




Vector Databases and Language Models: Synergies and Challenges

Toni Taipalus^(✉) 

Tampere University, Tampere, Finland
`toni.taipalus@tuni.fi`

Abstract. Vector databases are a critical component in modern system infrastructures. In this study, we discuss the principles behind vector database management systems, with a focus on their features, the concept of vector embeddings, and similarity search mechanisms. Furthermore, we examine the synergies between vector databases and language models, which rely on vector embeddings for semantic search and retrieval-augmented generation. We also discuss the challenges arising from the integration of language models with vector databases. Through this discussion, we aim to provide early-stage researchers with an overview of the integration of vector databases and language models.

Keywords: vector databases · vectorization · database · data management · language model · retrieval-augmented generation

1 Introduction

Language models, despite revolutionizing natural language processing, face well-known limitations: they can hallucinate false facts [5], struggle with out-of-date knowledge, and incur escalating costs as their parameters grow [8]. Integrating language models with vector databases offers a compelling solution to these issues [8].

In the retrieval-augmented generation (RAG) paradigm, a language model's parametric knowledge is supplemented with non-parametric memory from an external vector database. This synergy enables the model to fetch relevant information on the fly, which grounds the responses in up-to-date data and potentially reduces hallucinations. Early work on *k-nearest-neighbor* language models demonstrated that augmenting a neural language model with a vector-indexed datastore of examples dramatically improved the model's ability to recall rare factual patterns, consequently reducing perplexity on long-tail content without model re-training [10]. Modern RAG systems build on this idea by using neural dense embeddings to retrieve semantically relevant documents which the language model then uses to produce more informed outputs [11]. In specific knowledge-intensive tasks, retrieval augmentation has enabled smaller language models to approach the performance of substantially larger models. For example, the RETRO model has been shown to achieve comparable performance to

GPT-3 on certain knowledge-intensive benchmarks, while using 25 times fewer parameters [2].

In this study, we discuss system-level underpinnings of synergies similar to the one above, focusing on how vector databases support neural retrieval for language models. We examine dense embedding generation, indexing and storage in vector databases, query latency considerations, integration pipelines for language models and vector databases, the current challenges, and emerging research opportunities. The discussion strives for domain-agnosticism, and targets data management and retrieval aspects relevant to many applications.

2 Background Concepts

2.1 Vector Embeddings

Vector embeddings are numerical representations of data elements. Data objects such as words, images, or nodes in a graph can be mapped into continuous vector spaces. These embeddings capture semantic or structural relationships, and allow for the search of vectors that are similar to (but not the same as) another vector. In this study, we use two-dimensional vectors for illustration purposes and simplicity, but such vector may have dimensions (i.e., elements) in the hundreds or thousands. As an example, Fig. 1a shows the vectors of two plays, *PA* and *PB*, in a two-dimensional vector space. The elements of the vectors correspond to the amount of comedy and tragedy in the plays. Based on the positions of the vectors in the vector space, we can see that play *PA* is relatively comic and not very tragic, and that play *PB* is the opposite.

Using text data as an example of creating vector embeddings from data objects, the creation of vector embeddings is grounded in the distributional hypothesis, which posits that linguistic items with similar distributions have similar meanings. This is the basic principle of models like *Word2Vec*, which learns embeddings by predicting a word based on its context, or predicting surrounding words given a target word [15]. Such models utilize large corpora to capture co-occurrence statistics, resulting in vector spaces where semantic relationships are reflected in geometric proximity. As an example, when searching for “vitamin deficiency symptoms”, a suitable embedding model might retrieve documents about iron deficiency and B12 anemia. A poor embedding model might return blog posts about “vitamin shopping” or “best supplements in 2025”. All these are technically close, but not all are semantically useful.

Different embedding techniques are also better suited for different geometric proximity calculations (cf. Figure 1b). Mismatch between vector geometry and similarity function can lead to reduced effectiveness. Furthermore, Embeddings trained on in-domain data (e.g., biomedical text for a health app) drastically improve recall (i.e., finding all relevant items) and precision (i.e., avoiding irrelevant ones).

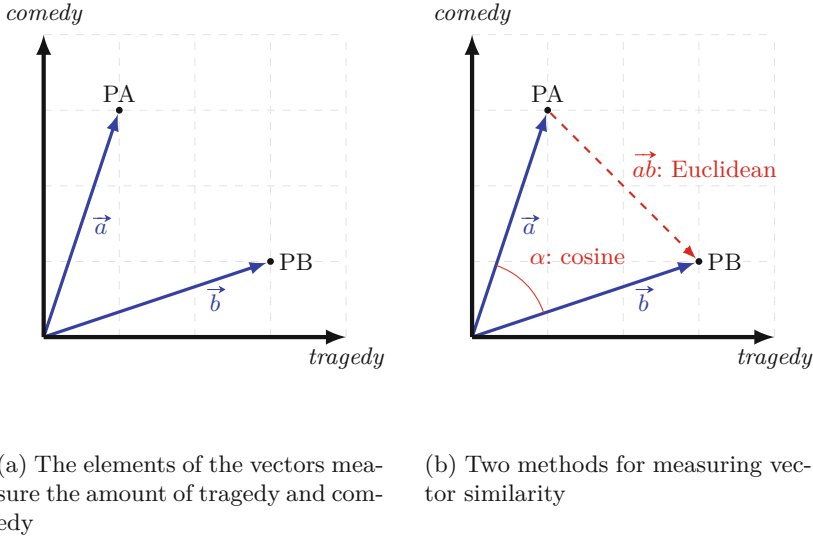


Fig. 1. Two-dimensional vector space with two vectors and two methods of measuring similarity: Euclidean distance and cosine similarity; adapted [19]

2.2 Vector Databases

Vector databases, or rather vector database management systems (VDBMS), can provide the means of efficiently storing vectors and metadata associated with them in forms of different vector indices. As with data objects other than vectors, indices form the basis of efficient data retrieval. Additionally, VDBMSs can provide different ways of measuring vector similarity. Commonly used methods are Euclidean distance, cosine similarity, and inner product. The two former are illustrated in Fig. 1b. One can likely see the implications of different similarity search methods in the figure.

In addition to vector-specific indices and searching, different vector databases provide different functionality, ranging from software libraries (e.g., FAISS [9]), to fully-fledged DBMSs with role-based access control, concurrency, and replication and sharding (e.g., Milvus [22]). Several DBMSs following some other database paradigm have also adopted features for vector data management, for example PostgreSQL, Redis, and MongoDB. Currently, the maximum number of vector dimensions in these systems are measured in thousands, while dedicated systems such as Milvus, Pinecone, Weaviate, and Manu [4] can manage dimensions in the tens of thousands.

2.3 Language Models

Simplified, language models function by converting textual input into high-dimensional vector representations (e.g., \mathbb{R}^{768}), which enables them to capture semantic relationships and contextual nuances. These models process input

within a fixed-size context window, which defines the maximum number of tokens the model can consider at once. Expanding this context window enhances the model’s ability to understand longer inputs. However, computational complexity of standard self-attention in transformers is $O(n^2)$ with respect to the sequence length n . Newer architectures such as Longformer [1] are specifically designed to reduce this cost for long-context scenarios.

Training and fine-tuning language models are typically resource-intensive processes that require substantial computational power and time. For example, training large-scale models can incur costs running into millions of dollars, making it impractical for many applications. To mitigate these challenges, vector databases are employed to provide external context to language models without the need for retraining. By storing precomputed vector embeddings of relevant data, these databases enable efficient retrieval of relevant data objects based on semantic similarity. When a prompt is issued, the system retrieves the most relevant vectors from the database and incorporates them into the model’s context window, enhancing the model’s responses with up-to-date and domain-specific information.

3 Supplementing Retrieval with Vector Databases

RAG refers to techniques that combine an language model with a retrieval mechanism to incorporate external knowledge. A typical RAG system first encodes a user query into a vector representation, then performs a similarity search in a vector database of background documents, and finally feeds the top-ranked retrieved documents (or their content) into the language model’s context before generating the answer [11] (cf. Figure 2). This pipeline effectively gives the language model access to an external knowledge base in real-time. The approach was introduced in knowledge-intensive NLP tasks [11], showing that a language model (BART in their case) augmented with a learned retriever outperformed fully-parametric models on open-domain QA benchmarks.

Subsequent systems have strengthened this paradigm. For example, Atlas [6], a pretrained retrieval-augmented model achieved higher accuracies on QA with 50 times fewer parameters than a 540 billion-parameter PaLM model by carefully training the retriever and generator together on knowledge tasks. Another line of work from the database community treats the vector store as a reliable long-term memory for language models [24]: rather than packing all world knowledge into model weights, one can store factual data in a vector database and let the model retrieve it as needed [8]. This design not only improves factual accuracy, but also allows updating the knowledge base without retraining the language model, which is a major advantage for keeping up with evolving information.

In practice, retrieval augmentation has been shown to mitigate hallucinations in several benchmark tasks, provided the retrieved information is relevant and accurate [8]. Even at inference time, an language model can be queried in a semi-open-book manner. That is, in experimental setups, frozen language models have demonstrated improved rare-token prediction accuracy by retrieving nearest neighbor tokens from a vectorized representation of the training corpus [10].

Across the studies mentioned, a clear picture emerges: dense neural retrieval is a key enabler for making language models more knowledgeable, accurate, and efficient by offloading memory to an external vector database.

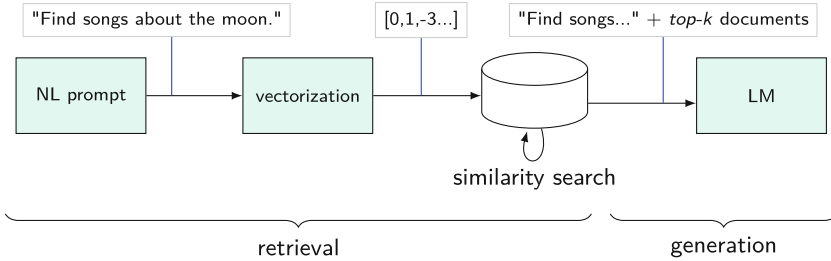


Fig. 2. The general principle of retrieval-augmented generation: the vectorized natural language (NL) prompt is used for vector similarity search to find n closest documents, which are then used in tandem with the natural language prompt for the language model (LM), enhancing the prompt with context

4 Vector Storage and Retrieval

Central to neural retrieval is the use of dense embeddings. Instead of sparse keyword matches, RAG pipelines represent text as high-dimensional vectors (hundreds or even thousands of dimensions) generated by transformer encoders [21] or language model embeddings. These vectors capture semantic similarity, meaning that a query vector will lie close (e.g., in cosine or Euclidean distance) to vectors of documents on the same topic to retrieve relevant information even when there are no shared keywords [2]. Storing and searching through millions or billions of such vectors is the core function of a vector database. This task is non-trivial: the vectors live in a continuous space lacking obvious structure, and brute-force search would be prohibitively slow, as each distance computation involves hundreds of multiplications.

Instead, approximate nearest neighbor (ANN) algorithms are used to index the embeddings and accelerate queries. One popular approach is building a small-world graph index (HNSW, or hierarchical navigable small world), where each vector is a node connected to its neighbors in such a way that a greedy graph traversal yields a near-optimal set of results [14] (cf. Figure 3). Malkov and Yashunin’s HNSW method exemplifies this, allowing sub-millisecond retrieval on million-scale datasets with high recall by navigating a hierarchical graph structure instead of exhaustive scanning [14].

Another class of methods uses vector quantization to compress and partition the space. Product quantization (PQ) compresses each vector into a short code by splitting the space into subspaces — dramatically reducing memory usage and enabling fast coarse search at some cost to accuracy [7]. Modern vector

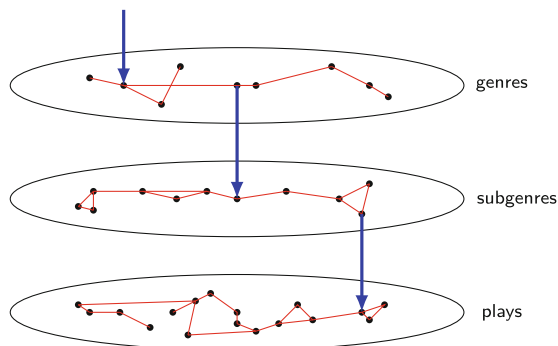


Fig. 3. Basic operating principle of an HNSW index: an arbitrary number of layers of indices are used to limit the search space; vector search starts at the top layer (blue arrow) from a random vector; each vector stores a list of pointers to a small set of neighbor vectors (red lines); ANN is used to find the most suitable match in a relatively small vector space, which then points to the next layer; again ANN is used in a subset of vectors to find the most suitable match, which is again followed to find the actual search results (Color figure online)

databases often combine different techniques: for example, the FAISS library [9] provides inverted file indexes with product quantization and HNSW graphs, supporting billion-scale similarity search on GPUs [9]. SPANN exemplifies a two-tier index architecture optimized for billion-scale corpora, although such systems remain more common in research contexts than production use today [3]. The common goal of these structures is to balance recall (retrieving true nearest neighbors) with efficiency (time and computer memory).

In practice, an ANN index can retrieve top-k neighbors from a million-vector set in just a few milliseconds with 95% recall, a sweet spot for many language model augmentation scenarios. Vector databases incorporate these algorithms under the hood, managing the indexing, compression, and search operations transparently. For instance, Milvus [22] is an open-source vector DBMS that offers multiple index types and automatically chooses an index depending on data scale and latency requirements. By leveraging such indices in optimized settings, vector databases can support similarity queries with latencies approaching those of traditional keyword search engines.

Finally, many VDBMSs support hybrid search or hybrid operators, meaning that in addition to the query vector, a set of more traditional conditions are imposed on the metadata, such as `price > 50` [18]. This limits the vector search to a subset of vectors in the vector space. Vector indices can also be constructed in subsets of vectors (i.e., shards), and metadata used in assigning certain vectors to certain shards, e.g., electronics to one shard, utilities to another.

5 Performance Considerations

Building a high-performance vector database-backed language model pipeline requires careful system-level design. One consideration is the latency budget for retrieval. In interactive applications (e.g. a question-answering chatbot), the vector lookup must be relatively fast not to bottleneck the language model’s response. A typical vector search on tens of thousands of embeddings can execute in a few milliseconds with an optimized ANN index in memory [3]. However, if the knowledge corpus is very large (e.g., hundreds of millions of entries), keeping the entire index in RAM may be unfeasible. In such cases, systems turn to disk-based or distributed solutions: SPANN, for example, demonstrates that a hybrid disk/RAM index was shown to serve a billion-scale vector corpus with only 64GB of memory, achieving query latencies on the order of tens of milliseconds with high recall [3].

Another approach is to exploit hardware parallelism. FAISS and other libraries can batch distance computations to utilize GPUs efficiently [9], yielding significant speedups for large query loads. The vector database acts as the orchestrator to route similarity computations to the appropriate hardware and to manage caching of vector data. Caching is indeed a valuable optimization in cases when certain queries or documents are frequently accessed, as the vector database can cache their embeddings or results in faster storage. In an end-to-end RAG pipeline, there is also a trade-off between retrieval time and generation time. Language model inference is typically the slower component (especially for large language models), so one might tolerate, say, 50 ms of retrieval delay from the vector database, if it substantially improves the quality of a generation that takes 2 s.

System designers often overlap retrieval with other stages (e.g. start the language model on the user query while concurrently fetching documents, then concatenate results when ready) to hide latency. The integration between language model and vector database can be tight or loose. A tight integration might use the language model’s internal representation to continually fetch new information mid-generation (iterative retrieval), whereas a loose integration uses a fixed retrieved set obtained before generation. Many practical implementations use an orchestration layer (such as a middleware or libraries like LangChain) to manage the sequence. That is, embed the query, query the vector store, retrieve top-N texts, and finally construct an augmented prompt for the language model. Each interface crossing between the language model and the database incurs overhead, so some research explores training the retriever and generator jointly so that the two components hand off information more fluidly [2].

Ensuring scalability requires monitoring how retrieval performance scales with data size. Vector databases often support sharding or partitioning of the index across nodes to handle very large corpora, with a slight loss in recall due to partition boundaries [3] (cf. Figure 4). Another issue is consistency in distributed settings: when multiple language model instances share access to a vector database, inserts to the database may need to be immediately visible to all models, which calls for transactional or eventually-consistent replication pro-

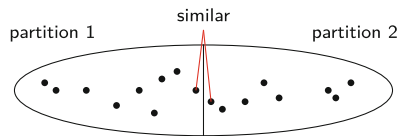


Fig. 4. Similar vectors near the partition boundaries can be missed if only a part of the partitions are searched

protocols in the vector database. Such requirements pose limitations to the selection of the vector database.

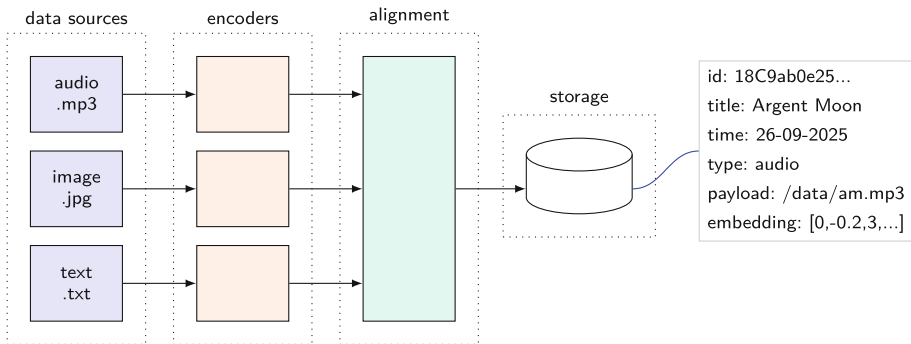


Fig. 5. Vectorization allows the storage and querying of multi-modality data in a single format: vectors; multi-modality data such as images and text are encoded with respective encoders into vector embeddings, these embeddings are then aligned into a single vector space, ensuring that vector embeddings of similar data objects are close to each other; the vectors are then stored into a vector database; some techniques such as CLIP and CLAP already provide the alignment without need for additional processing though other software libraries or models

6 Future Research

Beyond text, vector databases can store embeddings of images, audio, code, graphs, etc. An exciting current research direction is enabling language models to retrieve not just text documents but other modalities (images or knowledge graph substructures) to ground their understanding (Fig. 5). Multi-modality poses additional challenges to already recognized limitations in embedding models, data quality, and retrieval accuracy [23]. Early research explored unified systems that support hybrid queries combining vector similarity with structured constraints (e.g. temporal or graph-based conditions) [3]. In this regard, vector databases contribute to addressing challenges reminiscent of those seen in multi-model databases, such as the absence of a unified query language, which have been discussed in prior work [13].

When an language model’s internal knowledge conflicts with the retrieved data, the result can be an inconsistent or confusing answer. Mechanisms to detect and resolve these conflicts are needed. One idea is to train language models to defer to retrieved evidence in critical domains, i.e., effectively learning a form of truth alignment with the external database. Another aspect is keeping the database in sync with the world: unlike static model weights, a vector store can be updated continuously. Techniques for incremental indexing (i.e., updating indexes without full rebuilds) and for handling concept drift (i.e., when new data shifts the embedding space) will be crucial for systems where real-time consistency is required [3].

From hardware perspective, GPUs already excel at vector calculations due to parallelism and floating-point performance, yet there is interest in quantized or compressive transformers that integrate vector search natively, as well as hardware like dedicated ANN accelerators [17]. In our opinion, there is a need for more comprehensive benchmarks that evaluate the combined system of a language model and a vector database holistically. While components are individually benchmarked (e.g. ANN benchmarks for vector search [9], and natural language processing benchmarks for language model accuracy), the field lacks standardized tasks that measure end-to-end performance, including response quality and latency under varying loads, similarly to relational DBMS benchmarking. Developing such benchmarks would drive research into more efficient and effective integration.

Privacy and security pose additional research avenues. Embedding textual data could leak private information, and external retrieval might introduce adversarial inputs to the language model. Emerging work has begun to explore privacy-preserving vector retrieval mechanisms, such as secure ANN and filtered retrieval, which are likely to become increasingly important as such systems are adopted in sensitive domains. It remains an open questions whether this is a training-related challenge, whether such privacy issues should be ensured with guardrails outside model training, or perhaps with hybrid queries, which account not only similarity search but also search conditions on metadata (cf. right-hand side of Fig. 5).

Finally, even though vectors are not a novel way of storing data, the field has advanced rapidly in terms of popularization and usability of different tools and systems. In our opinion, vector databases and adjacent technologies are more than ripe for more *applied research* in different and exciting domains. Although we have seen concepts are opportunities in fields such as healthcare [16], cyber security [20], and finance [12], current vector data management techniques provide a vast frontier of opportunities that now only lacks more imagination.

7 Conclusion

Vector databases and large language models together form a powerful architecture for generative artificial intelligence. The dense neural retrieval capabilities of vector databases complement the generative features of language models and

enable systems that not only produce coherent language, but reason over timely, model-external knowledge. In this study, we reviewed how dense embeddings serve as the lingua franca between language models and vector database, and how efficient indexing and storage make real-time retrieval feasible at scale. Key system challenges such as reducing query latency, building robust integration pipelines, and ensuring scalability have seen rapid progress, yet continue to face challenges. As future research addresses the outlined directions which span from better indexes and multi-modal retrieval to alignment and security, we can expect language models to become more reliable and versatile by leveraging the right database systems behind the scenes. This interdisciplinary synergy connects the advances in data management and language models.

References

1. Beltagy, I., Peters, M.E., Cohan, A.: LongFormer: the long-document transformer. arXiv preprint [arXiv:2004.05150](https://arxiv.org/abs/2004.05150) (2020)
2. Borgeaud, S., Sifre, L., et al.: Improving language models by retrieving from trillions of tokens. In: Proceedings of the 39th International Conference on Machine Learning (ICML), pp. 2206–2240 (2022)
3. Chen, Q., et al.: SPANN: highly-efficient billion-scale approximate nearest neighbor search. In: Advances in Neural Information Processing Systems (NeurIPS 2021), vol. 34, pp. 5199–5212 (2021)
4. Guo, R., et al.: Manu: a cloud native vector database management system. arXiv preprint [arXiv:2206.13843](https://arxiv.org/abs/2206.13843) (2022)
5. Huang, L., et al.: A survey on hallucination in large language models: principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.* **43**(2), 1–55 (2025)
6. Izacard, G., et al.: Atlas: few-shot learning with retrieval augmented language models. *J. Mach. Learn. Res.* **24**(251), 1–43 (2023)
7. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.* **33**(1), 117–128 (2011)
8. Jing, Z., et al.: When large language models meet vector databases: a survey (2024)
9. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with GPUs. *IEEE Trans. Big Data* **7**(3), 535–547 (2019)
10. Khandelwal, U., Levy, O., Jurafsky, D., Zettlemoyer, L., Lewis, M.: Generalization through memorization: nearest neighbor language models. In: Proceedings of the International Conference on Learning Representations (ICLR) (2020)
11. Lewis, P.S.H., et al.: Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Advances in Neural Information Processing Systems (NeurIPS 2020), vol. 33, pp. 9459–9474 (2020)
12. Liu, X., Zhu, J.: FinanceQA-Agent: a high-precision comprehensive financial technology question-answering intelligent system based on vector databases. In: 2024 International Conference on Advances in Electrical Engineering and Computer Applications (AEECA), pp. 576–582. IEEE (2024)
13. Lu, J., Holubová, I.: Multi-model databases: a new journey to handle the variety of data. *ACM Comput. Surv.* **52**(3), 1–18 (2019). <https://doi.org/10.1145/3323214>
14. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* **42**(4), 824–836 (2020)

15. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. *arXiv preprint* (2013)
16. Ng, K.K.Y., Matsuba, I., Zhang, P.C.: RAG in health care: a novel framework for improving communication and decision-making by addressing LLM limitations. *NEJM AI* **2**(1), A1ra2400380 (2025)
17. Pan, J.J., Wang, J., Li, G.: Survey of vector database management systems. *VLDB J.* **33**(5), 1591–1615 (2024)
18. Pan, J.J., Wang, J., Li, G.: Vector database management techniques and systems. In: *Companion of the 2024 International Conference on Management of Data*, pp. 597–604. ACM (2024)
19. Taipalus, T.: Vector database management systems: fundamental concepts, use-cases, and current challenges. *Cogn. Syst. Res.* **85**, 101216 (2024). <https://doi.org/10.1016/j.cogsys.2024.101216>
20. Taipalus, T., Grahn, H., Turtiainen, H., Costin, A.: Utilizing vector database management systems in cyber security. In: *Proceedings of the 23th European Conference on Cyber Warfare and Security*. Academic Conferences International Ltd (2024)
21. Vaswani, A., et al.: Attention is all you need. *Adv. Neural Inf. Process. Syst.* **30** (2017)
22. Wang, J., et al.: Milvus: a purpose-built vector data management system. In: *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, pp. 2614–2627 (2021)
23. Wang, M., et al.: Must: an effective and scalable framework for multimodal search of target modality. In: *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 4747–4759. IEEE (2024)
24. Zhang, Y., Yu, Z., Jiang, W., Shen, Y., Li, J.: Long-term memory for large language models through topic-based vector database. In: *2023 International Conference on Asian Language Processing (IALP)*, pp. 258–264. IEEE (2023)