Multi-Modal RAG Search in Vector-Relational Databases: an Experimental Analysis

Suchitra Roy and Suprio Ray

University of New Brunswick, Canada
{sroy10,sray}@unb.ca

Abstract. Efficiently processing the ever-growing volumes of unstructured data is a pressing need. Traditional relational databases are not well-suited for managing and querying unstructured data. Advances in Large Language Model (LLM) have ushered in a new paradigm that enables natural language processing tasks on unstructured text. Since LLMs are susceptible to 'hallucination', Retrieval Augmented Generation (RAG) guided by vector indexes has emerged as a key enabler for LLM oriented data analysis. Consequently, several specialized vector database systems have been proposed. However, maintaining different data systems for different data tasks is fraught with challenges. A unified vector-relational database can offer the best of both worlds and hence may be preferable over custom solutions. Although vector-relational databases, based on vector extensions, have been introduced recently, their functionalities and performance vary widely.
Recent innovations in LLMs technology have led to Vision-Language Models (VLMs) that can incorporate visual inputs. This has opened up new possibilities for analyzing multimedia data, besides text, through multi-modal retrieval. In this paper, we present a multi-faceted and systematic experimental evaluation of VLM supported multi-modal RAG search with two open-source vector-relational databases. For our experimental analysis, we utilize an AI-assisted personalized nutrition application that we have developed, as well as a synthetic dataset with different data volumes to evaluate the scalability. Our study reveals several interesting findings and a few issues with existing systems, and it points to some future research directions.

Keywords: Vector-relational database · vision-language model (VLM) · Retrieval Augmented Generation (RAG) · multi-modal search.

1 Introduction

With the rapidly rising volumes of unstructured data, including text, image, audio and video, efficient processing of these data has become imperative. In recent years, pre-trained Large Language Model (LLM) [42] emerged as a transformative technology that has enabled the analysis of unstructured data and perform many tasks, such as, clustering, classification and information retrieval. LLMs

like ChatGPT [3] and Llama [11] have become quite popular within a short period of time. LLMs are typically Deep Neural Network (DNN) models that are pre-trained on vast bodies of natural language text corpus. A key idea behind LLMs is to represent data as vectors using neural embedding techniques.

Although LLMs have demonstrated remarkable feats with Natural Language Processing (NLP) applications, they suffer from 'hallucination', a term used to refer to the fact that LLMs may often provide responses that are fabricated and factually incorrect. This is caused by LLMs' inability to include domain-specific knowledge that is not part of the pre-training corpus. To address this issue, Retrieval-Augmented Generation (RAG) was introduced [30]. It can incorporate domain knowledge by integrating a pretrained retriever model with an external knowledge base, which is typically indexed by techniques, such as vector indexing [10, 31].

Recently, the advent of Vision-Language Models (VLMs) has made multi-modal analysis of text and multimedia data practical. There are several VLM training paradigms [26]. Contrastive training is a common approach that uses pairs of positive and negative examples, whereas the masking strategy involves reconstructing the missing patches of an image given an unmasked text caption. VLMs can leverage open-source LLMs, like Llama, as pretrained backbones. Generative VLMs are trained to generate complete images or long texts. By connecting visual inputs with language inputs, VLMs can enable applications, such as AI-assisted Personalized Nutrition (AIPN). Several projects [32,38,40,43] leveraged LLMs for tasks such as ingredient detection and nutrition assessment. In this paper, we explore how to extend the language model based approaches for ingredient recognition and nutrition assessment by utilizing multi-modal RAG search on vector-relational databases.

While traditional relational database systems are designed to manage well-structured data, they are not as suitable for querying unstructured data. Recently, a few vector databases have been introduced, such as Pinecone [17] and Milvus [13]. They are custom engines designed to efficiently process similarity searches on high-dimensional vectors. Previously, a few papers [24, 36] focused on evaluating Approximate Nearest Neighbor (ANN) queries. Recently, Kang et al. [28] developed a benchmark to evaluate the performance of three vector databases, Milvus, Weaviate [22] and Qdrant [18]. In contrast, a vector-relational database extends a traditional relational database to support both SQL queries and vector search. PostgreSQL with pgvector [16] extension, is an example of such a system. A vector-relational database is attractive because it enables users to leverage existing transactional and analytical query processing capabilities of a relational database, along with semantic search capabilities within one database system. Even a few vector databases use a relational database as the backend. For instance, ChromaDB [4] uses SQLite as the underlying relational database. However, to our knowledge no previous studies evaluated vector-relational databases in the context of VLM-based multi-modal RAG search.

To address the aforementioned research gap, we evaluate multi-modal RAG search with two open-source vector-relational databases: PostgreSQL (with pgvec-

tor extension) and SQLite (with vector extension) [20]. To that end, we develop a full-fledged real-world application. Specifically, we implement a prototype AI-assisted personalized nutrition (AIPN) application, demonstrating multi-modal RAG search empowered by a VLM and guided by a vector-relational database. We also evaluate the scalability of vector relational databases with a synthetic dataset consisting of vector data tables of varying sizes. We conduct a systematic experimental evaluation of different aspects of VLMs and vector-relational databases, as mentioned next.

1. Aspect 1: Vision Language Models (VLM) with multi-modal embedding and prompt response
2. Aspect 2: Multi-modal vector search strategies
3. Aspect 3: Vector-relational databases
4. Aspect 4: Vector data indexing strategies

Our evaluation reveals several interesting findings, which are summarized in the Conclusion section. These findings can provide some guidelines to VLM+RAG application developers, as well as researchers in the community.

The remainder of this paper is organized as follows. Section 2 outlines a motivating use-case and describes an application that we have developed based on this. In Section 3 we discuss our methodology, and details of the different aspects that we have analyzed. Then we present the experimental settings, datasets and evaluation results in Section 4. Finally, we conclude the paper in Section 5.

2 Motivating use-case

In this section, we describe a motivating use-case and prototype system that we have implemented, along with its workflows.

2.1 AI-assisted personalized nutrition (AIPN)

Food is a fundamental human need. Human body requires a number of basic nutritional elements to sustain life. Dietary practices and food choices depend on many factors, including, lifestyle, cultural background, socio-economic standing and location. Balanced nutritional intake is crucial for health and longevity. Many chronic diseases, such as type 2 diabetes, obesity, hypertension, stroke, and neuro-degenerative diseases, are related to dietary habits [37]. According to a report from the World Health Organization (WHO) [2], unhealthy diet along with sedentary lifestyle was the second-most leading contributing factor behind the 17 million deaths from cardiovascular and brain diseases. To reduce the risk of mortality from these chronic conditions, a healthy diet with adequate nutritional content is essential. An unhealthy diet is often characterized by foods with added sugars, and saturated fats, high sodium and low dietary fiber content. However, due to our busy urban life-style it is not easy to do advance meal planning and preparation. Moreover, the assessment of the nutritional quality of a meal that is

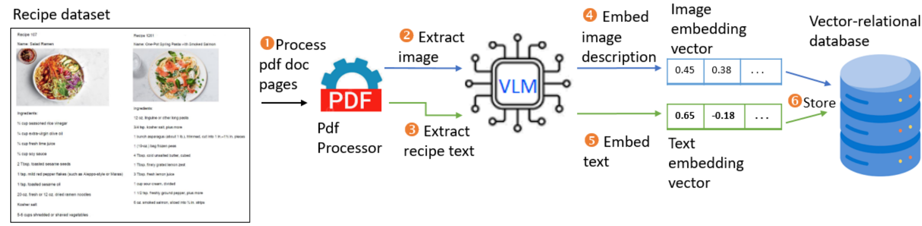


Fig. 1: Data ingestion workflow

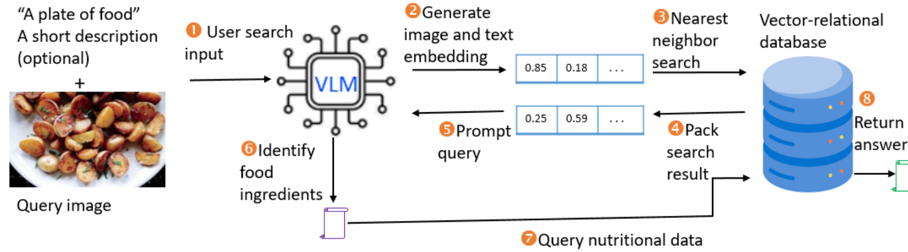


Fig. 2: User search workflow

appropriate for the health of a person requires domain expertise. In this context, AI-assisted personalized nutrition can play an important role to promote healthy eating habits.

With the growth of available food data and the advent of LLMs, several recent studies explored the application of LLMs for tasks such as ingredient detection and nutrition assessment. They include FoodLMM [40], FoodSky [43], CalorieLLaVA [38], and LLaVA-Chef [32]. However, these approaches mainly focused on developing domain specific customized LLM based solutions.

2.2 AIPN prototype implementation and workflows

We have developed a prototype application for AI-assisted personalized nutrition support. The primary use-case involves a user taking an image of her meal and posting this to our application, which generates a list of ingredients detected in the food and a list of nutritional components. This application can help a user determine whether her meal is healthy or not, based on the nutritional content.

There are two main workflows within our system that enable the prototype application. The first workflow involving data ingestion is shown in Figure 1. Step 1 entails processing each page of a pdf document, a Recipe Book, consisting of many food recipes, each with a picture and a text description. The details of the recipe dataset are discussed in Section 4.2. From each page of the pdf document, the image of the dish (if any) and a description of the ingredients in the recipe are extracted, which are then sent to the VLM as a prompt query (steps 2 and 3). The VLM generates a text description of the image, from which a

Table 1: VLM embedding capability

Comparison criteria	LayoutLMv3 [27]	Moondream [14]	CLIP-ViT-B-32 [5]	LLaVA-Llama3 [12]	Granite Vision3.2 [39]
Number of parameters	100M	1.4B	86M	8.0B	2.5B
Context length	512	2048	77	8192	16384
Embedding dimension (length)	768	2048	512	4096	2048

vector embedding is created (step 4). Similarly, the description of the recipe ingredients is also used to create another vector embedding (step 5). Then both embeddings are stored in a table (or tables) within the vector-relational database (step 6).

The second workflow involving user search is depicted in Figure 2. In step 1, the user submits a query image, such as a plate of food, a small text description (optional) and a short search query. The text description and query image are used to generate corresponding text and image embedding vectors using a text embedding model and a VLM based image embedding model respectively (step 2). These embedding vectors may be combined depending on vector search strategy (described in Section 3.2) and then submitted as part of nearest neighbor query/queries to the vector-relational database (step 3). All the matching query results are packed in step 4 and then sent as a prompt query back to the VLM (step 5). In step 6, the VLM generates a list of food ingredients for the user's query image. With this list of ingredients a query is issued to the vector-relational database (step 7) to determine the nutritional components. Based on this query, the result is returned back to the query user (step 8).

3 Methodology

In this section we describe our methodology and discuss the different aspects that we have considered in our analysis.

3.1 Aspect 1: Vision Language Models (VLM) with multi-modal embedding and prompt response

Large Language Models (LLMs) have revolutionized natural language processing tasks, including translation, text summarization, and question answering [35]. Vision-Language Models (VLMs) are the next step in this progression. VLMs are capable of integrating both text and visual inputs, such as image and videos and perform tasks involving those inputs. Several VLMs have already been developed. Based on the two workflows described in Section 2.2, two different capabilities are considered: i) generation of vector embeddings from (texts and) images, and

Table 2: VLM prompt (chat) capability

Comparison criteria	Moondream [14]	GraniteVision3.2 [39]	LLaVA-Llama3 [12]	Qwen2.5-VL [19]
Number of parameters	1.4B	2.5B	8.0B	8.3B
Context length	2048	16384	8192	128000
Embedding dimension (length)	2048	2048	4096	3584
Prompt response quality	○	◐	◐	●

●: excellent; ◐: good; ○: fair.

ii) generation of prompt (chat) responses based on user inputs and provided contexts.

To determine a suitable model for the first task of embedding generation from images, we considered several VLM embedding models. They are LayoutLMv3 [27], Moondream [14], CLIP-ViT-B-32 [5], LLaVA-Llama3 [12] and GraniteVision3.2 [39]. Their key features are summarized in Table 1, including, the number of model parameters, context length and embedding dimension (length). The *number of parameters* refers to the adjustable weights and biases within a neural network. The larger the number of parameters of a model, the more complex it is. As can be seen in the table, this value ranges from 86M with CLIP-ViT-B-32 to 8.0B with LLaVA-Llama3. The *context length* represents the maximum number of tokens a model can process at once i.e. in a single prompt [23]. Among these models, CLIP-ViT-B-32 has the smalled context length of 77, while GraniteVision3.2 has the longest context length of 16384. *Embedding dimension (length)* identifies the number of floating-point numerals used to represent a piece of data as a vector of numbers. This is the length of the embedding vector generated by a particular embedding model. It can be expected that a more complex model will usually take longer to generate embedding than a relatively simpler model.

For the second task of prompt (chat) response generation, several VLM models are considered. They are (as shown in Table 2): Moondream [14], GraniteVision3.2 [39], LLaVA-Llama3 [12] and Qwen2.5-VL [19]. Note that, not all VLM models support embedding creation, whereas not all embedding models support prompt response generation. Therefore, not all the entries in Table 1 and Table 2 are the same. Among the prompt generation capable VLMs, Qwen2.5-VL has the highest number of models parameters (8.3B) and the longest context length of 128000. In contrast, Moondream has the fewest model parameters and the shortest context length. VLM models GraniteVision3.2 and LLaVA-Llama3 sit in the middle of the pack. Hence, Qwen2.5-VL is expected to be the most complex of these models. Finally, the criteria *Prompt response quality* refers to the degree to which a VLM model's prompt response matches the ground truth or the expected response. We discuss about this further in Section 4.6.

3.2 Aspect 2: Multi-modal vector search strategies

Multi-modal data can take different forms, such as text, image, audio and video. Due to their unstructured nature, these data formats cannot be directly queried. Instead, the data must fist be converted into a multi-dimensional feature vector representation by using different embedding techniques, and stored in the database storage. During query time, the feature vectors are retrieved and queries are evaluated. Depending on how vector embedding is created for each data modality and utilized during query processing, there can be different search strategies. We classify them into three categories.

1. Separate embedding based search and re-ranking (SEBSR). In this approach, for an object a separate embedding vector is created for each modality. For instance, a text embedding vector can be created with nomic-embed-text [33] and an image embedding vector is created with LayoutLMv3. The embedding vectors for each modality corresponding to all objects are search independently. These search results are combined and re-ranked to produce the final result-set.

2. Unified-domain embedding based search and ranking (UEBSR). If different modalities can be mapped into a shared unified high dimensional vector space, then a single embedding vector is created per object. These vectors can then be directly compared to evaluate their similarity. For instance, CLIP (Contrastive Language-Image Pretraining) [1] embeddings provide high-dimensional unified representations of text and images. An example CLIP model is CLIP-ViT-B-32.

3. Ensemble embedding based search and ranking (EEBSR). As with the SEBSR approach, in this case separate embedding vectors are generated for every object, with one vector for each modality. Then the different embedding vectors for a particular object are fused to create a single ensemble embedding vector per object. There are different techniques to perform this embedding fusion. In our study, we utilize three different techniques: ConcatMLP [6], MLB [29] and MFH [41].

3.3 Aspect 3: Vector-relational databases

As previously mentioned, we consider two open-source vector-relational databases, namely, PostgreSQL (with pgvector extension) and SQLite (with vector extension). The primary datatype supported by PostgreSQL is `vector` that can be used to represent the data type of the vector embeddings generated by a language model. It also supports data type `halfvec` to store half-precision vectors. SQLite does not have a vector data type, but it supports vector data through the BLOB data type. Further details regarding the vector-relational features of PostgreSQL and SQLite are provided in Section 4.3.

3.4 Aspect 4: Vector data indexing strategies

Vector data indexing is integral to efficiently supporting vector similarity queries in a vector-relational database. For instance, PostgreSQL (with pgvector) sup-

Recipe 107

Name: Salad Ramen

Ingredients:

½ cup seasoned rice vinegar

¼ cup extra-virgin olive oil

¼ cup fresh lime juice

¼ cup soy sauce

2 Tbsp. toasted sesame seeds

1 tsp. mild red pepper flakes (such as Aleppo-style or Maras)

1 tsp. toasted sesame oil

20 oz. fresh or 12 oz. dried ramen noodles

Kosher salt

5-6 cups shredded or shaved vegetables

Recipe 1261

Name: One-Pot Spring Pasta with Smoked Salmon

Ingredients:

12 oz. linguine or other long pasta

3/4 tsp. kosher salt, plus more

1 bunch asparagus (about 1 lb.), trimmed, cut into 1 in.–1½ in. pieces

1 (10-oz.) bag frozen peas

4 Tbsp. cold unsalted butter, cubed

1 Tbsp. finely grated lemon zest

3 Tbsp. fresh lemon juice

1 cup sour cream, divided

1 1/2 tsp. freshly ground pepper, plus more

6 oz. smoked salmon, sliced into ½ in. strips

Fig. 3: A few example (recipe) entries from the dataset

ports two different index types: Inverted File with Flat Compression (IVFFlat) [10] and Hierarchical Navigable Small World (HNSW) [31]. An IVFFlat index operates by partitioning data vectors into a number of lists of vectors, which is a configurable parameter, during index construction time. Then while executing a vector search, it improves query efficiency over an exhaustive search approach by needing to search only a subset of those lists that are closest to the query vector. An HNSW index constructs a multilayer graph during index building time. In an HNSW index, proximity graphs are constructed with input vector data points. In a proximity graph nodes represent vector embeddings, whereas edges are the connections between nodes. The length or weight of each edge is determined by the distance between the corresponding nodes, calculated using a distance function, such as L2 (Euclidean) or cosine. Multiple layers of the proximity graph are organized such that the top layers contain graphs with fewer nodes for long-range navigation, and lower layers have graphs with more nodes for detailed search.

4 Experimental evaluation and analysis

In this section, we discuss our experimental evaluation of multi-modal RAG search, with an emphasis on vector-relational database operations. In Section 4.1 we describe the experimental settings, and in Section 4.2 we describe the datasets. The vector-relational databases that we used and workload queries are discussed in Section 4.3, along with some details of the vector indexing techniques in Section 4.4. Experimental evaluation of the embedding capability and prompt (chat) capability of VLMs are presented in Sections 4.5 and 4.6 respectively. Evaluation of data ingestion and multi-modal search with our AIPN application are presented in Sections 4.7 and 4.8 respectively. Then we evaluate the scalability of vector-relational databases, with the vector index build time evaluated in Section 4.9 and query execution time in Section 4.10.

4.1 Experimental settings

We used a machine, with an Intel Xeon w3-2423 processor having 6 cores and 2 threads per core, NVIDIA RTX A1000 GPU, and 32 GB of main memory. The machine runs Ubuntu 24.04 LTS. The code was implemented in Python and Ollama [15] was used as the framework to run VLMs locally on our machine.

Table 3: Details of the datasets

Dataset	**Type**	**Cardinality**	**Vector dimension (embedding length)**
Real-world dataset			
recipe	Real-world	13,582	Varies, depends on embedding model (see Table 1)
Synthetic datasets with different scale factors (sizes)			
items10K	Synthetic	10,000	512
items100K	Synthetic	100,000	512
items1M	Synthetic	1,000,000	512
items10M	Synthetic	10,000,000	512

4.2 Datasets

We used two different datasets: a real-world dataset and a synthetic dataset. The real-world dataset consists of a `recipe` table and three tables related to food nutrition. The `recipe` data table, as shown in Table 3, is based on the Food Ingredients and Recipes Dataset [8] created from Epicurious [7]. It consists of 13,582 images of different food dishes and a csv file containing text description of the images. For this experimental study, a Recipe Book was created as a pdf file, containing 13,582 recipes, with an image along with the associated

text description for each recipe. Figure 3 shows a few example recipes from the Recipe Book. The food nutrition data tables are based on The Food Database (FOODB) [21]. It is considered as the world's most comprehensive resource on food nutritional content and their biochemistry information. There are three tables to capture and maintain information based on the csv files obtained from FOODB. These tables are: `food`, `content` and `compound`. They are not shown in Table 3, as they do not involve any vector embedding or vector-relational operation and we do not evaluate any query execution on these three tables.

The synthetic dataset were generated to evaluate vector-relational database scalability. It consists of four tables, each with a column *embedding*. These tables are `items10K`, `items100K`, `items1M` and `items10M`, with each table containing 10× more records than the previous table respectively. The smallest table `items10K` contains 10 thousand records, whereas the largest table `items10M` consists of 10 million records.

4.3 Vector-relational database and queries

To evaluate vector-relational database features, we have selected two open-source relational databases with vector extensions. We have chosen PostgreSQL 18.1 with pgvector 0.8.1 extension and SQLite with vector extension as the vector-relational databases.

PostgreSQL. PostgreSQL with pgvector extension supports a vector data type, which can be declared with the keyword `vector`. It supports nearest neighbor queries based vector distance metrics, such as L2 distance (<->) and cosine distance (<=>). We illustrate the creation of a table with a vector column in PostgreSQL with the `items10K` table from the synthetic dataset.

Create table (PostgreSQL)

```
-- Create a table with a column embedding of type vector.
CREATE TABLE items10K (
    id serial PRIMARY KEY,
    embedding vector(512)
);
```

A cosine distance based nearest neighbor query on `items10K` is shown next. Note that the *qry_vector* symbolizes the query vector, which is used to compute cosine similarity against the `embedding` column entries of all rows in the `items10K` table.

Query (PostgreSQL)

```
-- Nearest neighbor query based on cosine distance.
SELECT id,  embedding <=> $qry_vector$::vector AS score
FROM items10k
ORDER BY  score limit 5;
```

SQLite. SQLite with vector extension does not support a dedicated vector data type, but rather it stores vector data in a BLOB field. This is shown in the illustration below, with `items10K` table. To initialize the BLOB field as a vector column, a User-Defined Function (UDF) `vector_init` is used. Then, the UDF `vector_quantize` is invoked to quantize the initialized vector column entries.

Create table (SQLite)

```
-- Create a table with a column embedding of type BLOB.
CREATE TABLE items10K (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    embedding BLOB
);

-- Initialize the vector. By default, the distance
-- function is L2. Specify a cosine distance metric.
SELECT vector_init('items10K', 'embedding',
'type=FLOAT32,dimension=512,distance=COSINE');

-- Quantize vector.
SELECT vector_quantize('items10K', 'embedding');
```

SQLite supports cosine distance based nearest neighbor query, an example of which is shown below. SQLite utilizes the UDF `vector_quantize_scan` to compute cosine distance of the query vector against the vectors in a table, where *`$qry_vector$`* is the query vector.

Query (SQLite)

```
-- Nearest neighbor query based on cosine distance
-- on the quantized version.
SELECT d.id,  v.*
FROM items10K AS d
JOIN vector_quantize_scan('items10K', 'embedding',
vector_as_f32($qry_vector$), 5) AS v
ON d.id = v.rowid;
```

4.4 Vector indexing

Vector indexing is not supported by SQLite (vector extension), whereas PostgreSQL (pgvector) support two techniques: IVFFlat and HNSW. These techniques can be used to construct indexes on the embedding vectors stored in a table with a `vector` column.

Note that, as indicated in Table 1, the dimensions (lengths) of the embeddings generated by different VLMs range from 512 to 4096. For instance, Moondream generates embeddings of dimension (length) 2048. When we attempted to create a vector index (IVFFlat or HNSW) on embedding vectors generated by Moondream, we received the following error message:

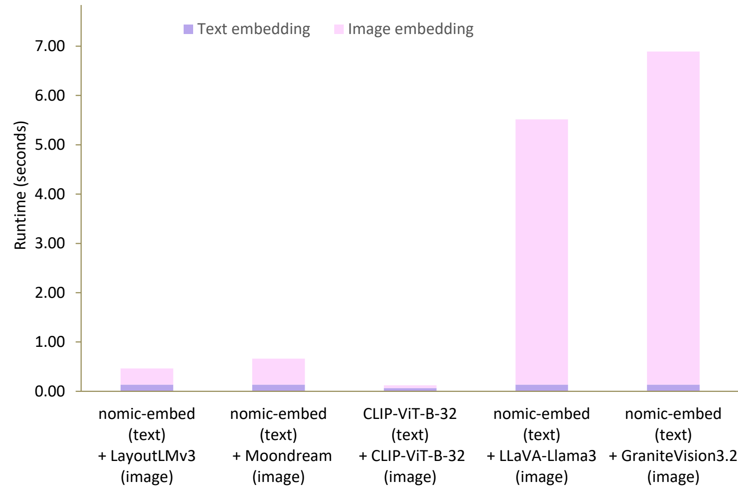


Fig. 4: Total embedding generation (text and image) time

> ERROR: column cannot have more than 2000 dimensions for ivfflat index

It suggests that the maximum length of a vector-indexed column with `vector` data type is 2000 in PostgreSQL. It appears that [9] this is due to the default size of the PostgreSQL page, which is 8KB and is not adjustable. This restricts the number of 4-byte floats that can be stored on a page in PostgreSQL. Based on this observation and VLM evaluation results from the next section, for our subsequent experimental evaluation in sections from Section 4.7 to 4.10, we utilize two VLMs, LayoutLMv3 and CLIP-ViT-B-32, which generate embeddings of dimension less than 2000.

4.5 Evaluation: VLM/LLM embedding capability

During a multi-modal RAG-guided search, the user specifies a text and an image as inputs. These inputs need to be converted into embedding vectors, so that a vector-relational database can be queried with them to find the top-K matches using nearest neighbor queries.

In this section, we evaluate the capability of embedding models, particularly that of VLM models to generate embeddings from images. We evaluate 5 different VLM models: LayoutLMv3, Moondream, CLIP-ViT-B-32, LLaVA-Llama3 and GraniteVision3.2. For text embedding, nomic-embed-text is used along with all VLM models, except for CLIP-ViT-B-32. Since, CLIP-ViT-B-32 utilizes a unified embedding domain, we use it for both text and image embedding. The performance of these approaches are shown in Figure 4, and as can be seen CLIP-ViT-B-32 performs the best in terms of the lowest image embedding time. LLaVA-Llama3 and GraniteVision3.2 take the longest times, with

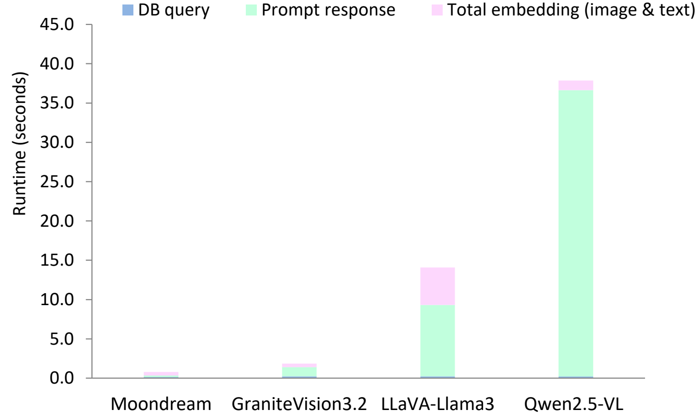
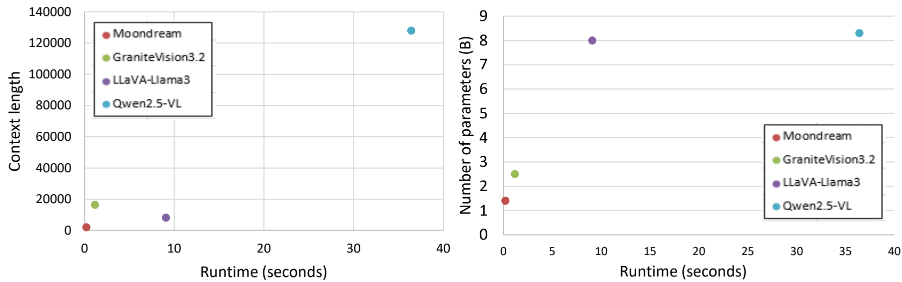


Fig. 5: Overall response time



(a) Context length vs. prompt response time

(b) Num. params (B) vs. prompt response time

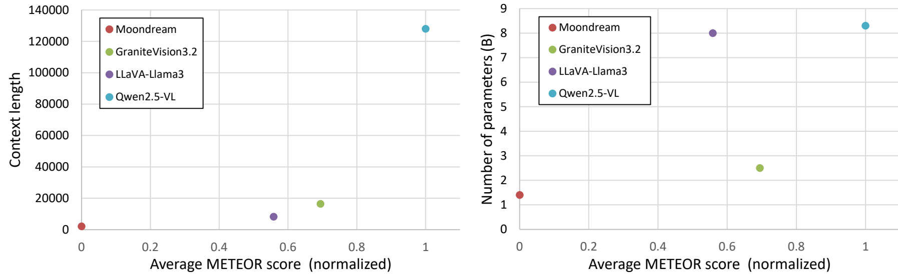
Fig. 6: Prompt response time: context length vs. number of parameters (Billion)

GraniteVision3.2 performing the worst. To explain these results, we observe the features of these VLM models in Table 1. GraniteVision3.2 has the longest context length of 6384, whereas CLIP-ViT-B-32 has the shortest context length of 77. Also, CLIP-ViT-B-32 has the fewest number of model parameters of 86M.

Based on these findings, LayoutLMv3 is utilized as the default image embedding model and nomic-embed-text as the default text embedding model in the subsequent experiments, except for cases where CLIP-ViT-B-32 is applicable.

4.6 Evaluation: VLM prompt (chat) capability

After retrieving the top-k matching nearest neighbors based on the text and image embedding, a context is created using the best matches. Then a prompt query along with the context is issued to a VLM chat model. To provide an ap-



(a) Context length vs. prompt quality (b) Num. params (B) vs. prompt quality

Fig. 7: Prompt quality: context length vs. number of parameters (Billion)

pripriate response to the user's prompt query, the VLM model uses the provided context.

In this section, we evaluate the performance of the prompt capability of several VLM models. These models are: Moondream, GraniteVision3.2, LLaVA-Llama3 and Qwen2.5-VL. In terms of response generation latency, as shown in Figure 5, Moondream has the lowest response latency, whereas Qwen2.5-VL has the longest latency. To further analyze this, we plot the prompt response time and the corresponding context length of the VLM models in Figure 6a. We observe that they are not always correlated, for instance, LLaVA-Llama3 has a smaller context length than GraniteVision3.2 but higher runtime than that of GraniteVision3.2. Figure 6b shows the number of model parameters (Billion) vs. the runtime of the VLM models. The figure indicates that a higher number of model parameters generally imply a longer runtime. Qwen2.5-VL has the most number of model parameters (8.3B, see Table 2) and the highest runtime of prompt response generation among the VLM models.

A lower runtime does not imply a better performance for a particular VLM model with regards to prompt response, since the quality of the response matters significantly. To evaluate the prompt response quality, we compute the METEOR score [25]. We plot of the average METEOR score of the VLM models normalized over the best observed score against (a) the context length in Figure 7a, and (b) the number of model parameters (Billion) in Figure 7b. The figures suggest that a longer context length typically implies a better prompt response quality. In contrast, this is not necessarily the case with the number of model parameters. Overall, the prompt response quality of Moondream was the worst, whereas the response quality of Qwen2.5-VL was the best. These observations are noted with (partially-)colored circles in Table 2. In future, we intend to conduct a more thorough evaluation of prompt response quality.

4.7 Evaluation: data ingestion

In this section, we evaluate the performance of the data ingestion pipeline of our AIPN application, which involves generating embedding vectors from the

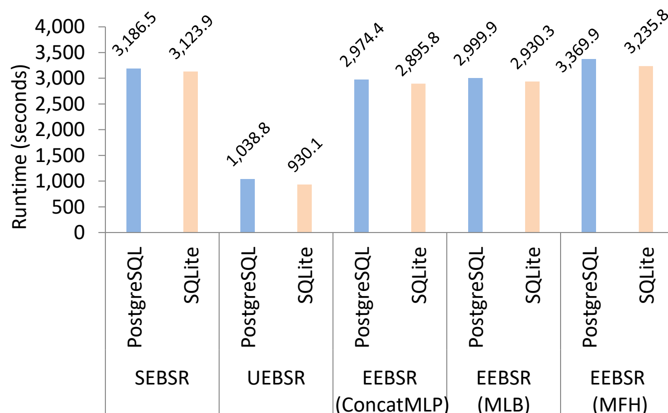


Fig. 8: Data ingestion performance (PostgreSQL vs. SQLite) with different multi-modal vector search strategies

input datasets, and then storing them in the `recipe` table in a vector-relational database. We compare the performance of two open-source vector-relational databases: PostgreSQL (with pgvector) extension and SQLite (with vector extension). Note that, in these experiments no vector index is created for PostgreSQL and the dataset used is `recipe` (see Table 3). We vary the vector search strategies, which determine how the text embedding vector and image embedding vector are combined. These strategies are: SEBSR, UEBSR, EEBSR (ConcatMLP), EEBSR (MLB) and EEBSR (MFH). The results are shown in Figure 8. As can be seen, UEBSR approach performs the best, and with both databases. The reason is that UEBSR utilizes CLIP-ViT-B-32 for both text and image embedding generation. With all other vector search strategies, nomic-embed-text is used as the text embedding model, whereas LayoutLMv3 is used as the image embedding model. Between the two databases, SQLite performs slightly better than PostgreSQL.

4.8 Evaluation: Search performance with different multi-modal vector search strategies

We evaluate the overall multi-modal RAG search performance in our AIPN application. In this search process, the user specifies a query image, a small text description of the image and a brief search text. From these inputs corresponding text and image embeddings are generated, and they are combined using one of the vector search strategies: SEBSR, UEBSR, EEBSR (ConcatMLP), EEBSR (MLB) and EEBSR (MFH). The combined vector is searched in the vector-relational database table `recipe` to find the nearest neighbor matches on cosine distance and then to create a context using the query results. This context and

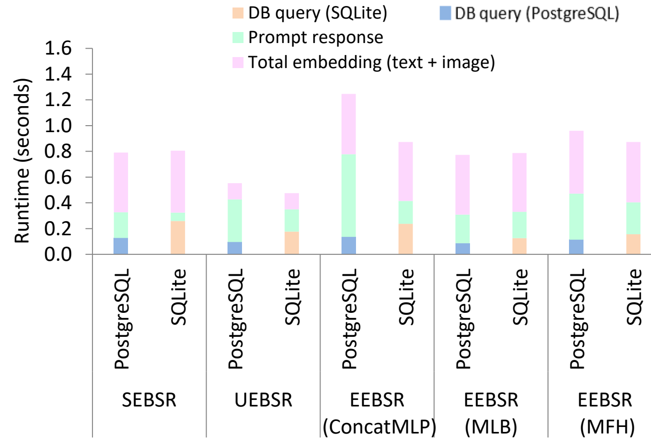


Fig. 9: User search performance (PostgreSQL vs. SQLite) with different multi-modal vector search strategies

the user's search text is then sent to the VLM prompt model. We use Moondream as the VLM for prompt response and the dataset used is the `recipe` dataset. Figure 9 shows the overall user search performance, with a breakdown of times spent in each step. As can be seen, the prompt response time constitutes a significant portion of each bar in the stacked bar chart. The total embedding generation time (text and image) varies significantly from approach to approach, taking the least amount of time for UEBSR and significantly longer times for EEBSR approaches. Database query execution times remain steady across different scenarios for both the databases. As for the overall time, USBSR shows the best performance with the lowest overall runtime, as it requires the lowest prompt response time owing to CLIP-ViT-B-32.

4.9 Evaluation: vector-relational database scalability - vector index build time (PostgreSQL)

In this section and the next section, we focus on the scalability of vector-relational databases. For this purpose, we use the synthetic datasets, which differ in data volumes (see Table 3).

In this section, we evaluate the different vector indexing techniques. Specifically, we evaluate the construction time of two vector indexes supported by PostgreSQL (with pgvector extension), namely, IVFFlat and HNSW. SQLite (with vector extension) does not support any vector index. Note that, we deployed the databases as "out-of-the-box" and did not change any configuration parameters to tune them.

Figure 10 shows the time to build index on the tables in the synthetic datasets: items10K, items100K, items1M and items10M. For all the four ta-

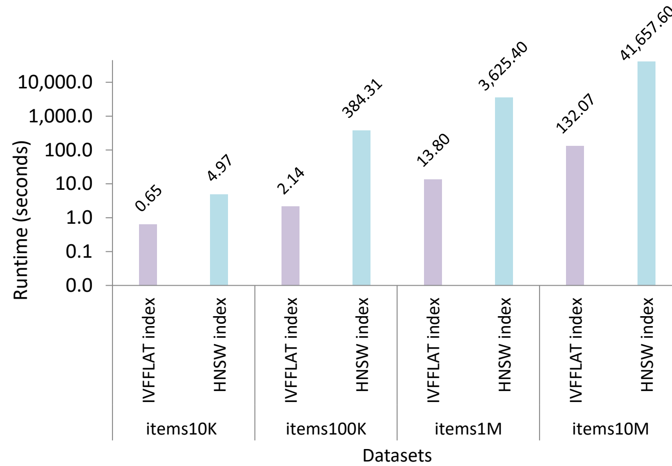


Fig. 10: PostgreSQL vector index construction time

bles, IVFFlat was faster than HNSW in terms of index building time. This is expected, as HNSW is a more complex index structure than IVFFlat, and HNSW construction involves building multilayer proximity graphs based on the embedding vectors. However, what is somewhat surprising is that as the table size increases, the performance gap (index building time) between the two indexes grow. For instance, with items10K table the index building time of IVFFlat is 7.6× faster than that of HNSW, while with items10M table, IVFFlat is 315.4× faster. Although index construction is usually done once and *a priori*, these results may provide some guidance for vector index selection, particularly if the *time-to-query* is an important consideration.

4.10 Evaluation: vector-relational database scalability with query execution time (PostgreSQL vs. SQLite)

Next we present the results based on the query execution time, by executing the vector search query directly on the corresponding database tables in the synthetic dataset. To evaluate the impact of data volume and indexing on the query performance, our evaluation involves three different versions of PostgreSQL setup: PostgreSQL without any index, and PostgreSQL with the two vector indexes, IVFFlat and HNSW. Note, we only evaluate SQLite without any index due to a lack of support for a vector index. The performance results are plotted in Figure 11. As can be observed, PostgreSQL generally performs better than SQLite. Regarding the impact of using vector indexes on query performance, PostgreSQL with either of the indexes, IVFFlat or HNSW, performs better than PostgreSQL without any index. Between, the two indexes, there is no appreciable difference in performance at lower scale factors. However, at the highest scale factor, i.e.

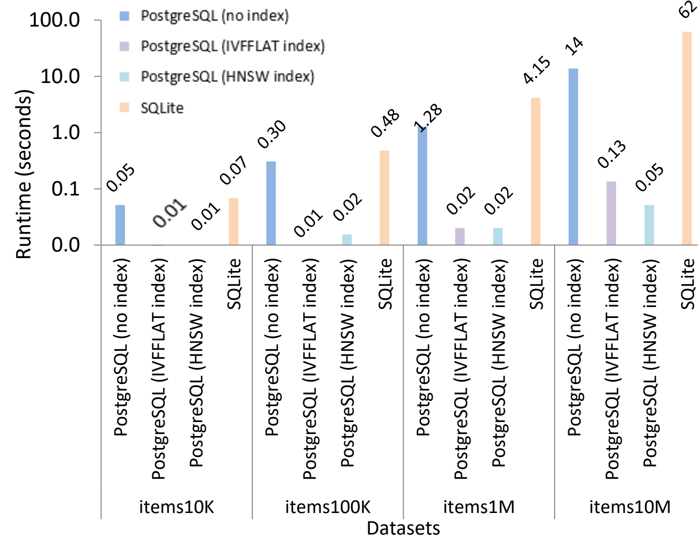


Fig. 11: Query performance (PostgreSQL vs. SQLite) with different data sizes

with the items10M table, PostgreSQL (HNSW) is slightly faster than that of PostgreSQL (IVFFlat).

5 Conclusion and future research directions

We developed an AI-assisted personalized nutrition application, as a novel use-case of multi-modal RAG search with VLM and vector-relational database. We have conducted a systematic evaluation of several aspects of current VLM systems and two vector-relational database PostgreSQL (with pgvector) and SQLite (with vector). Our experimental study reveals several issues that are summarized next. (1) The performance of embedding models, in terms of embedding generation latency, varies widely and it depends on factors such as context length and the number of model parameters. (2) The latency of VLM models' prompt (chat) response varies widely as well, depending on the number of model parameters and other factors. Moreover, a longer context length generally implies a better prompt response quality. (3) The supported vector features in the evaluated vector-relational databases differ significantly, in terms of dedicated vector data type, query syntax, storage constraint and query performance. If vector indexing is required, PostgreSQL limits the vector embedding dimension length to a maximum of 2000. Therefore, it can only index vector embeddings with dimension length less than 2001. (4) Vector indexes can significantly improve (reduce)

query latency. However, index construction time, such as with HNSW, can be substantially higher than that of IVFFLAT. (5) The vector embedding generation and high quality prompt response processes can be significantly time-consuming with the more advanced VLMs, even with consumer-grade GPUs. It is hoped that these findings will provide some guidance to the practitioners.

Future work will involve a more detailed evaluation of the prompt response quality and retrieval effectiveness with appropriate metrics. Another future work is to evaluate more complex queries with RAGs, including multi-hop queries [34]. Exploring other modalities, besides text and image, is another area of future investigation. The evaluation of the energy efficiency of different approaches, particularly of embedding generation and prompt response, is also a potential direction of future work.

References

1. Learning transferable visual models from natural language supervision. In: ICML. vol. 139, pp. 8748–8763 (2021)
2. World health organization (2022), WHO Manual on Sugar-Sweetened Beverage Taxation Policies to Promote Healthy Diets; World Health Organization, Geneva
3. ChatGPT (2026), https://chatgpt.com
4. ChromaDB (2026), https://www.trychroma.com/
5. CLIP-ViT-B-32 (2026), https://huggingface.co/sentence-transformers/clip-ViT-B-32
6. ConcatMLP (2026), https://github.com/Cadene/block.bootstrap.pytorch?tab=readme-ov-fileconcatMLP
7. epicurious (2026), https://www.epicurious.com/
8. Food Ingredients and Recipes Dataset with Images (2026), https://www.kaggle.com/datasets/pes12017000148/food-ingredients-and-recipe-dataset-with-images/
9. Increase max vectors dimension limit for index (2026), https://github.com/pgvector/pgvector/issues/461
10. IVFFlat (2026), https://github.com/pgvector/pgvector#ivfflat
11. Llama (2026), https://huggingface.co/meta-llama
12. Llava-llama3: Large language and vision assistant with llama3 (2026), https://github.com/Michel-liu/LLava-LLama3
13. Milvus (2026), https://milvus.io/
14. moondream (2026), https://github.com/vikhyat/moondream
15. Ollama (2026), https://ollama.com/
16. pgvector (2026), https://github.com/pgvector/pgvector
17. Pinecone (2026), https://www.pinecone.io/
18. Qdrant (2026), https://qdrant.tech/
19. Qwen2.5-vl vision-language model series based on qwen2.5 (2026), https://huggingface.co/collections/Qwen/qwen25-vl
20. SQLite Vector (2026), https://github.com/sqliteai/sqlite-vector
21. The Food Database (FOODB) (2026), https://foodb.ca/
22. Weaviate (2026), https://weaviate.io/platform
23. What is a context window? (2026), https://www.ibm.com/think/topics/context-window

24. Aumüller, M., Bernhardsson, E., Faithfull, A.: Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. Information Systems **87** (2020)
25. Banerjee, S., Lavie, A.: METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In: ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization. pp. 65–72 (Jun 2005)
26. Bordes, F., et al.: An introduction to vision-language modeling (2024), https://arxiv.org/abs/2405.17247
27. Huang, Y., Lv, T., Cui, L., Lu, Y., Wei, F.: Layoutlmv3: Pre-training for document ai with unified text and image masking (2022), https://arxiv.org/abs/2204.08387
28. Kang, G., Ge, Z., Hu, J., Zhang, X., Wang, L., Zhan, J.: Bigvectorbench: Heterogeneous data embedding and compound queries are essential in evaluating vector databases. Proceedings of the VLDB Endowment **18**(5), 1536–1550 (2025)
29. Kim, J., On, K.W., Lim, W., Kim, J., Ha, J., Zhang, B.: Hadamard product for low-rank bilinear pooling. CoRR **abs/1610.04325** (2016)
30. Lewis, P., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. In: NeurIPS (2020)
31. Malkov, Y.A., Yashunin, D.A.: Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs . IEEE TPAMI **42**(04) (2020)
32. Mohbat, F., Zaki, M.J.: Llava-chef: A multi-modal generative model for food recipes. In: CIKM. p. 1711–1721 (2024)
33. Nussbaum, Z., Morris, J.X., Duderstadt, B., Mulyar, A.: Nomic embed: Training a reproducible long context text embedder (2025), https://arxiv.org/abs/2402.01613
34. Osipjan, A., Khorashadizadeh, H., Kessel, A.L., Groppe, S., Groppe, J.: Graphtrace: A modular retrieval framework combining knowledge graphs and large language models for multi-hop question answering. Computers **14**(9) (2025)
35. Raffel, C., et al.: Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of machine learning research **21**(140), 1–67 (2020)
36. Simhadri, H.V.: big-ann-benchmarks: Framework for evaluating ANNS algorithms on billion scale datasets
37. Tafuri, D., Latino, F.: Association of dietary intake with chronic disease and human health. Nutrients **13**(3) (2025)
38. Tanabe, H., Yanai, K.: Caloriellava: Image-based calorie estimation with multimodal large language models. In: ICPR. p. 63–75 (2025)
39. Team, G.V., Karlinsky, L., et al.: Granite vision: a lightweight, open-source multimodal model for enterprise intelligence (2025), https://arxiv.org/abs/2502.09927
40. Yin, Y., et al.: Foodlmm: A versatile food assistant using large multi-modal model. IEEE Transactions on Multimedia (2025)
41. Yu, Z., Yu, J., Xiang, C., Fan, J., Tao, D.: Beyond Bilinear: Generalized Multimodal Factorized High-order Pooling for Visual Question Answering (2017), https://arxiv.org/abs/1708.03619
42. Zhao, W.X., et al.: A survey of large language models (2023), https://arxiv.org/abs/2303.18223
43. Zhou, P., et al.: Foodsky: A food-oriented large language model that can pass the chef and dietetic examinations. Patterns **6**(5) (2025)