Functional Programming Meets Pinecone: Recommending Graph Structured Documents

Georgios Drakopoulos
ICE Department
University of West Attica
0000-0002-0975-1877

Leonidas Theodorakopoulos
DEPT Department
University of Patras
0000-0002-0891-6780

Spyros Sioutas
CEID
University of Patras
0000-0003-1825-5565

Phivos Mylonas
ICE Department
University of West Attica
0000-0002-6916-3129

***Abstract*—Vector databases such as Faiss, Pinecone, Chroma, and Milvus facilitate efficient similarity evaluation between embeddings drawn from high dimensional attribute spaces. Recommendation schemes for massive document collections where each such document is internally structured as a large graph are a prime application since the plethora of connectivity patterns associated with said structure lead to long embeddings which can lead to superior performance in terms of established evaluation metrics. Moreover, documents abound with recursive patterns and immutable strings representing key-value pairs or formatted data, paving thus the way for functional implementations. Additionally, vector databases exploit the geometry inherent in data points and queries alike as graph databases rely heavily on topology, allowing flexible query design as well as embedding engineering in order to maintain efficiency against strict efficiency constraints. As a concrete case, the effect of alternative embedding schemes on precision and recall is explored in a collection consisting of documents with synthetic graph structure stored in Pinecone and processed with a Python application following the functional programming paradigm. Furthermore, an extensive overview of the aforementioned four vector databases is given.**

***Index Terms*—document recommendation; higher order embeddings; high dimensionality search; HNSW; vector databases; Pinecone; Faiss; Chroma; Milvus; dimensionality reduction; geometric analytics; functional programming; information retrieval**

I. Introduction

Mining large collections of structured documents play a central role in contemporary information retrieval (IR) in general and document retrieval in particular with models like the Boolean and the vector ones as well as numerous variants thereof having been proposed. Document representation determines the type as well as the efficiency of the queries evaluated in terms of metrics such as precision, recall, and the F1 score. Vector databases such as Pinecone are designed to hold large document collections as shown in equation (1) as a set D of $|D|$ documents in total where each document is represented by a single vector with n attributes as will be later explained. The collection size is a major factor as it determines the scalability of document retrieval methodologies.

$$D \triangleq \{d_1, \ldots, d_N\} \tag{1}$$

In this context geometry plays an important role since the similarity between vectors representing documents is measured primarily by the angle between them expressed by the cosine similarity of equation (3) or by squared differences in magnitude expressed by a norm defined on a suitable norm as shown in equation (9). Said vectors consist of heterogeneous attributes extracted from the documents themselves, frequently extracted by a machine learning (ML) model, which can be pretrained like BERT or trained for a very specific purpose at the expense of possibly considerable computational resources. Once a suitable document representation is obtained, then operations such as similarity assessment, topic modeling, or summarization can subsequently take place.

TABLE I
Notation Synopsis.

Symbol	Meaning	First in
$\triangleq$	Equality by definition	Eq. (1)
$\{s_1, \ldots, s_n\}$	Set with elements $s_1, \ldots, s_n$	Eq. (1)
$(t_1, \ldots, t_n)$	Tuple with elements $t_1, \ldots, t_n$	Eq. (13)
$\lvert\cdot\rvert$	Set or tuple cardinality functional‡	Sec. I
$\otimes$	Kronecker tensor product	Eq. (14)
$\deg(v_k)$	Degree of vertex v_k	Sec. III-C
$\lVert\cdot\rVert$	Vector norm	Eq. (3)
$\mathbf{I}_n$	$n \times n$ identity matrix	Eq. (11)
$\lvert\cdot\rvert$	Absolute value‡	Eq. (9)
$f^{(n)}(\cdot)$	n-th derivative of function f	Eq. (7)

‡ context dependent

Contemporary major document formats based on XML like .docx of Microsoft Word as described by the Office Open XML (OpenMXL or OOXML) standard[1] or Open Office .odt rely on nested tags corresponding to document elements like tables, figures, headers, footnotes, sections, their formatting options, and the relationships between them forming recursive and nested patterns. Graph patterns extracted from such documents can be subsequently form the basis for subsequenct processing such as clustering or recommendation. Moreover, these patterns can be handled through functional programming not only because of their nested form, but also because adjacency lists can be readily processed in this paradigm.

In the document vector model similarity one of the most common distance metrics is the cosine similarity shown in equation (3) which ignores vector magnitudes which is useful when queries come from an external system or API with same

[1]http://officeopenxml.com

encoding but different scaling. Still, numerical instability may occur when one vector has sufficiently small magnitude. An alternative metric is the p norm shown in equation (9) which is a generalization of the Euclidean norm defined on Hilbert spaces. Both similarity metrics allow meaningful document clustering with a relatively low computational cost.

The primary research objective of this conference paper is the development of a functional implementation in the *functools* Python module for recommending documents represented as graphs containing structural and semantic information and stored in Pinecone, a major vector database. As a concrete case the proposed scheme has been applied to a synthetic graph dataset created by the Kronecker model with various generator graphs. The principal motivation is to provide a bridge between graph models and vector databases.

The remaining of this work is structured as follows. The recent scientific literature regarding vector databases, functional programming, and graph mining is briefly reviewed in section II. The recent scientific literature regarding vector databases, functional programming, and graph mining is briefly reviewed in section II. The proposed methodology is presented in section III, whereas the results obtained are described in section IV. Possible future research directions are explored in section V. The terms *attribute* and *feature* are used interchangeably. Technical terms are explained the first time they are encountered in the text. In function definitions parameters follow arguments separated by a semicolon. Finally, table I summarizes the symbols used in this work.

II. Related Work

Vector databases such as Faiss[2] by Meta, Pinecone[3], Milvus[4], and Chroma[5] [6] are suitable for hosting and processing large document collections [1] [2]. Among their advantages are scalability [3] and flexibility [4]. Their applications include storing vectors representing linguistic patterns and their variations on Twitter [5], interacting with knowledge graphs for augmenting financial data [6], supporting large language models (LLMs) for education [7], developing bioinformatics knowledge bases [8], knowledge cache [9], and computing similarity between vectors with attributes pertaining to the MBTI of people extracted from written text [10]. The potential of relational databases to support vector operations is explored in [11] using PostgreSQL[7] as a concrete example. Vector quantization considerations are explored in [12], indexing for large data segments in [13], the nature of vector embeddings in [14]. A comprehensive survey of the field is [15] and an introductory book encompassing many topics is [16].

Graph mining is a broad field with subfield as diverse as community structure with the Boolean semiring factorization of the adjacency matrix is the focus of [17], graph anomaly detection [18], subgraph mining [19], sequential graph collaborative filtering [20], topological uncertainties [21], and graph neural networks (GNNs) [22]. Many graph operations can be considerably accelerated with moving computations to GPUs [23]. Outside the field applications include Twitter communities with multiple criteria [24], credit risk prediction with XGBoost [25], ranking cloud service providers [26], recommending cultural items [27], drug target inference [28], long data streaming [29], and clustering players in cultural games based on the Bartle taxonomy [30].

Functional programming is a programming paradigm placing heavy emphasis on function compositions operating on lists [31] [32]. Haskell[8] is a functional programming language which has been used among others for graph mining and Scala[9] supports fully the functional paradigm [33] [34]. Although Python is not purely a functional language, it has extensive tools [35] primarily through the *functools* module [36]. Functional data strucures are extensively studied in [37].

III. Methodology

A. Similarity metrics

Document representation is central in both the selection of a document retrieval model as well as in the subsequent processing including the computational infrastructure. Each document $d_i \in D$ is represented by a column vector as shown in equation (2). The n attributes of the vector representation depend on the application and they may be related to the document internal structure, its actual content including semantics and sentence building, and possible metadata such as creation timestamp and the number of changes.

$$d_i \leftrightarrow \mathbf{d}_i \triangleq \begin{bmatrix} d_i[1] & \dots & d_i[n] \end{bmatrix}^T \tag{2}$$

The cosine similarity metric of equation (3) is the primary metric used in vector databases and it is the cosine of the angle between these two vectors, paving thus the way for developing geometric analytics in document information retrieval (IR). The proper cosine value can be obtained irrespective of the scale of the two operands. Still, a small vector length in the denominator may result in numerical instability. For unitary length vectors the cosine similarity coincides with the deterministic correlation.

$$\operatorname{sim}(\mathbf{q}, \mathbf{s}) \triangleq \frac{\mathbf{q}^T \mathbf{s}}{\|\mathbf{q}\|_2 \|\mathbf{s}\|_2} = \cos\vartheta(\mathbf{q}, \mathbf{s}) \tag{3}$$

Computationally the cosine function is appealing because of its bounded nature as follows from equation (5). Additionally, it decays quadratically with respect to its argument as it can be seen from its Taylor expansion given in equation (4). This is the reason why the cosine is used as the learning rate in neural network architectures such as multilayer neural perceptrons (MLPs) and self organizing maps (SOMs).

$$\cos\theta = \sum_{k=0}^{+\infty} (-1)^{2k} \frac{\theta^{2k}}{(2k)!} \approx 1 - \frac{\theta^2}{2} \tag{4}$$

[2]https://ai.meta.com/tools/faiss/
[3]https://www.pinecone.io
[4]https://milvus.io/
[5]https://github.com/chroma-core/chroma
[6]https://www.trychroma.com/
[7]https://www.postgresql.org
[8]https://haskell.org
[9]https://scala-lang.org

In equation (3), assuming that $\mathbf{q}$ and $\mathbf{s}$ are of unitary length, the distance metric becomes a bilinear form. Linear transforms may represent an alternative topic representation or, under suitable conditions, a domain transfer knowledge operation because of the unitary constraints. The Cauchy-Schwarz-Bunyakowski (CSB) inequality of equation (5) provides an upper bound of the absolute value of the inner product of any two vectors in a Hilbert space as a function of the respective lengths of these vectors. This inequality along with the rule of the parallelogram form the basis for the geometric insight in Hilbert spaces. As a sidenote, said inequality is tight with equality holding when $\mathbf{q}$ and $\mathbf{s}$ are colinear.

$$\left|\mathbf{q}^T\mathbf{s}\right| \;\leq\; \|\mathbf{q}\|_2 \, \|\mathbf{s}\|_2 \tag{5}$$

Similar relationshps to equation (5) or even more general inequalities involving norms in Hilbert or Banach spaces can be proven. For instance the Hölder inequality shown in equation (6) couples two norms that may be easier to compute in their respective spaces. Moreover, it is the starting point for proving the Minkowski inequality, namely the triangle inequality, in higher dimensions. In the context of vector databases such bounds help delineate cluster boundaries.

$$\|f\, g\|_1 \;\leq\; \|f\|_p \, \|g\|_q \quad \text{with} \quad \frac{1}{p} + \frac{1}{q} \;=\; 1 \tag{6}$$

Since the cosine always remains bounded between minus and plus one, its numerical properties are attractive. The latter can be seen from metrics evaluating numerical tractability. From a numerical perspective, the stability of any function $f(x)\colon \mathbb{R} \to \mathbb{R}$ can be evaluated with the condition number κ_2 defined as in equation (7) which evaluates not the computational complexity but the numerical difficulty. The latter can lead to unexpected results when left unchecked.

$$\kappa_2 \triangleq \left| \frac{x f^{(1)}(x)}{f(x)} \right| \tag{7}$$

In the case of the similarity metric of equation (3) the condition number is the one in equation (8). It indicates that numerically the computation is not everywhere stable. Therefore, care should be taken around these regions in order to ensure reliable results, perhaps by using alternative yet more stable forms. The latter may include power series, parametric forms, and iterative computations. The cosine function in particular can be computed through table interpolation, truncated series, and Monte Carlo simulations.

$$\kappa_2 \;=\; \left| \frac{\theta(-\sin\theta)}{\cos\theta} \right| \;=\; |\theta \tan\theta| \tag{8}$$

The p norm shown in equation (9) is another vector similarity metric offering flexibility through the vector $\mathbf{w}$ containing non-negative weights. Unlike equation (3), both operands must have the same scale for the results to be meaningful. The same norm in estimation theory is the natural estimator which is often a good starting point in terms of computational cost and accuracy for more refined iterative estimators such as those based on expectation-maximization (EM) strategies.

$$\mathrm{sim}\,(\mathbf{q}, \mathbf{s}; \mathbf{w}) \triangleq \left\|\mathbf{w}^T(\mathbf{q} - \mathbf{s})\right\|_p \;=\; \sqrt[p]{\sum_{i=1}^{n} \mathbf{w}[i]|\mathbf{q}[i] - \mathbf{s}[i]|^p} \tag{9}$$

The inverse weighted Euclidean distribution of equation (10) is also a popular vector similarity metric which has numerous applications in recommendation systems because of its easy interpretability. Moreover, it is at the core of the t-SNE clustering scheme. As a sidenote, this metric is also the probability density function of the Cauchy distribution which can be interpreted as the ratio of two Gaussian distributions. The positive hyperparameter γ_0 determines the relative weight of the second term of the denominator.

$$\mathrm{sim}\,(\mathbf{q}, \mathbf{s}; \gamma_0) \;=\; \frac{1}{1 + \gamma_0 \|\mathbf{q} - \mathbf{s}\|_2} \tag{10}$$

The above equations can be efficiently implemented in dedicated linear algebra software such as basic linear algebra subprograms (BLAS) and they can even be executed directly on GPUs for additional performance or they can be accelerated with single instruction on multiple data (SIMD) processors.

Observe that equation (3), equation (10), and equation (9) when p equals two are invariant with respect to orthogonal transforms represented by an orthogonal matrix $\mathbf{Q}$. The key is equation (11) where In the context of document retrieval orthogonal transforms, which consist exclusively of rotations and reflections, can be used to express alternative bases of the same topic or cross-domain knowledge transfer operations.

$$(\mathbf{Q}\mathbf{q})^T(\mathbf{Q}\mathbf{s}) \;=\; \mathbf{q}^T \underbrace{\mathbf{Q}^T\mathbf{Q}}_{\mathbf{I}} \mathbf{s} \;=\; \mathbf{q}^T\mathbf{s} \tag{11}$$

As a general remark topological properties play an important role in mappings between any two spaces or developing efficient descriptions of an attribute space such as the one representing a document collection. Moreover, the connection between IR and linear algebra is close with perhaps the most typical example being latent semantic indexing (LSI) which in its core is the singular value decomposition (SVD). Moreover, the basic form of PageRank is an eigenvalue problem and many implementations rely heavily on iterative methods for finding the primary eigenvector of a stochastic matrix representing inbound and outbound links between Web pages, essentially simulating a random walker and seeking the stationary distribution of the associated Markov chain.

B. Graph patterns

As stated earlier the documents of the collection D are represented as vectors under the model supported by the databases under consideration. In the context of this work their entries come from the graph representation of a document and they are shown in table II. The rationale for this decision is that many contemporary document formats such as .docx or .odt consist of balanced XML tags which can be nested. Moreover, natural language models (NLP) such as word2vec

represent documents at paragraph, sentence, or even word level as graphs with points of interest as vertices and syntactic and potentially semantic coherence denoted by edges.

TABLE II
GRAPH PATTERNS.

Pattern	Pattern
Number of vertices	Number of edges
Density	Completeness
Has one edge	Has two edges
Local diameter	Local 80% diameter
Local 70% diameter	Local 90% diameter
Number of triangles	Clustering coefficient I
Number of squares	Clustering coefficient II
Number of cliques of size four	Adamic-Adar index
Minimum degree	Allocation index
Average degree	Newman-Girvan score
Maximum degree	Attachment metric
Eigenvalue centrality	PageRank score

Once the attributes of each vertex have been computed, their values have been normalized with respect to the corresponding maximum value of each vector. All local properties have been computed over a neighborhood of size four. The eigenvalue centrality and the PageRank score were computed for the entire graph as they are global properties.

The final document vector results from the vectors of its vertices by summing them. Even though a normalization process is necessary at the attribute level in order to signify the relative importance of a given feature in the document, this is not necessary for entire documents since their respective lengths are normalized in the cosine similarity metric.

$$\mathbf{d}_i = \sum_{j \in d_i} \mathbf{d}_{i,j} \tag{12}$$

Nevertheless, if normalization is necessary at the document level, then schemes like the tf-idf can be used in order to evaluate the weight of both terms, or attributes in this case, and documents in the collection.

C. Functional mining

Functional programming is a paradigm with close ties to declarative programming and most importantly feels different from the imperative and object oriented ones as heavy emphasis is placed among others on immutability and function composition. As a result, stateful variables, especially loop counters, are avoided. On the contrary, lists, graphs, and strings are preferred as they lend themsleves to list representation in a straightforward manner. Python supports functional programming through the built-in *map* method as well as through the module *functools* which provides decorators, the built-in *reduce*, and partially implemented functions which allow the implementations of higher order functions. For instance, the entropy of a list or tuple representing a probability distribution in Python can be written as follows.

```
import functools as f
import numpy as np
vals = d.values()
H = f.reduce(lambda H,p: \
    H-p*np.log2(p) if p else H,\
    vals, 0)
```

In table III the functionality of the *functools* module and certain Python concepts necessary to build functional programs at the top and bottom part of it respectiely.

TABLE III
MODULE FUNCTOOLS FUNCTIONALITY AND PYTHON CONCEPTS.

Name	Description
map	Application of a function to iterables
filter	Selection of data from iterables
reduce	Yields a result from a list of iterables
partial	Implementation of partial objects
@lru_cache	Result cache decorator
@totalordering	Simplifies comparison operators
yield	Obtain the next result from a generator
lambda	Create anonynous functions
callable	Objects which can be called as a function
iterable	Objects which can be iterated

Within the context of this computing paradigm emphasis is placed on list and tuple processing and transformation, list and dictionary comprehensions, recursive processing which is suitable for processing documents and graphs as both data types have recursive structure abounding with nested patterns, function compositions, and object immutuability. Thus, stateful programming constructs especially loops are to be avoided.

The **yield** statement is useful in iterables like the keys of a dictionary or the values of a tuple to construct code segments which give a value and suspend their functionality until a new value is needed. For instance, the following code counts the vertices of an undirected graph with a degree $\deg(v_k)$ equal to one using a generator so that one degree at a time is read.

```
import functools as f

def row_gen(adj):
    for row in range(adj.shape[0]):
        yield adj[row]

y = f.reduce(lambda x,r: \
    x+1 if (2 == sum(r)) \
    else x, row_gen)
```

In the above code degrees can also be stored in a tuple as in equation (13) in order to ensure immutuability inside the generator body as well as code readability.

$$(t_1, \ldots, t_m) \tag{13}$$

In the above code segment the **lambda** expression acts as a one line method and in general it can be a lightweight constructor or a short method saving thus time to the function stack. Along a same line of reasoning, decorators, which are also lightweight functions, can alter computation by changing method call sequences and speed up computations by caching results. In fact, the latter is a very common use of decorators outside the functional paradigm.

The last functional tool used in the creation of partially implemented methods and objects through the *partial* method in order to prevent computational side effects. For instance, the following code snippet creates a highly parameterized method which sends its input to the desired stream with flushing. Recall that in order for this to work, the target function must support a keyword-pair structure of arguments.

```
import functools as f

ffprint = f.partial(print,\
    file=sys.stderr, flush=True)
```

IV. Results

A. Synthetic dataset

In order to generate documents with graph based structure, a synthetic benchmark dataset generation using the Kronecker model has been created. At the heart of the Kronecker model lies equation (14). This model has many appealing properties with the main one being the generation of graphs with power laws in the degree distribution, eigenvalue distribution, and in the primary eigenvector components among others.

$$\mathbf{A}^{[k]} \triangleq \begin{cases} \mathbf{I}_n & k = 0 \\ \mathbf{A} & k = 1 \\ \mathbf{A}^{[k-1]} \otimes \mathbf{A} & k \geq 2 \end{cases} \tag{14}$$

The generator matrices for the above model had seven and eight vertices respectively. Since the Kronecker model generates a sequence of graphs of exponential growth, two large template documents were obtained, one for each generator graph. These served as the template for the experiments by creating two collections of a thousand documents each by randomly sampling their edges with the sampling probability p_0 used as a parameter to obtain multiple collections.

B. Geometric properties

The power iteration clustering (PIC) method was used to cluster the documents with each collection. PIC essentially simulates the choices of a random walker on a Markov chain where edge crossing probabilities are a function of the similarity metric. After a sufficiently high number of steps and under mild ergodicity assumptions the steady state emerges, allowing a clustering of the states to emerge naturally and progressively with each step of the random walker.

The average angle between each document in the collections has been computed using equation (3). The results are seen in table IV. The values of p_0 ensure dense documents with sufficient structural variability. From its values it can be inferred that on avarage documents are not colinear of close to that and as such they can be clustered relatively easily. Moreover, the CSB inequality can be used to delineate cluster boundaries to acceleare indexing and retrieval.

Distance value distribution between the documents has been modeled as a power law as in equation (3). For each case the exponent γ_0, which critically determines the shape of the power law, can been fitted with least squares (LS) with the results being shown in table V. When the exponent is above two, then the mean of the power law is finite, implying a more balanced distribution of distances.

TABLE IV
Average Angle (Degrees)

p_0	0.5	0.6	0.7	0.8
Collection I	24.6667	28.3614	33.5097	42.5613
Collection II	22.3618	25.6619	30.6718	37.4545

$$y[k] \triangleq \alpha_0\, k^{-\gamma_0} \;=\; \alpha_0\, e^{-\gamma_0 \ln k} \tag{15}$$

TABLE V
Power Law Exponent (LS Fit)

p_0	0.5	0.6	0.7	0.8
Collection I	1.8999	2.0117	2.1343	2.2215
Collection II	1.75198	2.0092	2.1071	2.1566

V. Future Work

This conference paper focuses on presenting an algorithmic framework for processing documents structured as nested graphs with balanced tags stored in a vector database. The vectors representing each such document were mined using functional Python code using the programming tools from the *functools* module. The functional paradigm draws its power from function composition, avoiding stateful constructs such as loops, and extensive list processing with series of filters. These traits make it ideal for graphs since filters can efficiently discover local patterns like triangles and the Adamic-Acar index. The results obtained from clustering the documents of the becnchmark dataset based on attributes derived from the graph representation using the cosine similarity metric and the power iteration clustering (PIC) scheme reveal a broad spectrum of geometric properties ranging from the power law describing cluster distances to the numerical difficulty of evaluating the angle for a given document pair. These can not only yield important insight into any document collection, but they can also model the trajectories of query sequences over time and over users allowing thus the development of personalized user profiles as well as the optimization of storage arrangement and indexing of the underlying vector database.

The proposed methodology can be extended in a number of ways. First and foremost more synthetic and real benchmark datasets can be used to assess the recommendation performance. Moreover, graphs can be extended with attributes in their vertices and edges in order to represent nested tags.

Acknowledgment

This conference paper is part of Project 451, a long term research initiative with a primary objective of developing novel, scalable, numerically stable, and interpretable higher order analytics.

REFERENCES

[1] T. Taipalus and J. Lu, "Vector representations of multi-modal data," in *European Conference on Advances in Databases and Information Systems*. Springer, 2025, pp. 272–279.

[2] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," *IEEE Transactions on Big Data*, 2025.

[3] J. J. Pan, J. Wang, and G. Li, "Vector database management techniques and systems," in *Companion of the 2024 International Conference on Management of Data*, 2024, pp. 597–604.

[4] T. Taipalus, "Vector database management systems: Fundamental concepts, use-cases, and current challenges," *Cognitive Systems Research*, vol. 85, 2024.

[5] J. S. Hammer, K. Theodoropoulos, G. Drakopoulos, G. Bardis, and P. Mylonas, "Twitter recommendations across dialects: Seeing the big board through Faiss," in *SMAP*. IEEE, 2025.

[6] B. Sarmah, D. Mehta, B. Hall, R. Rao, S. Patel, and S. Pasquali, "Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction," in *International Conference on AI in Finance*. ACM, 2024, pp. 608–616.

[7] A. T. Neumann, Y. Yin, S. Sowe, S. Decker, and M. Jarke, "An LLM-driven chatbot in higher education for databases and information systems," *IEEE Transactions on Education*, 2024.

[8] N. Matsumoto, J. Moran, H. Choi, M. E. Hernandez, M. Venkatesan, P. Wang, and J. H. Moore, "KRAGEN: A knowledge graph-enhanced RAG framework for biomedical problem solving using large language models," *Bioinformatics*, vol. 40, no. 6, 2024.

[9] C. Jin, Z. Zhang, X. Jiang, F. Liu, S. Liu, X. Liu, and X. Jin, "Ragcache: Efficient knowledge caching for retrieval-augmented generation," *ACM Transactions on Computer Systems*, 2024.

[10] G. Drakopoulos and P. Mylonas, "Clustering MBTI personalities with graph filters and self organizing maps over Pinecone," in *Big Data*. IEEE, 2024.

[11] Y. Zhang, S. Liu, and J. Wang, "Are there fundamental limitations in supporting vector data management in relational databases? A case study of PostgreSQL," in *ICDE*. IEEE, 2024, pp. 3640–3653.

[12] J. Gao, Y. Gou, Y. Xu, Y. Yang, C. Long, and R. C.-W. Wong, "Practical and asymptotically optimal quantization of high-dimensional vectors in euclidean space for approximate nearest neighbor search," *Proceedings of the ACM on Management of Data*, vol. 3, no. 3, pp. 1–26, 2025.

[13] M. Wang, W. Xu, X. Yi, S. Wu, Z. Peng, X. Ke, Y. Gao, X. Xu, R. Guo, and C. Xie, "Starling: An I/O-efficient disk-resident graph index framework for high-dimensional vector similarity search on data segment," *ACM on Management of Data*, vol. 2, no. 1, pp. 1–27, 2024.

[14] J. Xian, T. Teofili, R. Pradeep, and J. Lin, "Vector search with OpenAI embeddings: Lucene is all you need," in *International Conference on Web Search and Data Mining*. ACM, 2024.

[15] J. J. Pan, J. Wang, and G. Li, "Survey of vector database management systems," *The VLDB Journal*, vol. 33, no. 5, pp. 1591–1615, 2024.

[16] S. Bruch, *Foundations of Vector Retrieval*. Springer, 2024, vol. 1.

[17] G. Drakopoulos and P. Mylonas, "A genetic algorithm for Boolean semiring matrix factorization with applications to graph mining," in *Big Data*. IEEE, 2022.

[18] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[19] L. B. Nguyen, I. Zelinka, V. Snasel, L. T. Nguyen, and B. Vo, "Subgraph mining in a large graph: A review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2022.

[20] Z. Sun, B. Wu, Y. Wang, and Y. Ye, "Sequential graph collaborative filtering," *Information Sciences*, vol. 592, pp. 244–260, 2022.

[21] E. Ceci and S. Barbarossa, "Graph signal processing in the presence of topology uncertainties," *IEEE Transactions on Signal Processing*, vol. 68, pp. 1558–1573, 2020.

[22] L. Wu, P. Cui, J. Pei, L. Zhao, and L. Song, "Graph neural networks," in *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer, 2022, pp. 27–37.

[23] G. Drakopoulos, X. Liapakis, G. Tzimas, and P. Mylonas, "A graph resilience metric based on paths: Higher order analytics with GPU," in *ICTAI*. IEEE, 2018, pp. 884–891.

[24] G. Drakopoulos, K. C. Giotopoulos, I. Giannoukou, and S. Sioutas, "Unsupervised discovery of semantically aware communities with tensor Kruskal decomposition: A case study in Twitter," in *SMAP*. IEEE, 2020.

[25] J. Liu, S. Zhang, and H. Fan, "A two-stage hybrid credit risk prediction model based on XGBoost and graph-based deep neural network," *Expert Systems with Applications*, vol. 195, 2022.

[26] T. E. Trueman, P. Narayanasamy, and J. Ashok Kumar, "A graph-based method for ranking of cloud service providers," *The Journal of Supercomputing*, vol. 78, no. 5, pp. 7260–7277, 2022.

[27] G. Drakopoulos, I. Giannoukou, S. Sioutas, and P. Mylonas, "Self organizing maps for cultural content delivery," *NCAA*, vol. 31, no. 7, 2022.

[28] F. Zhong, X. Wu, R. Yang, X. Li, D. Wang, Z. Fu, X. Liu, X. Wan, T. Yang, Z. Fan *et al.*, "Drug target inference by mining transcriptional data using a novel graph convolutional network framework," *Protein & cell*, vol. 13, no. 4, pp. 281–301, 2022.

[29] Z. Abbas, V. Kalavri, P. Carbone, and V. Vlassov, "Streaming graph partitioning: An experimental study," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1590–1603, 2018.

[30] G. Drakopoulos, Y. Voutos, and P. Mylonas, "Annotation-assisted clustering of player profiles in cultural games: A case for tensor analytics in Julia," *BDCC*, vol. 4, no. 4, 2020.

[31] D. Pollak, V. Layka, and A. Sacco, "Functional programming," in *Beginning Scala 3*. Springer, 2022, pp. 79–109.

[32] P. Hudak, "Conception, evolution, and application of functional programming languages," *ACM CSUR*, vol. 21, no. 3, pp. 359–411, 1989.

[33] M. Hartmann and R. Shevchenko, *Professional Scala: Combine object-oriented and functional programming to build high-performance applications*. Packt Publishing Ltd, 2018.

[34] S. Chellappan and D. Ganesan, "Scala: Functional programming aspects," in *Practical Apache Spark*. Springer, 2018, pp. 1–37.

[35] R. Dyer and J. Chauhan, "An exploratory study on the predominant programming paradigms in Python code," in *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2022, pp. 684–695.

[36] S. F. Lott, *Functional Python programming: Discover the power of functional programming, generator functions, lazy evaluation, the built-in itertools library, and monads*. Packt Publishing Ltd, 2018.

[37] C. Okasaki, *Purely functional data structures*. Cambridge University Press, 1999.