Towards AI-Powered Multi-Modal Generative OLAP

Sandro Bimonte[1], Chant Boyadjian[1,*], Stefano Rizzi[2] and Sana Sellami[3]

[1]TSCF, INRAE - University of Clermont Auvergne, Aubière, France
[2]DISI, University of Bologna, Italy
[3]LIS, Aix Marseille University, France

Abstract

Recent advances in AI allow decision makers to move beyond traditional analysis towards sophisticated decision-making tasks that require human intuition and perception. This opens the door to a novel form of OLAP, one that integrates unstructured data —such as text and images— into its analytical workflows. In this work we propose OLAP-AI, a novel and enriched form of the OLAP paradigm aimed at supporting, besides the analysis of categorical and numeric data, also semantically rich operations over free text and images. OLAP-AI is multi-modal, in that it supports crossed semantic searches between text and images for filtering and grouping. Besides, it significantly extends aggregation by operating on text and images rather than on numeric data only, and by relying on generative models. To investigate the technical feasibility of the OLAP-AI paradigm, we provide a proof-of-concept by relying on the open-source vector DBMS Weaviate.

Keywords

AI, OLAP, Multi-modal databases, Vector databases

1. Introduction and motivation

In recent years, Artificial Intelligence (AI) has provided an important transformation, achieving unprecedented capabilities in the understanding and generation of natural language text and images. These advancements allow decision makers to move beyond traditional analysis towards sophisticated decision-making tasks that require human intuition and perception [1]. Therefore, AI represents an important opportunity to rethink and enhance the OnLine Analytical Processing (OLAP) paradigm, which is still one of the main Business Intelligence tools for analyzing huge volumes of numeric data stored in multidimensional cubes by means of simple statistical indicators [2].

Traditionally, OLAP systems have been focused on the manipulation of structured data, primarily numeric measures and categorical attributes. Some academic work investigated the use of textual data in OLAP operators to provide aggregation functions over textual documents. For instance, an OLAP-oriented textual aggregation method that identifies the most representative keywords from multiple documents is proposed in [3]. Statistical importance is then combined with semantic similarity to aggregate keywords. In [4], a contextualized warehouse is proposed, where facts from the structured warehouse are linked to their relevant contextual documents allowing users to analyze context by providing a sequence of keywords. In [5], the XML-OLAP framework introduces a multidimensional analysis approach designed for XML data warehouses. A new multidimensional expression language called XML-MDX is proposed to enable textual aggregation in the form of summarization, classification, and top keyword extraction over XML cubes. Text Cube [6] is a data cube with a term hierarchy to analyze multi-dimensional text data. It supports two measures, term frequency vector (TF) and inverted index (IV), which facilitate the application of information retrieval techniques. The cube also supports traditional OLAP operations. TextRank [7] is an unsupervised graph-based ranking model for natural language processing. The model supports two main applications: keyword extraction, useful for text

ER2025: Companion Proceedings of the 44th International Conference on Conceptual Modeling: Industrial Track, ER Forum, 8th SCME, Doctoral Consortium, Tutorials, Project Exhibitions, Posters and Demos, October 20-23, 2025, Poitiers, France
*Corresponding author.
sandro.bimonte@inrae.fr (S. Bimonte); chant.boyadjian@inrae.fr (C. Boyadjian); stefano.rizzi@unibo.it (S. Rizzi); sana.sellami@lis-lab.fr (S. Sellami)
0000-0003-1727-6954 (S. Bimonte); 0000-0002-4617-217X (S. Rizzi)

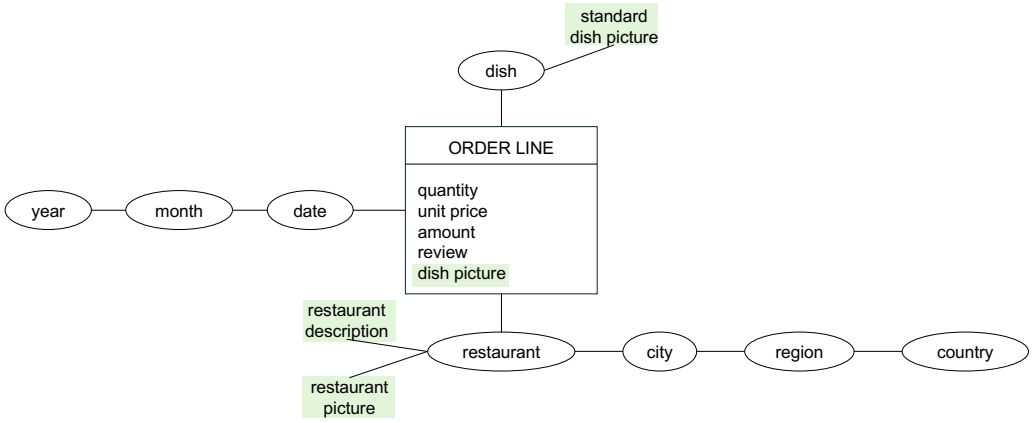

Figure 1: The conceptual schema for the ORDER LINE cube shown in the DFM [2] notation: ovals are dimensions and hierarchy levels; descriptive attributes (such as restaurant description) are attached to the level they describe; measures are listed within the box. The OLAP-AI extensions are highlighted in green.

classification, and sentence extraction for automatic summarization. It ranks each word or sentence based on its importance to the entire text. Similarly, a few works have studied the use of images in OLAP analyses. For instance, iCube [8] extends a traditional data warehouse by including a dimension table that stores intrinsic image features as vectors, which are used with similarity for filtering queries. Overall, these works show an interest in using non-structured data for OLAP analyses to enhance the decision-making process. However, they do not support a tight integration of multi-modal data within the same conceptual and technological framework, nor do they extend OLAP operators to cope with text, images, and numeric data in a seamless way. Moreover, the aggregation functions provided by these works are basic data summarization methods based on statistical and data mining approaches.

We claim that the advent of AI models opens the door to a novel form of OLAP, one that integrates unstructured data —such as text and images— into its analytical workflows and enables generative multi-modal selection, grouping, and aggregation of these data. Thus, in this work we propose a novel vision of the OLAP paradigm, which we call *OLAP-AI* and aims at extending the expressive power of OLAP to support, besides the analysis of categorical and numeric data, also semantically rich operations over free text and images. Our approach can be labeled as *multi-modal*, in that it supports crossed semantic searches between text and images for filtering and grouping [9]. Remarkably, it extends aggregation in two ways: (i) with reference to classical OLAP, by operating on text and images rather than on numeric data only; (ii) with reference to the approaches to text/image-based OLAP mentioned above, by relying on generative models. From a technical point of view, the implementation of this paradigm shift requires the integration of multi-modal and generative AI models into DBMSs (Database Management Systems). One promising direction to achieve this integration would be by relying on the emerging technology of *vector databases*, which provide a practical foundation for indexing, searching, and aggregating high-dimensional representations of diverse data types [10]. To investigate the feasibility of this solution, we provide a proof-of-concept for OLAP-AI by relying on the vector DBMS Weaviate.

The paper is structured as follows. Section 2 introduces the OLAP-AI paradigm, while Section 3 describes our proof-of-concept implementation. Section 4 closes the paper by discussing the main open challenges for research.

2. Multi-modal generative OLAP

This section describes the OLAP-AI paradigm we envision for multidimensional analysis of structured and unstructured data. We illustrate OLAP-AI with a multidimensional cube that models all orders placed across a chain's restaurants; its conceptual schema is shown in Figure 1.

OLAP analyses are commonly based on the multidimensional model [2], whose main citizens are

Sweet, but the price felt slightly high for what it was

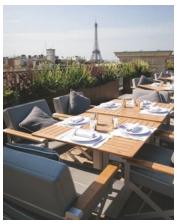

Figure 2: An example of review (left) and one of restaurant picture (right)

facts to be analyzed by means of *dimensions*. Dimensions are organized in *hierarchies* with nested *levels*, and facts are described by numeric *measures* to be aggregated by means of basic aggregation functions (e.g., SUM, AVG, MIN). Instances of levels are categorical textual data and are called *members*. The conceptual schema for our ORDER LINE fact features four dimensions:

- date, i.e., the order data with levels day, month, and year.
- customer, i.e., the account of the customer who placed each order.
- dish, whose members correspond to all the dishes offered by the restaurant chain. Note that, for each dish offered by multiple restaurants, there will be a single member of dish.
- restaurant, whose members are the chain's various restaurants with their geographical locations and descriptions.

Each order line is described by some numeric measures: quantity, unit price, and amount (computed as quantity $\times$ unit price). This schema supports classical OLAP queries such as "What is the total and average amount of the orders placed each year by each customer in Paris?"

In OLAP-AI, this multidimensional schema could be enriched with text and images as shown (in green) in Figure 1. For instance:

- each dish could be associated with a standard picture of that dish;
- each restaurant could have a textual description, and a picture showing whether it is outdoor or indoor (see Figure 2, right);
- each order line could have two additional non-numeric measures: the picture of the dish actually served to the customer and a textual review given by the customer (see Figure 2, left).

To illustrate how OLAP-AI leverages multimodal data and generative AI models, we showcase a few representative queries enabled by the enriched schema:

- **Q1**. *For each city and year, only for the orders of burgers placed in restaurants with a terrace, show the average amount paid, the average unit price, and the dish picture that is most similar to the standard burger picture offered by the chain.* Here, the filter selects the restaurants with a terrace by means of a multi-modal semantic search (search stored images based on input text) on the restaurant photos. Besides, an aggregation operator is applied to the dish picture measure to return the burger picture most similar to the standard one (search stored images based on input image).
- **Q2**. *For each type of restaurant (indoor or outdoor), show a representative image of the burgers ordered.* To classify restaurants as indoor or outdoor, their pictures are analyzed via multi-modal semantic search (search stored images based on input text). The aggregation operator returns, for each group of pictures, a single picture that emphasizes the common visual characteristics of the burgers ordered, computed via a generative model.
- **Q3.** *For each city, year, and restaurant category (fast food or traditional), return a summary of the customer reviews and the average amount paid, but only for restaurants considered expensive.* Restaurants are considered to be expensive if at least one of their reviews mentions a high price, while the restaurant category is inferred from the restaurant descriptive text. Both tasks are achieved via a semantic search (search stored text based on input text). Aggregation involves

Query	Filtering	Grouping	Aggregation
Q1	*multi-modal* (search restaurant pictures for a terrace)	categorical (city, year)	numeric (average amount and unit price); images (find most similar burger picture)
Q2	–	*multi-modal* (classify restaurant pictures as indoor or outdoor)	*generative* over images (generate representative picture)
Q3	text (classify review text as expensive or not)	categorical (city, year); text (classify restaurant descriptions as fast food or traditional)	numeric (average amount); *generative* over text (generate summary of reviews)

Table 1
Examples of OLAP-AI queries on the ORDER LINE cube

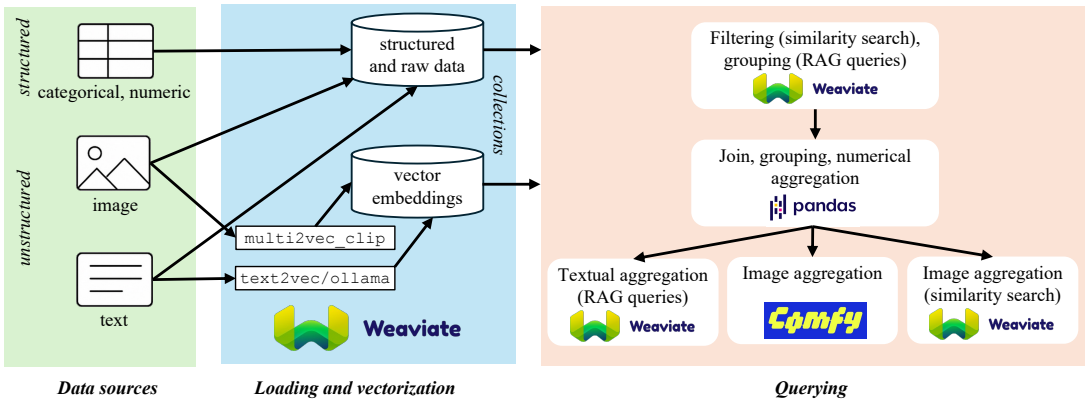

Figure 3: Data pipeline for OLAP-AI

both numeric data (the average amount) and text (the summary of customer reviews obtained via a generative model).

Table 1 summarizes the use of images and text in each query for filtering (WHERE clause), grouping (GROUP BY clause), and aggregating (SELECT clause). Note that, in the table, the terms 'categorical' and 'numeric' refer to the standard way of specifying filtering, grouping, and aggregation in OLAP. Noticeably, semantic search is applied over multi-modal data to filter and aggregate multidimensional data based on information (such as the presence of a terrace) that is not explicitly stored in the cube in categorical form. Moreover, generative AI models enable the extension of classical (numeric) aggregation to operate on images (e.g., to create a representative image) and text (e.g., by creating review summaries).

3. Proof-of-concept

In this section we give a proof-of-concept for the OLAP-AI paradigm by describing a prototype implementation of the enriched ORDER LINE cube and of the queries described in the previous section. The overall data pipeline is sketched in Figure 3.

Data is stored using Weaviate (https://weaviate.io), an open-source vector database designed for storing and searching data via machine learning-generated vector embeddings. Weaviate supports efficient semantic search, i.e., users can retrieve information based on the meaning of queries rather than on exact keyword matches. With its built-in support for various AI models, Weaviate is especially useful for applications involving natural language processing, image search, and recommendations. Its scalability and flexibility make it a powerful tool for developers building AI-powered search and data analysis solutions.

To implement multi-modal filters, we used *CLIP* (Contrastive Language-Image Pre-training) [11], a deep search model developed by OpenAI that learns to associate images with their corresponding textual descriptions by relying on a dataset of 400 million image-text pairs collected from the internet. To

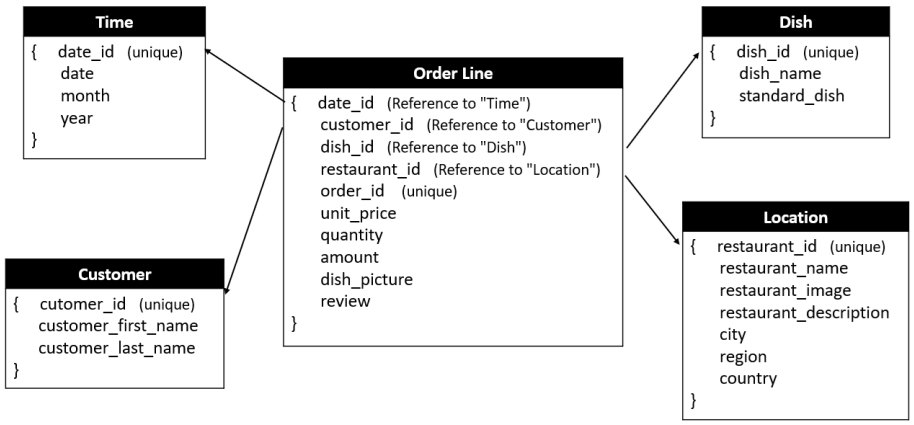

Figure 4: The shattered schema of the ORDER LINE cube; the Order Line collection contains a reference to each dimension's unique identifier

compute text and image similarities, we used the `near_media` and `near_text` functions of Weaviate. For generative textual aggregation, we used the `generate.near_text` function, which combines semantic text search with a text generation step using an LLM. Finally, since the generation of images is not supported by Weaviate yet, for this task we used an external application called *ComfyUI*.

To store the cube data, we have adopted a *shattered schema* [12] and created one document collection per dimension plus one for the fact, as shown in Figure 4. Weaviate does not natively support joins, so we used the Pandas library to manually implement them. Images and textual descriptions were transformed into embeddings using Weaviate functions.

Overall, to design and feed the enriched LINE ORDER cube with Weaviate, we followed two steps:

1. *Multidimensional design.* Here, we created the collections of dimensions and facts according to the shattered schema and configured the vectorization task. In Weaviate, vectorization is configured during the initialization of a collection using the `vectorizer_config` parameter, which accepts an array of vectorizer configurations. The configuration parameters depend on the chosen module.

2. *Data loading and vectorization.* We fed the shattered schema with both structured alphanumeric data and unstructured data in their native forms together with their corresponding vectors. A vector is a numeric representation that captures the semantic content of one or more data fields. These vectors are produced by machine learning algorithms called *vectorizers*, which transform unstructured data into embeddings. In Weaviate, unstructured data are stored both in their original form and as vectors. Similarity searches are performed on these vectors, enabling comparison and retrieval based on semantic closeness rather than exact matches. This allows, for example, the similarity between two texts, two images, or even between an image and a text to be computed. This feature is supported by the multi-modal approach of Weaviate, where vector representations of different data modalities are represented in the same vector space. During data ingestion, vector embeddings are automatically created and stored alongside the raw data in the database. For our proof-of-concept, we have configured two types of vectorizers: (**i**) The `text2vec_ollama` module, which relies on the `nomic-embed-text` model to generate vectors from the textual fields restaurant_description of Location and review of Order Line; (**ii**) The `multi2vec_clip` module, which applies the CLIP model to the image fields restaurant_image of Location and dish_picture of Dish.

To implement the OLAP-AI queries introduced in Section 2 we followed four sequential steps:

1. *Filtering*, aimed at filtering the data based on conditions applied to hierarchy levels using native Weaviate operators. This step takes a collection as input and returns only the elements that satisfy the conditions.

2. *Join*, which merges the collections data of dimensions and fact required to execute the query. Joins are implemented using the Pandas library's `merge` function. The input is a set of DataFrames constructed by extracting the properties of objects retrieved from Weaviate collections and by converting Python dictionaries into lists of tuples. The output is a single DataFrame that combines all the data.

3. *Grouping and numeric aggregation*, where the data is grouped by specified hierarchy levels and numeric aggregation functions —such as average or sum— are applied to selected measures. The implementation uses the `groupby` method of Pandas, which takes the merged DataFrame as input and produces a new DataFrame containing grouped and aggregated results.

4. *Multi-model aggregation.* Different methods and tools are used to aggregate data according to their type. For generative textual aggregation (as in Q3) we use RAG queries, while to find the most similar image to a given one (as in Q1) we use the semantic search provided by Weaviate. Generative image aggregation (as in Q2) is handled with ComfyUI, since Weaviate does not support the integration of generative models for images.

To illustrate the query implementation, in the remainder of this section we explain how Q3 has been implemented using the Python client of Weaviate. Noticeably, this query requires the use of the prompting methods offered by Weaviate to generate information that is not explicitly stored in the cube, so as to enable textual grouping and aggregation. Weaviate provides a specific kind of query for prompting, called *RAG* (Retrieval Augmented Generation) query. During initialization, the configuration of a generative model can be defined to enable RAG queries that accept prompts. The configuration includes specifying an API endpoint and the model name, which in our case is `llama3:8b`. A RAG query in Weaviate first performs a similarity search to retrieve relevant objects, then applies a prompt on the search results using a generative AI model. Weaviate supports two main types of RAG queries: (i) single prompt, which generates one response per returned object, and (ii) grouped task, which generates a single response for the entire set of returned objects. Here are the steps we followed:

1. Firstly, we implemented the WHERE clause (filtering step). In order to identify restaurants considered as expensive, we performed a text semantic search using the `near_text` operator to return the orders whose dish review embedding in the `Order Line` collection has a small distance from that of the text "high price".

2. Query Q3 asks to group order lines by restaurant category (fast food or classic). To this end, we wrote a single prompt RAG query over the Location collection; in this query, we asked via a prompt (shown in Figure 5) to classify restaurants into "fast food" and "classic" by applying the `near_text` operator to the embeddings of textual restaurant descriptions. The result consists of a collection of objects, each containing the generated classification result and the corresponding restaurant's metadata. Then, we mapped each restaurant_id to its corresponding class and stored them in a dictionary called `category_map`.

3. Once data has been classified for grouping, three successive left joins must be executed. Firstly, we joined the restaurants considered expensive with their orders through the restaurant identifiers. Then, we joined each order in the result with its date, month, and year, using the relevant order identifiers. Finally, we joined the generated restaurant category (either fast food or classic) with the result of the second join, using the restaurant identifiers.

4. At this stage, grouping and numeric aggregation could be implemented. We grouped order lines by city, year, and restaurant category and computed the average amount for each group.

5. Finally, Q3 requires textual aggregation, which was implemented by generating a summary of the dish reviews through a grouped task RAG query with the `near_text` operator (the prompt is shown in Figure 5).

Classify the following restaurant description {restaurant_description} into one of these two types only: Fast Food or Classic. Return only the classified type.
Provide a short summary of the essential comments, using your own words, starting immediately with the summary and without adding any prefix or information not present in the reviews.

Figure 5: Prompts for classifying restaurants (top) and summarizing reviews (bottom)

```
Elapsed time: 58.69 seconds
       city  year restaurant_type     amount                                    review_summary
0  Barcelona  2024         Classic  42.333333  The food is excellent, with exceptional spice ...
1  Barcelona  2025       Fast Food  13.750000  The dishes had good visual appeal, but some fo...
2      Paris  2024         Classic   5.900000  The dish was unremarkable but pricey, while th...
3       Rome  2025         Classic   6.000000  Overall, customers appreciated the great taste...

Full summaries:

0:  The food is excellent, with exceptional spice blends, but the prices are high. The atmosphere is also noteworthy.

1:  The dishes had good visual appeal, but some found them overpriced.

2:  The dish was unremarkable but pricey, while the atmosphere was pleasant.

3:  Overall, customers appreciated the great taste and friendly service. Some even praised the expert seasoning that
elevated the dish's flavors, while others noted that the price point was a bit high.
```

Figure 6: The results of query Q3

One point deserves some further considerations. The `near_text` operator, which in Q3 computes the distance between the embedding of each review and the embedding of text "high price", ranks the reviews from the nearest to the farthest one. If no threshold was provided, this ranking would also include reviews that actually do not mention a high price. Thus, to ensure that the retrieved reviews are truly relevant, we specified a maximum distance (0.55) using the `distance` parameter. This value could not be automatically determined; we had to select an appropriate threshold by observing the distances of reviews known to mention a high price. The results of Q3 are shown in Figure 6. All restaurants are represented except the one with identifier 1, as it has no review mentioning a high price. The restaurants with identifiers 5, 4, and 2 were classified as classic, while the one with identifier 3 was classified as fast food.

4. Conclusion and open challenges

Recent advances in AI offer a compelling research opportunity to reimagine OLAP beyond its traditional focus on structured (categorical and numeric) data. The integration of multi-modal and generative AI models into DBMSs supported by vector DBMS gave us a chance to envision OLAP-AI, a significant enrichment of the OLAP paradigm to make it capable of seamlessly handling semantically rich queries over text, images, and numeric data within a unified analytical framework. Our approach moves beyond basic summarization techniques, enabling more insightful and flexible decision-making processes. To validate it, we have provided a proof-of-concept implementation using Weaviate, a vector database that supports semantic indexing and retrieval of multi-modal data.

Several open research challenges remain:

- *Formalization.* To support our OLAP-AI paradigm, the multidimensional model must be extended and the OLAP operators must be formally redefined. The use of AI will allow new selection and grouping levels to be defined at query time (as done, for instance, in Q2 by introducing a restaurantType level on-the-fly); in the same way, it will allow the separation between measures and dimension, which is a foundation of the classical multidimensional model, to be overcome (e.g., by classifying order lines into popular and unpopular based on their review).
- *Design and implementation*: Traditional OLAP systems rely on structured, relational schemas with well-defined dimensions and measures. Adopting vector databases, which rely on high-dimensional embeddings, requires redefining how unstructured data is represented, stored, and queried. In this work we used a vector database since it natively supports the integration of AI models. However, some recent efforts are going on to extend relational databases to support

vectors and AI models; an example of this is *PgAI* (https://github.com/timescale/pgai), an open-source extension of PostgreSQL that integrates AI directly in the DBMS.

- *Performance and optimization*: While generally well-handled by vector-based architectures, scalability raises some challenges when dealing with large and heterogeneous datasets, particularly in maintaining consistent performance across diverse data modalities. The design and optimization of new indexing strategies for multimodal data —especially when combining textual, visual, and numeric information— is an active research area, together with the definition, maintenance, and update of materialized views in a vector database context.
- *Querying*: OLAP query languages (such as MDX or SQL-based variants) are not natively designed to express semantic or similarity-based queries. Extending these languages to support vector search operations, such as nearest-neighbor retrieval or semantic joins, requires careful abstraction and standardization.
- *Risks*: Using AI models to produce (possibly generative) outputs in analytical settings raises risks, such as hallucinations and bias amplification, whose impact on the decision-making process must be investigated.

Declaration on Generative AI

No Generative AI tools have been used during the preparation of this work.

Acknowledgment and Declaration

This work was partially supported by the French projects 22-PEAE-0007, ANR-16-IDEX-0001, and ANR-24-CHR4-0004-0. No Generative AI tools have been used during the preparation of this work.

References

[1] E. Eigner, T. Händler, Determinants of LLM-assisted decision-making, arXiv 2402.17385 (2024).
[2] M. Golfarelli, S. Rizzi, Data Warehouse design: Modern principles and Methodologies, McGraw-Hill, 2009.
[3] M. Bouakkaz, Y. Ouinten, S. Loudcher, OLAP textual aggregation approach using the Google similarity distance, International Journal of Business Intelligence and Data Mining 11 (2016) 31–48.
[4] J. M. Pérez-Martínez, R. Berlanga-Llavori, M. J. Aramburu-Cabo, T. B. Pedersen, Contextualizing data warehouses with documents, Decision Support Systems 45 (2008) 77–94.
[5] B. Park, H. Han, I. Song, XML-OLAP: A multidimensional analysis framework for XML warehouses, in: Proc. DaWaK, Copenhagen, Denmark, 2005, pp. 32–42.
[6] F. Zhu, J. Han, B. Ding, C. X. Lin, B. Zhao, Text cube: Computing IR measures for multidimensional text database analysis, in: Proc. ICDM, 2008, pp. 905–910.
[7] R. Mihalcea, P. Tarau, TextRank: Bringing order into text, in: Proc. EMNLP, Barcelona, Spain, 2004, pp. 404–411.
[8] L. P. Annibal, J. C. Felipe, C. D. Ciferri, R. R. Ciferri, iCube: A similarity-based data cube for medical images, in: Proc. CBMS, Perth, Australia, 2010, pp. 321–326.
[9] L. Mei, S. Mo, Z. Yang, C. Chen, A survey of multimodal retrieval-augmented generation, CoRR abs/2504.08748 (2025).
[10] J. J. Pan, J. Wang, G. Li, Survey of vector database management systems, The VLDB Journal 33 (2024) 1591–1615.
[11] A. Radford, et al., Learning transferable visual models from natural language supervision, in: Proc. ICML, 2021, pp. 8748–8763.
[12] M. Chevalier, M. E. Malki, A. Kopliku, O. Teste, R. Tournier, Document-oriented models for data warehouses, in: Proc. ICEIS, Rome, Italy, 2016, pp. 142–149.