

Additionally, we provide various workloads to evaluate the search performance of vector databases for fundamental compound queries, categorized into multi-vector, multi-modal, filter, and big query workloads. Multi-vector workloads, derived from tasks like multi-image retrieval, are vital in contexts like video recommendation, where keyframes are extracted to find similar content. Multi-modal workloads, such as those combining image and text for retrieval tasks, are prevalent in industry production, facilitating the search for similar images, texts, or combinations thereof. Filter workloads are often used to filter content by specific criteria. For instance, users can filter news by specific time ranges or select books with high ratings for purchase. Finally, the characteristics of big datasets are notable in two respects: first, the colossal volume of datasets, and second, the rich information contained within each data point, as seen in tasks like long-text retrieval.

Workloads formats. Workloads are stored in HDF5 format. Each workload is comprised of two components: a training set and a testing set. The original dataset comes pre-partitioned into training and testing sets, which we will directly adopt. For workloads lacking a predefined testing set, we will employ a pseudorandom method to extract 10,000 data points as the testing set. This pseudorandom approach ensures reproducibility, guaranteeing that the testing set comprises the same data points in each instance.

Consistent with the basic type of compound queries, the data point formats in the training set are categorized into four basic types. The first type comprises an identifier (id) accompanied by multiple identical modal vectors, represented as Equation 1. The second type includes an id alongside multimodal vectors, as shown in Equation 2. The third type involves an id, a single vector, and a set of filters, detailed in Equation 3. The final type consists of an id paired with a single vector, formalized in Equation 4. In specific application contexts, data point formats may naturally exist and can be represented by any combination of the four basic types previously outlined.

The workloads consist of training sets, testing sets, and ground truth, where the ground truth consists of the IDs of the top-k nearest neighbors and the distances between each testing point and its respective neighbors. We will provide a detailed explanation of the ground truth computation process in Section 4.3.3.

$$[\text{id}, \text{vector}_1, \text{vector}_2, \ldots, \text{vector}_n] \tag{1}$$

$$[\text{id}, \text{vector}_{1\text{m}}, \text{vector}_{2\text{m}}, \ldots, \text{vector}_{n\text{m}}] \tag{2}$$

$$[\text{id}, \text{vector}, \text{label}_1, \ldots, \text{label}_n] \tag{3}$$

$$[\text{id}, \text{vector}] \tag{4}$$

Ground truth computation. In our approach, for each testing point in the testing dataset, we employ the k Nearest Neighbors (kNN) algorithm to identify the top-k nearest neighbors from the corresponding training set. We formalize the identities of these

neighbors as $[\mathrm{id}_1, \mathrm{id}_2, \ldots, \mathrm{id}_k]$ and also record the Euclidean distances between the testing point and each of the top-k nearest neighbors, presented as $[\mathrm{dis}_1, \mathrm{dis}_2, \ldots, \mathrm{dis}_k]$.

The method for computing ground truth varies depending on the type of compound query. For multi-vector queries, where all vectors are of the same modality, we apply fusion techniques to amalgamate them into a single comprehensive vector. This facilitates the comparison of distances between each testing point and all training points. The top-k nearest neighbors are determined as the K training points that exhibit the smallest distances to this testing point. For multi-modal queries, common existing in image-text and audio-text retrieval tasks, we utilize textual information to establish a baseline ground truth. In the case of labeled queries, we initially refine the candidate set based on the filters before selecting the top-k nearest neighbors according to the distance metrics from the testing point to the candidates. For big queries and unimodal queries, the selection of the top-k nearest neighbors is straightforwardly based on the distances between the testing points and the entire training set, ensuring simplicity and consistency in BigVectorBench.

4.4 BigVectorBench implementation

BigVectorBench is crafted to assess the latency and efficiency of the embedding process, as well as to gauge the performance of insertion, updating, deletion, and search operations within vector databases. Implemented in Python, the architecture of BigVectorBench, as depicted in Figure 3, begins by parsing configuration settings. It loads the training data into a chosen vector database and constructs the specified index on this data while the time taken to build the index is recorded. Subsequently, the query sender dispatches the original query to a designated embedding module, and the embedding time is logged.

To accelerate data loading and query processing, BigVectorBench utilizes a thread pool technique. Once the original compound queries are converted into testing vectors, they are channeled through the database interface layer into the vector database for various operations, with performance metrics being meticulously captured and reported by the statistics module.

Furthermore, BigVectorBench pioneers the establishment of a standardized interface protocol for vector database interactions, positioning it as a trailblazing benchmark in the field. It encompasses four basic operations, as detailed below, allowing any new vector database to be evaluated through the implementation of these operations:

- **Insert()**: Adds individual or multiple vector points to a designated vector collection, respecting the specific index type.
- **Update()**: Modifies an existing vector point within the vector collection.
- **Delete()**: Removes a vector point from the vector collection.
- **Search()**: Retrieves the top-k approximate nearest neighbors for a testing point or locates other specific vector points within the vector collection.

5 EVALUATION

5.1 Experiment setup

The server includes 2 Intel Xeon 5218R@2.10GHz CPUs, 512 GB memory, and an NVIDIA V100-PCIE-16GB GPU connected via PCIe 3.0. Each CPU has 20 physical cores, and hyper-thread is enabled. We used all of the 80 hardware threads. The operating system is Ubuntu 20.04 with the Linux kernel 5.15.0. The GPU driver version is 535, and CUDA 12.2 is used for GPU computing. We use Python 3.10 and Docker 26.1 for all experiments. We choose Milvus, Weaviate, and Qdrant because they support diverse computational hardware resources, employ various indexing types, and utilize unique strategies for processing compound queries, exemplifying the diverse landscape of vector databases.

5.2 Experiment methodology

As illustrated in Table 2, our experimental methodology is structured into three related parts:

First, we verify the critical design decision of BigVectorBench on two subjects, Jina Embedding v2 [41] and Milvus [4], detailed in subsection 5.3. Our focus narrows to two pivotal aspects: (1) **why heterogeneous data embedding performance is essential for vector database evaluation**; and (2) **how compound queries impact the vector database performance**.

5.2.1 Image embedding. Second, we utilize BigVectorBench to investigate the impacts of key influencing factors of the embedding layer and expose the trade-offs between embedding latency and efficiency, as outlined in subsection 5.4. While not exhaustively testing all existing embedding models, we aim to highlight critical latency and efficiency trade-offs crucial for stakeholders' considerations. Specifically, we evaluate:

- Image Embedding: Using clip-ViT [76] as the subject, utilizing image patches in base 16, base 32, and large 14 formats to measure embedding latency, GPU memory consumption, and accuracy. Here, "accuracy" is defined as the proportion of predictions that the clip-ViT model correctly identifies in comparison to the actual labels in the ImageNet [26] dataset.
- Text Embedding: Employing Jina Embeddings v2 [41] as the subject with truncation lengths of 128, 1024, and 8192 to assess embedding latency and GPU memory usage.
- Multimodal Embedding: Implementing ImageBind [35] as the subject to handle data across six different modalities. In real-world business applications, two predominant modal alignment strategies are observed. The first leverages a joint embedding model, such as the open-source ImageBind, to integrate multimodal data into a unified embedding space. The second strategy converts multimodal data into a singular modality, employing techniques like OCR (Optical Character Recognition) for text-rich images [1] or ASR (Automatic Speech Recognition) for audio [91], subsequently processed using ImageBind.

Third, employing BigVectorBench, which is built on the datasets [6, 15, 36, 42, 69, 74, 82, 88, 103] to thoroughly evaluate the subjects — leading-edge vector databases like Milvus [4, 93] version 2.4.1, Weaviate [3] version 1.24.12, and Qdrant [5] version 1.9.2. These

Table 2: Experimental Methodology.

Experimental Methodology		Subject	Subject's Key Factors	Dataset / Workload	Performance Metric
Key Feacture Verification	Embedding	Jina Embeddings v2	Truncation length	D1: arXiv and PubMed (768)	Latency and efficiency
	Retrival	Milvus 2.4.1	Index Types: HNSW, IVF_PQ, GPU_CAGRA, GPU_IVF_PQ	W3: app_reviews (384)	QPS and recall
Embedding Latency and Efficiency	Image	CLIP-ViT-B-16	Pixel dimensions of the image patches	D2: ImageNet (512)	Embedding latency GPU memory usage and accuracy
		CLIP-ViT-B-32			
		CLIP-ViT-L-14			
	Text	Jina Embeddings v2	Truncation length	D1: arXiv and PubMed (768) D3: squad_v2 (768) [78, 79]	
	Multi-modal	ImageBind	Modal alignment strategy	D4: img-wikipedia (1024) D5: librispeech_asr (1024)	
		ASR + ImageBind		D4: img-wikipedia (1024)	
		OCR + ImageBind		D5: librispeech_asr (1024)	
Retrieval speed and accuracy	Filter queries	Milvus 2.4.1 Weaviate 1.24.12 Qdrant 1.9.2	Index Types Milvus: HNSW, IVF_PQ, GPU_CAGRA, GPU_IVF_PQ, GPU_IVF_FLAT Weaviate: HNSW Qdrant: HNSW	W1: ag_news (384) W2: cc_news (384) W3: app_reviews (384) W4: amazon_books (384)	QPS and recall
	Multi-modal queries			W5: img-wikipedia (1024) W6: librispeech_asr (1024) W7: gpt4vision (1024)	
	Multi-vector queries			W8: webvid (512)	
	Big queries			W9: dbpedia-entities (1536) W10: dbpedia-entities (3072)	

Note: D_i represents the dataset, W_i denotes the workload, and the parentheses following the dataset/workload indicate the dimensions of the vector embeddings.

databases collectively handle workload volumes ranging from hundreds of thousands to tens of millions of records. Our evaluations meticulously assess performance metrics such as Queries Per Second (QPS) and top 100 recall across various query types, including filter, multi-vector, multi-modal, and large queries. To ensure a comprehensive assessment and to explore the balance between retrieval speed and accuracy, we configure nearly the entire parameter space for retrieval indexes in each database, extending our performance evaluation to include insert, update, and delete operations within these mainstream vector databases.

5.3 Key features verification

5.3.1 Embedding latency and efficiency. We demonstrate that the performance of heterogeneous data embedding is critical for evaluating vector databases. To our knowledge, no existing benchmarks specifically address the performance of heterogeneous data embedding in vector databases. However, our experiments reveal that the latency of heterogeneous data embedding on GPUs is 91.474 times greater than its retrieval latency. For our study, we utilized the Jina Embeddings v2 model, the Milvus database, and a summarization dataset from arXiv and PubMed [21].

Our findings show that the embedding latency for a truncation length of 8192 is 15.548 times higher than for a truncation length of 128 on GPUs, and 268.615 times higher on CPUs. While shorter truncation lengths achieve lower latency, they result in significant information loss, thereby reducing embedding efficiency. Our results indicate that the Approximate Nearest Neighbor search accuracy with a truncation length of 128 is only 11.407% of that achieved with a truncation length of 8192. Additionally, the GPU DRAM consumption for the truncation length of 8192 is 3.582 times that of the truncation length of 128.

5.3.2 Retrieval speed and accuracy. Another important feature of our benchmark is the ability to comprehend the behavior of a vector database with compound queries. BigVectorBench supports four basic types of compound queries: multi-vector, multi-modal, vector with labels, and big queries. In this experiment, we focus on comparing unimodal queries and vector queries with labels in Milvus across both CPU and GPU platforms.

In real-world applications, vector queries are commonly paired with one or more labels. For instance, users of a news application may prefer to retrieve Approximate Nearest Neighbors based on timestamps, while e-commerce app users might prioritize retrieval based on price ranges and commodity ratings. Despite this, the state-of-the-art benchmark — ANN-Benchmarks, typically only provides unimodal queries for evaluation.

Figure 1 illustrates the performance impact when labels are injected into unimodal queries. On the GPU platform, the throughput and recall for a standard unimodal query are 148.510 queries per second and 61.093%, respectively. However, with labels added, these figures decrease by 23.388% and 48.542%. Similarly, on the CPU platform, unimodal query performance stands at 172.839 queries per second and 61.023% recall. When labels are incorporated, throughput and recall drop by 5.462% and 31.224%, respectively.

5.4 Embedding latency and efficiency

Building on subsection 5.3.1, which highlights the importance of embedding performance in evaluating vector databases, this subsection provides a deeper analysis of heterogeneous data latency

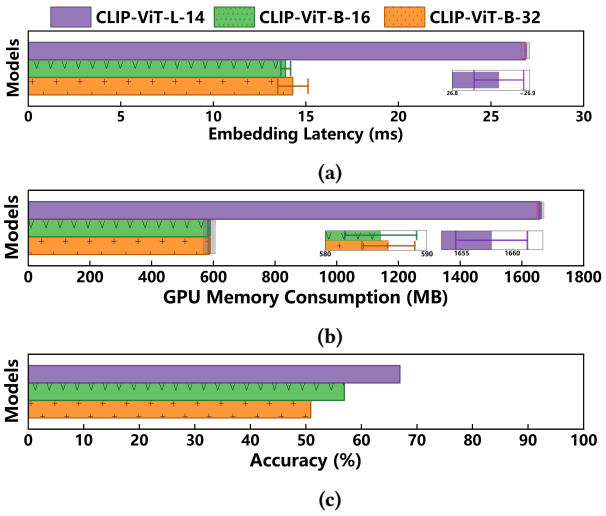


Figure 4: Image embedding latency, GPU memory consumption, and accuracy on D2: ImageNet.

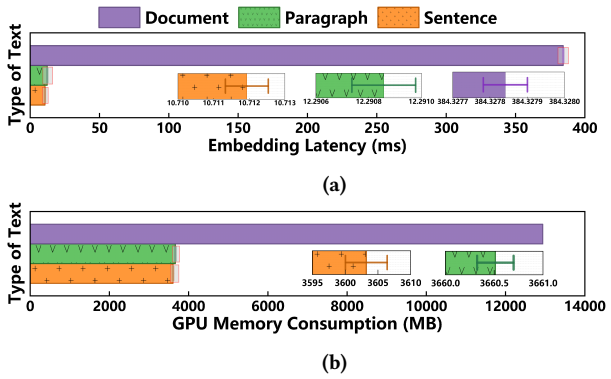


Figure 5: Text embedding latency and GPU memory consumption on D1: arXiv and PubMed and D3: squad_v2.

and efficiency. This analysis is important as it directly impacts user experience and the search performance of vector databases.

We conducted experiments using BigVectorBench to examine the trade-off between image embedding latency and efficiency, allowing users to determine the most suitable embedding model for their specific application scenarios. BigVectorBench incorporates the CLIP-ViT-B-16, CLIP-ViT-B-32, and CLIP-ViT-L-14 models [28, 29, 77], which vary in model size and the resolution of processed image patches. Here, "B" and "L" denote base and large model sizes, respectively, while "16", "32", and "14" indicate the pixel dimensions of the image patches. The benchmark records average embedding latency, GPU memory usage, and the accuracy of these models.

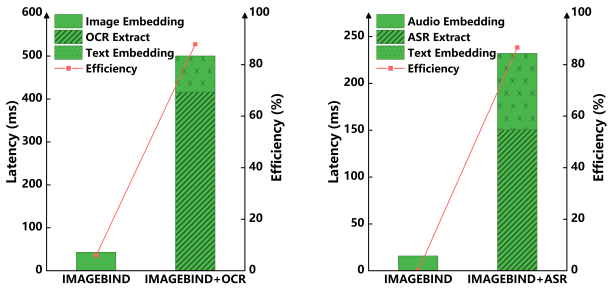
Figure 4 presents the results on embedding latency, GPU memory consumption, and accuracy for each model. The CLIP-ViT-L-14, being the largest model with the most parameters, exhibits the highest embedding latency at 26.863 milliseconds and the greatest GPU memory consumption at 1657.924 MB. The CLIP-ViT-B-16 and CLIP-ViT-B-32 models show similar performances in terms of latency and GPU usage. Specifically, the latency of CLIP-ViT-L-14 is approximately twice that of the CLIP-ViT-B-16 and CLIP-ViT-B-32 models, and its GPU memory usage is over 2.8 times higher. In terms of accuracy, the CLIP-ViT-B-16 achieves 85% of the CLIP-ViT-L-14's accuracy, while the CLIP-ViT-B-32 reaches 76% of that figure. This trade-off highlights the need for careful image embedding model selection based on specific stakeholders' concerns, balancing embedding accuracy with latency demands.

5.4.1 Text embedding. Next, we utilize BigVectorBench to evaluate the performance of text embeddings, which is crucial for improving the semantic search capabilities of vector databases. As detailed in Section 4.3.3, our experiments focus on measuring the latency for embedding texts of varying lengths — specifically sentences, paragraphs, and documents — using the Jina Embeddings v2 model [41]. This approach contrasts with our image embedding experiments, where we compared models with varying numbers of parameters; here, we are exclusively concerned with the embedding latency associated with different text lengths.

Our findings reveal that embedding latency significantly increases with the length of the text. As shown in Figure 5a, embedding a document containing 8192 tokens can take as long as 400 milliseconds on a GPU. Furthermore, the retrieval latency for document-level vectors in Milvus is noted to be around ten milliseconds. These results highlight a significant bottleneck within vector databases — the excessive latency of document embedding, which differs from past focuses in vector database optimizations that predominantly aimed at reducing retrieval latency and improving recall for vector searches. Moreover, our experiments in Figure 5b underscore that document embedding is highly demanding in terms of computational resources. Specifically, the GPU memory consumption for document embedding is 3.590 times greater than that required for sentence embedding.

5.4.2 Multi-modal data embedding. In this subsection, we evaluate the latency and efficiency of various strategies for embedding multi-modal data using BigVectorBench. The first approach utilizes a joint embedding model, exemplified by the open-source ImageBind, to integrate multimodal data into a single embedding space. Although this model eliminates the need for aligning multimodal data, its current stage of development shows limited alignment capabilities, resulting in suboptimal embedding efficiency. Our experimental results (Figure 6) indicate that the image embedding latency for ImageBind is only 42.943 milliseconds, while the audio embedding latency is 15.811 milliseconds. However, the inefficiency in handling multimodal data leads to a recall of no more than 0.06 for the top 100 nearest-neighbor searches.

The second strategy involves converting multimodal data into a single modality, such as transforming text-rich images to text via OCR (Optical Character Recognition) or converting audio to text using ASR (Automatic Speech Recognition). Once the data is translated into text, it is processed using a text embedding model. This approach significantly enhances the recall of the top 100 nearest neighbor searches to over 0.8. However, this improvement in recall is achieved at the expense of increased latency. OCR and ASR exhibit extraction delays nearing 100 milliseconds, and the latency for text embedding extraction is even higher.



(a) D4: img-wikipedia [6] (b) D5: librispeech_asr [74]

Figure 6: Comparison of multi-modal embedding strategy performance.

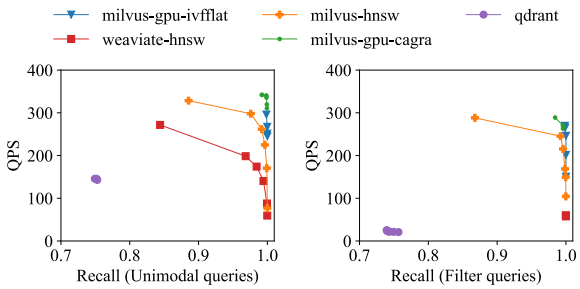


Figure 7: QPS/Recall tradeoff on W1: ag_news
Left: unimodal query, right: filter query with one label.
Up and to the right is better. Subsequent figures follow the same pattern and will not be reiterated.

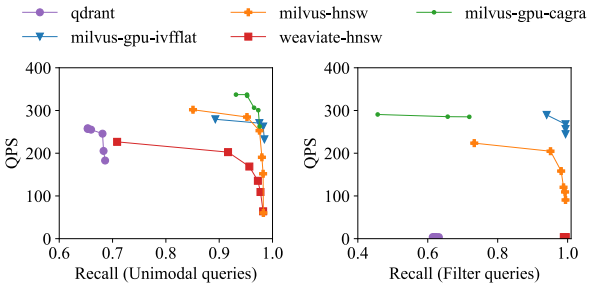


Figure 8: QPS/Recall tradeoff on W2: cc_news
Left: unimodal query, right: filter query with one label.

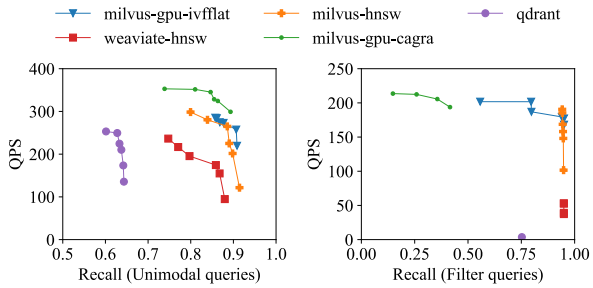


Figure 9: QPS/Recall tradeoff on W3: app_reviews
Left: unimodal query, right: filter query with 3 labels.

5.5 Retrieval speed and accuracy

In this subsection, we explore the trade-off between retrieval speed and accuracy for compound queries, which are categorized into four types: filter, multi-modal, multi-vector, and big queries. This analysis provides insights that help users determine the optimal trade-off between speed and accuracy depending on the specific index and vector database configurations. To accommodate the diverse scales of workloads, we have established several distinct experimental setups using BigVectorBench. This structured approach allows us to systematically evaluate performance across a range of data volumes and query complexities.

5.5.1 Filter queries performance. Figure 7-9 illustrate the performance of various databases on the ag_news, cc_news, as well as app_reviews workloads. The hardware configuration employed in these tests encompassed 16 hardware threads, 64GB of memory, and one NVIDIA V100-PCIE-16GB GPU. Indexes (Milvus with GPU_CAGRA index and Milvus with GPU_IVF_FLAT) running on the GPU generally exhibit higher QPS than the HNSW index on the CPU, and QPS without filters tend to be higher than QPS with filters applied. Notably, Qdrant shows the steepest decline in QPS in the presence of filters, plummeting from over 100 to less than 25.

Figure 10 presents the outcomes of the amazon_books workload on Milvus under the resources of 80 hardware threads, 512GB memory, and one NVIDIA V100-PCIE-16GB GPU. Due to the limited resources, we only test with GPU_IVF_PQ and HNSW index. The app_reviews and amazon_books workloads show an increase in maximum recall when filters are applied, as opposed to when no filters are used. Specifically, on the app_reviews workload, the use of filters boosts the maximum recall from 0.91 to 0.95, with the HNSW index demonstrating a particularly pronounced effect, as the recall approaches the maximum value under all parameter settings. Furthermore, on the amazon_books workload, the recall of the GPU_IVF_PQ index is also enhanced, rising from around 0.3 to a maximum of 0.56 with filter conditions. This might be due to the filtering process, which potentially reduces the density of the data, making it more distinct and thereby increasing recall.

5.5.2 Multi-modal queries performance. Figure 11-13 presents the search performance of various databases on multi-modal workloads. The upper limit of Recall for direct embeddings on these workloads (Figure 11 left, Figure 12 left and Figure 13) is not very high, which is influenced by the insufficient alignment capability of the embedding models when dealing with multi-modal data (see Section 5.4.2 and Figure 6). Figure 12 right displays the results for the audio workload librispeech_asr after text extraction via ASR. There is a trade-off between QPS and Recall, with the following order of performance from highest to lowest: Milvus with GPU_CAGRA index > Milvus with GPU_IVF_FLAT index > Milvus with HNSW index > Weaviate

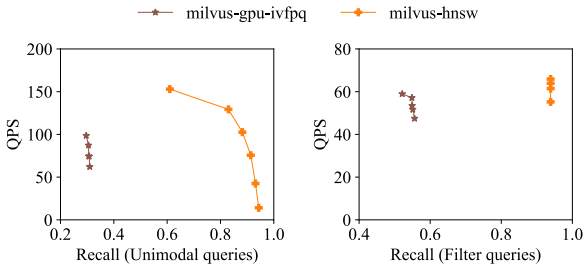


Figure 10: QPS/Recall tradeoff on W4: amazon_books
Left: unimodal query, right: filter query with five labels.

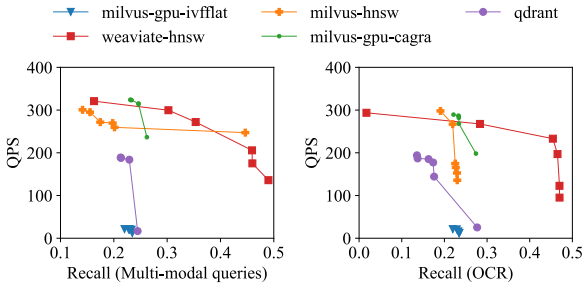


Figure 11: QPS/Recall tradeoff on W5: img-wikipedia
Left: directly embedding multimodal data into a vector space, right: unifying via OCR before embedding.

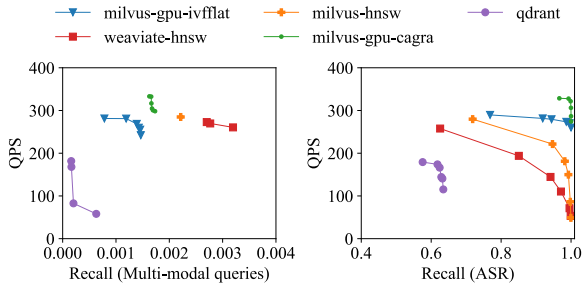


Figure 12: QPS/Recall tradeoff on W6: librispeech_asr
Left: directly embedding multimodal data into a vector space, right: unifying via ASR before embedding.

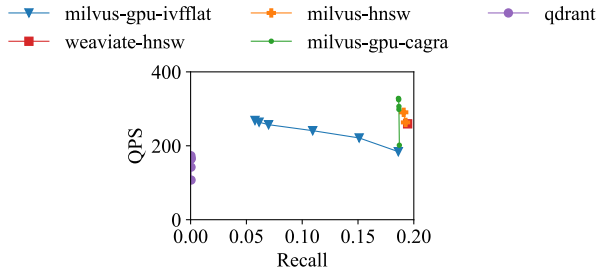


Figure 13: QPS/Recall tradeoff on W7: gpt4vision
Embedding multimodal data into a vector space.

with HNSW index > Qdrant Standard mode. Consequently, indexes running with the GPU indexes exhibit notably better performance.

5.5.3 Multi-vector queries performance. Since only Milvus supports multi-vector retrieval, Figure 14 solely illustrates the performance of the webvid dataset on various indexes within Milvus. Apart from Milvus with the GPU_IVF_PQ index, the recall for other indexes hovers around 0.72, which might be attributed to differences in the re-ranking strategies employed.

5.5.4 Big queries performance. Figure 15 presents the results of the HNSW index on Weaviate and the HNSW and GPU_IVF_PQ indexes on Milvus. The Recall of the GPU_IVF_PQ index is significantly lower than that of the HNSW index. For the two workloads with different vector dimensions, the HNSW index achieves a Recall close to 1.0, while the GPU_IVF_PQ index only reaches a maximum Recall of 0.237 for the 3072-dimensional vectors and 0.299 for the 1536-dimensional vectors. Additionally, it can be observed that an increase in dimensionality leads to a decrease in QPS. Moreover, under the same resources and index of HNSW, Milvus exhibits a higher throughput rate compared to Weaviate.

5.6 Other operation performance

In this section, we delve into the performance of other essential operations in vector databases, namely insert, update, and delete. These operations are crucial for dynamic vector data management and are often performance bottlenecks in real-world applications. We evaluate the latency of these operations across different vector dimensions and data types to provide insights into their efficiency and scalability. The experiments were conducted using the HNSW (Hierarchical Navigable Small World) index with parameters M=36 and efConstruction=200. The results for vector dimensions 384 and 1024 were obtained on a hardware configuration with 16 cores and 64GB of memory, while the results for dimensions 1536 and 3072 were obtained on a more robust setup with 32 cores and 128GB of memory.

5.6.1 Insert Performance. The insert operation's latency is a critical metric for scenarios where large volumes of vector data need to be ingested into the database. As shown in Table 3, the inclusion of a filter in the Milvus HNSW algorithm increases the insert latency. This suggests that the additional processing required for filtering impacts the efficiency of data insertion. Additionally, there is a general trend of increasing latency with higher dimensionalities, with Qdrant showing the most significant increase. This is likely due to the increased computational complexity of processing higher-dimensional vectors.

5.6.2 Update Performance. The update operation's latency is essential for applications that require frequent modifications to vector data. Our results indicate that the update latency follows a similar trend to insert latency, with higher dimensionalities leading to increased latency. Notably, Milvus HNSW exhibits higher update latency compared to insert, indicating that updating vectors is more computationally intensive than inserting them. In contrast, Weaviate HNSW shows a slightly faster update latency compared

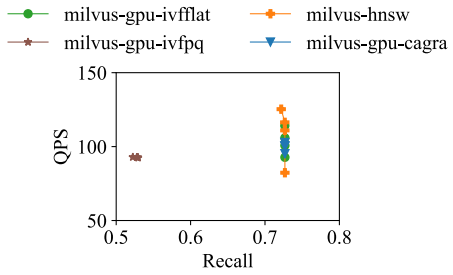


Figure 14: QPS/Recall tradeoff for multi-vector query on W8: webvid

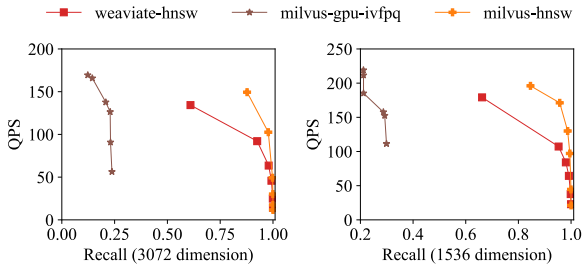


Figure 15: QPS/Recall tradeoff on W9&W10: dbpedia-entities
Left: Big query of dimension 3072, right: dimension 1536.

to insert, suggesting that its internal data structures and algorithms are optimized for updates.

5.6.3 Delete Performance. The delete operation's latency is crucial for scenarios where vector data needs to be removed or pruned from the database. Our findings show that the delete operation consistently demonstrates the lowest latency across all systems and dimensions, indicating that vector removal is the most efficient operation. However, an anomaly was observed in Milvus HNSW at 1536 dimensions compared to 3072 dimensions, where the former exhibits a slightly higher latency.

5.7 Experimental results summary

First, we observe that the quality of embedding heterogeneous data significantly influences the retrieval recall of complex queries. Existing unified embedding models face challenges in accurately aligning audio and text, leading to a compound query recall rate of less than 0.4% across three databases.

Secondly, the Amazon dataset, with the lowest filter ratio of 0.1%, encounters the most significant decline in query-per-second (QPS) performance, plummeting by more than 53% in Milvus's HNSW index. We attribute this decrease to Milvus transitioning from approximate to exact searches when the filter ratio falls below 7%. In contrast, Weaviate and Qdrant employ unique filtering approaches. When Weaviate handles filter queries, the QPS diminishes rapidly due to its integration of inverted indexing with HNSW for query processing. This method involves using an inverted index to precisely identify and store data points meeting filtering criteria in an allow_list, retrieving only the nearest neighbors from this refined

Table 3: Insert, update, and delete latency.

Operator	Algorithm / Dimension	Milvus HNSW	Weaviate HNSW	Qdrant
Insert Latency (ms)	384-filter	4.73	43.27	73.51
	384	3.73	20.97	37.55
	1024	4.36	30.91	167.80
	1536	5.66	23.26	-
	3072	6.36	37.29	-
Update Latency (ms)	384-filter	5.28	8.46	68.52
	384	4.17	7.22	52.10
	1024	4.59	12.81	169.99
	1536	5.02	19.73	-
	3072	7.89	29.86	-
Delete Latency (ms)	384-filter	2.46	1.4	71.92
	384	2.19	0.98	34.57
	1024	2.25	1.39	120.09
	1536	2.40	0.95	-
	3072	2.08	1.57	-

list. On the other hand, when Qdrant executes filter queries, the QPS almost drops to zero, while maintaining stable recall. This is because Qdrant filters out navigational points, disrupting the HNSW structure and necessitating the establishment of new costly edges to compensate for this disruption.

Thirdly, a notable decrease in recall to less than 40% is noted when utilizing Milvus's IVFPQ algorithm for retrieving high-dimensional data at 1536 and 3072 dimensions. This decline is attributed to the limitations of clustered algorithms in high-dimensional spaces, where they encounter challenges in distinguishing closely positioned data points, consequently impacting retrieval efficiency.

Lastly, heightened vector dimensions significantly raise the latency of retrieval, insertion, and update operations as a result of heightened computational complexity and resource requirements, thereby prolonging overall processing durations.

6 CONCLUSION

In this paper, we presented BigVectorBench, the first benchmark suite explicitly designed for vector databases. We quantitatively explained why modeling heterogeneous data embedding and compound query handling abilities in benchmarking vector databases is essential. We meticulously designed and developed an extensible benchmark framework tailored for vector databases. BigVectorBench comprises five datasets, ten workloads, four fundamental compound query types, and various variations.

Our extensive evaluation across leading-edge vector databases, including Milvus [93], Weaviate [3], and Qdrant [5] reveals that the primary bottleneck in serving a user query lies in the embedding latency rather than the latency of similarity searches. Furthermore, we have observed that compound queries have a significant impact on both the query per second and recall of vector databases, resulting in a halving of the recall. The results highlight a critical need within current vector databases for a unified embedding model to align multimodal data and decrease embedding latency effectively. Moreover, there is an evident demand for advanced indexing mechanisms that enhance the handling of compound queries.

REFERENCES

[1] 2007. Tesseract ocr tool. https://tesseract-ocr.github.io/

[2] 2015. Annoy. https://github.com/spotify/annoy

[3] 2016. Weaviate. https://github.com/weaviate/weaviate

[4] 2019. Milvus. https://github.com/milvus-io/milvus

[5] 2021. Qdrant. https://github.com/qdrant/qdrant

[6] 2022. Image wikipedia dataset. https://huggingface.co/datasets/israfelsr/img-wikipedia-simple

[7] 2024. OpenAI embeddings. https://platform.openai.com/docs/guides/embeddings

[8] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. *Advances in neural information processing systems* 28 (2015).

[9] Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 793–801.

[10] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *arXiv preprint arXiv:1804.06829* (2018).

[11] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2017. ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In *International conference on similarity search and applications*. Springer, 34–49.

[12] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.

[13] Artem Babenko and Victor Lempitsky. 2016. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2055–2063.

[14] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems* 33 (2020), 12449–12460.

[15] Max Bain, Arsha Nagrani, Gül Varol, and Andrew Zisserman. 2021. Frozen in Time: A Joint Video and Image Encoder for End-to-End Retrieval. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 1728–1738.

[16] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1335–1349. https://doi.org/10.1145/3318464.3389742

[17] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216* (2024).

[18] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Jeffery Zhu, Jason Li, Lintao Zhang, and Jingdong Wang. 2018. *SPTAG: A library for fast approximate nearest neighbor search*. https://github.com/Microsoft/SPTAG

[19] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems* 34 (2021), 5199–5212.

[20] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. 2020. Uniter: Universal image-text representation learning. In *European conference on computer vision*. Springer, 104–120.

[21] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 615–621. https://doi.org/10.18653/v1/N18-2097

[22] Limeng Cui, Suhang Wang, and Dongwon Lee. 2020. SAME: sentiment-aware multi-modal embedding for detecting fake news. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (Vancouver, British Columbia, Canada) *(ASONAM '19)*. Association for Computing Machinery, New York, NY, USA, 41–48. https://doi.org/10.1145/3341161.3342894

[23] 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.

[24] Sanjoy Dasgupta and Kaushik Sinha. 2013. Randomized partition trees for exact nearest neighbor search. In *Conference on learning theory*. PMLR, 317–337.

[25] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.

[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[27] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*. 577–586.

[28] Xiaoyi Dong, Jianmin Bao, Ting Zhang, Dongdong Chen, Gu Shuyang, Weiming Zhang, Lu Yuan, Dong Chen, Fang Wen, and Nenghai Yu. 2022. CLIP Itself is a Strong Fine-tuner: Achieving 85.7% and 88.0% Top-1 Accuracy with ViT-B and ViT-L on ImageNet. *arXiv preprint arXiv:2212.06138* (2022).

[29] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[30] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[31] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2017. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).

[32] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. *ArXiv* abs/2312.10997 (2023). https://api.semanticscholar.org/CorpusID:266359151

[33] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2946–2953.

[34] Allen Gersho and Robert M Gray. 2012. *Vector quantization and signal compression*. Vol. 159. Springer Science & Business Media.

[35] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. 2023. Imagebind: One embedding space to bind them all. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 15180–15190.

[36] Giovanni Grano, Andrea Di Sorbo, Francesco Mercaldo, Corrado A. Visaggio, Gerardo Canfora, and Sebastiano Panichella. 2017. Android apps and user feedback: a dataset for software evolution and quality improvement. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics* (Paderborn, Germany) *(WAMA 2017)*. Association for Computing Machinery, New York, NY, USA, 8–11. https://doi.org/10.1145/3121264.3121266

[37] Robert Gray. 1984. Vector quantization. *IEEE Assp Magazine* 1, 2 (1984), 4–29.

[38] Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE transactions on information theory* 44, 6 (1998), 2325–2383.

[39] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, Zhenshan Cao, Yanliang Qiao, Ting Wang, Bo Tang, and Charles Xie. 2022. Manu: a cloud native vector database management system. *Proc. VLDB Endow.* 15, 12 (aug 2022), 3548–3561. https://doi.org/10.14778/3554821.3554843

[40] 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*. PMLR, 3887–3896.

[41] Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, Maximilian Werk, Nan Wang, and Han Xiao. 2023. Jina Embeddings 2: 8192-Token General-Purpose Text Embeddings for Long Documents. arXiv:2310.19923 [cs.CL]

[42] Felix Hamborg, Norman Meuschke, Corinna Breitinger, and Bela Gipp. 2017. news-please: A Generic News Crawler and Extractor. In *Proceedings of the 15th International Symposium of Information Science* (Berlin). 218–223. https://doi.org/10.5281/zenodo.4120316

[43] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. 2014. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567* (2014).

[44] Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5713–5722.

[45] 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.

[46] Michael E Houle and Michael Nett. 2014. Rank-based similarity search: Reducing the dimensional dependence. *IEEE transactions on pattern analysis and machine intelligence* 37, 1 (2014), 136–150.

[47] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[48] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane A. Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Few-shot Learning with Retrieval Augmented Language Models. *ArXiv* abs/2208.03299 (2022). https://api.semanticscholar.org/CorpusID:251371732

[49] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems* 32 (2019).

[50] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

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

[52] Alexandros Komninos and Suresh Manandhar. 2016. Dependency based embeddings for sentence classification tasks. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 1490–1500.

[53] Sanjay Kukreja, Tarun Kumar, Vishal Bharate, Amit Purohit, Abhijit Dasgupta, and Debashis Guha. 2023. Vector Databases and Vector Embeddings-Review. In *2023 International Workshop on Artificial Intelligence and Image Processing (IWAIIP)*. 231–236. https://doi.org/10.1109/IWAIIP58158.2023.10462847

[54] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 793, 16 pages.

[55] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488.

[56] Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, et al. 2020. Oscar: Object-semantics aligned pre-training for vision-language tasks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXX 16*. Springer, 121–137.

[57] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281* (2023).

[58] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[59] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in neural information processing systems* 32 (2019).

[60] Chenxu Luo and Alan L Yuille. 2019. Grouped spatial-temporal aggregation for efficient action recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*. 5512–5521.

[61] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.

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

[63] Adam C. Mater and Michelle L. Coote. 2019. Deep Learning in Chemistry. *Journal of Chemical Information and Modeling* 59, 6 (2019), 2545–2559. https://doi.org/10.1021/acs.jcim.9b00266 arXiv:https://doi.org/10.1021/acs.jcim.9b00266

[64] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[65] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[66] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. MTEB: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316* (2022).

[67] Marius Muja and David Lowe. 2009. Flann-fast library for approximate nearest neighbors user manual. *Computer Science Department, University of British Columbia, Vancouver, BC, Canada* 5, 6 (2009).

[68] Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence* 36, 11 (2014), 2227–2240.

[69] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 188–197.

[70] Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. 2024. Nomic Embed: Training a Reproducible Long Context Text Embedder. arXiv:2402.01613 [cs.CL]

[71] Julius Odede and Ingo Frommholz. 2024. JayBot – Aiding University Students and Admission with an LLM-based Chatbot. In *Proceedings of the 2024 Conference on Human Information Interaction and Retrieval* (Sheffield, United Kingdom) *(CHIIR '24)*. Association for Computing Machinery, New York, NY, USA, 391–395. https://doi.org/10.1145/3627508.3638293

[72] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of vector database management systems. *The VLDB Journal* 33, 5 (2024), 1591–1615.

[73] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Vector Database Management Techniques and Systems. In *Companion of the 2024 International Conference on Management of Data*. 597–604.

[74] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: an ASR corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 5206–5210.

[75] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[76] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.

[77] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. arXiv:2103.00020 [cs.CV] https://arxiv.org/abs/2103.00020

[78] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, Melbourne, Australia, 784–789. https://doi.org/10.18653/v1/P18-2124 arXiv:1806.03822 [cs.CL]

[79] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Jian Su, Kevin Duh, and Xavier Carreras (Eds.). Association for Computational Linguistics, Austin, Texas, 2383–2392. https://doi.org/10.18653/v1/D16-1264 arXiv:1606.05250 [cs.CL]

[80] 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.

[81] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. 2019. wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862* (2019).

[82] Christoph Schuhmann and Peter Bevan. 2023. https://huggingface.co/datasets/laion/220k-GPT4Vision-captions-from-LIVIS.

[83] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.

[84] Harsha Vardhan Simhadri. 2023. big-ann-benchmarks: Framework for evaluating ANNS algorithms on billion scale datasets. https://github.com/harsha-simhadri/big-ann-benchmarks

[85] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[86] Hao Tan and Mohit Bansal. 2019. Lxmert: Learning cross-modality encoder representations from transformers. *arXiv preprint arXiv:1908.07490* (2019).

[87] Jiajie Tan, Jinlong Hu, and Shoubin Dong. 2023. Incorporating entity-level knowledge in pretrained language model for biomedical dense retrieval. *Computers in Biology and Medicine* 166 (2023), 107535. https://doi.org/10.1016/j.compbiomed.2023.107535

[88] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=wCu6T5xFjeJ

[89] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu, et al. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* 12 (2016).

[90] Chenxi Wang and Xudong Luo. 2022. A Legal Question Answering System Based on BERT. In *Proceedings of the 2021 5th International Conference on Computer Science and Artificial Intelligence* (Beijing, China) *(CSAI '21)*. Association for Computing Machinery, New York, NY, USA, 278–283. https://doi.org/10.1145/3507548.3507591

[91] Changhan Wang, Yun Tang, Xutai Ma, Anne Wu, Dmytro Okhonko, and Juan Pino. 2020. fairseq S2T: Fast Speech-to-Text Modeling with fairseq. In *Proceedings of the 2020 Conference of the Asian Chapter of the Association for Computational Linguistics (AACL): System Demonstrations*.

[92] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. 2012. Scalable k-nn graph construction for visual descriptors. In *2012 IEEE*

Conference on Computer Vision and Pattern Recognition. IEEE, 1106–1113.

[93] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.

[94] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. 2018. Temporal segment networks for action recognition in videos. *IEEE transactions on pattern analysis and machine intelligence* 41, 11 (2018), 2740–2755.

[95] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text Embeddings by Weakly-Supervised Contrastive Pre-training. *arXiv preprint arXiv:2212.03533* (2022).

[96] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Improving Text Embeddings with Large Language Models. *arXiv preprint arXiv:2401.00368* (2023).

[97] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Multilingual E5 Text Embeddings: A Technical Report. *arXiv preprint arXiv:2402.05672* (2024).

[98] Wenping Wang, Yunxi Guo, Chiyao Shen, Shuai Ding, Guangdeng Liao, Hao Fu, and Pramodh Karanth Prabhakar. 2023. Integrity and Junkiness Failure Handling for Embedding-based Retrieval: A Case Study in Social Network Search. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 3250–3254. https://doi.org/10.1145/3539618.3591831

[99] Hang Xie and Tiffany Y. Tang. 2018. Vector projection on lyrics and user comments for a lightweight emotion-aware chinese music recommendation system. In *Proceedings of the 1st International Conference on Big Data Technologies* (Hangzhou, China) *(ICBDT '18)*. Association for Computing Machinery, New York, NY, USA, 88–94. https://doi.org/10.1145/3226116.3226132

[100] Michihiro Yasunaga, Armen Aghajanyan, Weijia Shi, Rich James, Jure Leskovec, Percy Liang, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2023. Retrieval-augmented multimodal language modeling. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 1659, 15 pages.

[101] Peter N Yianilos. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Soda*, Vol. 93. 311–21.

[102] Jianfeng Zhan, Lei Wang, Wanling Gao, Hongxiao Li, Chenxi Wang, Yunyou Huang, Yatao Li, Zhengxin Yang, Guoxin Kang, Chunjie Luo, et al. 2024. Evaluatology: The science and engineering of evaluation. *BenchCouncil Transactions on Benchmarks, Standards and Evaluations* 4, 1 (2024), 100162.

[103] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *NIPS*.

[104] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2024. Chat2Data: An Interactive Data Analysis System with RAG, Vector Databases and LLMs. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4481–4484.