Towards a Universal Query Representation for Multimodal Information Retrieval

Luca Rossetto
luca.rossetto@dcu.ie
Dublin City University
Dublin, Ireland

Heiko Schuldt
heiko.schuldt@unibas.ch
University of Basel
Basel, Switzerland

Ralph Gasser
ralph.gasser@unibas.ch
University of Basel
Basel, Switzerland

Abstract

The field of information retrieval, especially when targeting multimodal content, has found ways of satisfying a broad range of information needs, which can be expressed in a multitude of ways. In contrast to related fields, such as relational databases, no universal way of representing the queries to be answered by a retrieval system has emerged. In this paper, we present an initial proposal for a universal query representation mechanism for multimodal information retrieval. The proposed approach imperatively expresses arbitrary information needs, using a DAG of query primitives. We show how such a representation can be used for both feature extraction and query processing pipelines and how it can serve as a foundation towards a query language for information retrieval.

CCS Concepts

• **Theory of computation** → *Database query languages (principles)*; *Database query processing and optimization (theory)*; • **Information systems** → *Database query processing*; **Query languages for non-relational engines**; **Query representation**.

Keywords

Multimedia Retrieval, Query Representation, Query Execution

ACM Reference Format:
Luca Rossetto, Heiko Schuldt, and Ralph Gasser. 2025. Towards a Universal Query Representation for Multimodal Information Retrieval. In *Proceedings of the 33rd ACM International Conference on Multimedia (MM '25), October 27–31, 2025, Dublin, Ireland.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3746027.3758155

1 Introduction

The field of machine-based information retrieval has developed over the last several decades from humble beginnings, where documents had to be manually and carefully categorized in order to be findable later, to its current state, where modern methods, often based on learned features, enable search in any digitally representable information using a multitude of complex queries that can express all sorts of different information needs. However, this wealth of possible information needs and query expressions poses a challenge in and of itself. While relational databases have relatively quickly converged to a standard mechanism for querying – SQL, a standard that has endured for more than a half century [1] – information retrieval lacks a comparable analog. The closest thing to a standardized mechanism comes not from explicit theoretical considerations, but from the popularity of certain implementations, such as Apache Lucene,[1] which forms the basis of several widely used text retrieval systems, such as, Apache Solr[2] or Elasticsearch[3].

When considering retrieval in media domains other than text, there is currently no established way for the expression of an information need. A multitude of different retrieval systems implement different interfaces and APIs, and generally, no two are compatible. Such systems can generally be understood as fixed-function pipelines with a small set of possible input parameters, rather than as flexible query engines capable of processing a broad range of different queries of different types. Especially in the face of interactive retrieval and multimedia analytics, this restriction is important.

In this paper, we introduce first steps towards a *universal query representation* for multimodal information retrieval and a prototypical implementation thereof. The proposed approach consists of a flexible information flow architecture, which is expressive enough to cover arbitrary information needs. In contrast to SQL, which *declares* a query and leaves the execution plan up to the database management system, our proposed approach uses an *imperative* definition, which uses an abstract information flow graph representation that can be transformed into a concrete execution plan by a retrieval system. We consider such a universal query representation to be a first, necessary step towards declarative query formulation, query execution planning and query optimization in multimedia retrieval systems, both of which are state-of-the-art in traditional databases.

The remainder of this paper is structured as follows: Section 2 provides a brief overview of current multimedia retrieval approaches and the way they handle query expression. Section 3 introduces our universal query representation scheme, while Section 4 discusses parallel and complementary representations in feature extraction. Section 5 then introduces our newest retrieval system, the *vitrivr-engine*, which serves as both a proof-of-concept implementation and a testbed for the development of our proposed approach. Section 6 then discusses challenges and opportunities that go beyond our current proof-of-concept and outline next steps, before Section 7 concludes the paper.

2 Related Work

Our proposed representation is, of course, not the first approach to ever attempt the expression of information retrieval queries systematically. Attempts can be found in the literature as far back as the 1990s [4, 9, 10, 13].

MM '25, Dublin, Ireland

ACM ISBN 979-8-4007-2035-2/2025/10
https://doi.org/10.1145/3746027.3758155

[1] See https://lucene.apache.org
[2] See https://solr.apache.org
[3] See https://www.elastic.co

One such early approach was MQuery [4], a purely visual query 'language'. In a manner similar to UML, MQuery enables the specification of both data schemas and queries. In contrast to our approach, MQuery is rather verbose and cumbersome due to its visual nature, which also makes it more difficult to parse.

Rather than inventing entirely new query languages for multimodal information retrieval, approaches such as MOQL [13] or SPARQL-MM [12] extend existing query languages not originally designed with multimodality. Both extensions augment their base languages (OQL and SPARQL, respectively) by certain primitives to capture spatial or temporal relationships. While these extensions support certain queries that were difficult to express using the base languages, neither is sufficiently expressive to capture arbitrary information needs.

A more theoretical approach can be found in [14], which defines a formal model of multimedia information retrieval. While this model is quite expressive and allows to reason about certain aspects of the retrieval process, its theoretical nature makes it impractical for actual query expression.

The possibly closest analog to our proposed approach is the MPEG Query Format [5], an XML-based query specification mechanism supporting a broad range of queries. The format specifies a number of query types, which can be combined into more complex representations. While this offers more flexibility compared to some of the 'fixed function' pipelines of other retreival systems, it ultimately is limited to the combination of such existing pipelines. In contrast, our approach aims to be more universal by breaking down the retrieval process into a series of base operators, rather than pre-defined query types.

3 Universal Query Representation

Fundamentally, one can ask why a *universal query representation* for multimedia search is desirable. We refer to the *exploration-search axis* as proposed by Zahálka and Worring [26] to construct an argument in favor. In their quest to fulfill an information need or generate an insight based on a multimedia collection, a human user typically iterates between different activities in a system (often via a user-interface), ranging from exploration of a collection to search in a collection. These different activities give rise to different types of workloads that go beyond the simple paradigms traditionally used in multimedia retrieval. Such workloads may not only involve basic retrieval operations, such as, *nearest neighbour search (NNS)*. Instead, they might involve more advanced operations such as fetching of relevant attributes (projection), score fusion, re-ranking, aggregation and more.

To this day, the orchestration and execution of such complex workloads – potentially across different components – are (re-)conceived and (re-)implemented for every multimedia retrieval system. This often creates a tight coupling between components, especially between user interfaces offering certain modes of interaction and the retrieval backends that execute the necessary workloads. What we envision is that similar to SQL being the interface for interacting with relational databases, a universal query representation (and language) for multimedia search could act as an interface to multimedia retrieval backend systems.

Our first draft of such a universal query representation consists of three parts: (i) A model of how arbitrary units of retrieval can be represented (data model), (ii) a model of how queries can be constructed given the data model (model for information flow) and (iii) a minimal set of operators that are required to facilitate multimedia retrieval use-cases.

3.1 Data Model

In order to be able to discuss about query representation, we must first get an understanding of the data that might be queried. In multimedia retrieval, the structure and semantics of this data is often dependent on the modality in question and the particular use-case. If we consider early text retrieval, for example, literature often refers to *documents* [2], which for the purpose of the data- and query model, act as a *unit of retrieval* and represent the object that contains the text, e.g., a PDF or an article on the Internet.

In multimedia retrieval, things are similar but a bit more complicated since any particular data collection may be organized in different ways. Very often, multimedia data collections can be naturally organized in terms of *files* being indexed, simply due to the fact that this is the way operating systems organize all data. In some cases, e.g., when searching an image collection, this may be sufficient since we are ultimately interested in the file that holds the desired information. A retrieval system must therefore simply be able to produce the files that satisfy a query.

However, if we consider more complex modalities such as video, for example, the content in the file may be further sub-divided into temporal chunks (e.g., shots) that – for the purpose of query evaluation – are considered as standalone units that can be retrieved on their own. Analogously, documents might be segmented across other axes as well, depending on the represented information. Such a segmentation is necessitated by the fact that knowing only the file may not be sufficient if it consists of hours of video material that must be reviewed manually. In such a scenario, the retrieval system must not only be able to connect between a query and the file. It should also single out the exact time points within the file, during which relevant content occurs.

In addition, in both of the aforementioned multimedia retrieval cases, different content representations may exist and be used for a particular unit of retrieval. Which of these representations is used, depends on the concrete query. Consequently, while in the traditional text retrieval model the document is the proxy for the content, multiple such proxies may exist in multimedia retrieval. In order to allow for the different granularities in the retrieval process, be they temporal, spatial or of any other nature, we propose the *retrievable* abstraction, which acts as the *unit of retrieval* in any multimedia retrieval system. How exactly retrievables are constructed and what they represent is use-case dependent. But for the purpose of storage and management, retrievables share a set of traits:

Proxy and Typing A retrievable is typically a representation of some real thing produced or processed during the indexing process. A retrievable may represent an entire *file*, a *temporal or spatial segment* thereof or some other relevant unit. The retrievable therefore determines *what* can be retrieved and we assume that in any given use case, there is a finite number of *retrievable types* known beforehand.

Fields and Descriptors A *retrievable* is described by different *descriptors*. Descriptors may be anything from technical information (e.g., a timestamp) to some derived feature representation (e.g., a word histogram, CLIP [16] vector, etc.). Most importantly, though, the available descriptors determine *how* a retrievable can be retrieved, since often, they are a representation of the retrievable's content. Descriptors are strongly typed and typically, the descriptors a retrievable can have – its *fields* – are predetermined by a schema.

Relationships Any retrievable may have arbitrary *relationships* to other retrievables. These relationships can be used to model dependencies, e.g., between a retrievable that represents a video file and the retrievables that represent the temporal segments within the file. Again we can assume that the types of relationships are known prior to indexing.

Ergo, we propose to model a media collection as an arbitrarily complex graph, where each vertex (i.e., retrievable) is described by attributes (i.e., descriptors) that act as an entry point for search operations. This retrievable graph is the data structure that is stored and managed by a multimedia retrieval system and it must be processed when dealing with different retrieval scenarios. With regards to this data structure, there are two practical comments to be made:

Firstly, while this retrievable graph may be very complex in terms of connectivity, one will find that in practice, the vertices may only have little to no explicit relationship among one another. Due to how this graph is translated to results, connectivity is not a requirement at all and even a set of disconnected retrievables, e.g., representing a set of files without any relationships among one another, can produce valid search results. Furthermore, even in the case of a highly connected graph, primitive search operations usually give rise to a very isolated snapshot thereof.

Secondly, it is therefore not necessary to store this data structure in an actual graph database. In fact, other types of databases (e.g., a vector database or relational database) may be better suited for particular use-cases. Consequently, the model presented herein is agnostic to the type of database used and is employed at a higher abstraction level.

3.2 Information Flow

Given the described retrieval graph we can now turn to how query *results* can be represented and how they can be constructed given a query. Typically, in multimedia retrieval, human users are presented with a ranked list of potentially matching objects (or a representation derived from such a list). In our model, a query can therefore be thought of as a transformation from the original retrievable graph to a sequence of retrievables – the results. The rank of the retrievable is its position in the sequence, which is often determined by a *score* descriptor.

It is worth pointing out that in the simplest case, any retrievable in the sequence is merely a projection of an instance found in the retrievable graph. These projections can still exhibit an arbitrary number of descriptors and can still have connections to other retrievables in the graph. Their presence or absence is merely a function of how the transformation from graph to sequence is designed. If a retrievable in the sequence has a direct correspondent in the retrievable graph, we call it *persistent* whereas derived retrievables not explicitly found in the graph are *transient*.

We now propose to represent such a transformation as a directed acyclic graph (DAG) of basic *query operators*, as common in many database management systems. Every query operator takes zero to many retrievables as input and generates zero to many output retrievables. Certain operators can take additional input types, such as user-defined data to be used during query evaluation. At the base of such a DAG lie fundamental (source) retrieval operations. These source(s) produce the initial sequence of retrievables. The resulting sequence is then transformed by downstream operators until the final form of the result as specified by the query is reached. Hence, retrievables flow from the base to the root of the DAG and are transformed in the process.

This imperative approach to query formulation by explicitly defining transformation operations offers a high degree of flexibility and expressiveness as it is only limited by the capability of the operators. At the same time, this structure allows for reasoning about query execution process. For example, in theory, semantically equivalent DAGs could be identified for the same query, whic is often done in database systems as part of a optimization process.

3.3 Operators

To complete the query representation model, we define a list of basic *query operators* that can occur in the proposed query DAG.

Retriever (Source) Retriever operators correspond to fundamental search operations used in retrieval. These operators are about finding the retrievables of interest, often based on a specific field. The operators often correspond to a basic database operation. Typical examples involve nearest neighbor search (NNS) on some vector field, text search on a text field or some Boolean comparison.

Field Projection Field projection operators fetch or generate specific descriptors for every incoming retrievable and append them. For example, a field projection can be used to fetch a `title` descriptor for every item in the result set.

Relationship Projection Relationship projection operators resolve relationships for every incoming retrievable and append the matching retrievables. For example, relationship projections can be used to fetch the retrievable that represents the file to which a retrieved (video-)segment belongs.

Selection / Filter Filters remove retrievables from the sequence that match a given predicate. For example, these operators can be used to filter based on an attribute value. In some cases, these filter operations can also be expressed by a *retriever*, however, in other cases late filtering leads to different results than early filtering.

Ranker Rankers change the order of retrievables in a sequence. For example, rankers can be used to change the ranking after an explicit scorer operation.

Transformers Transformers combine a sequence of 0 to N incoming retrievables into a sequence of 0 to M output retrievable. The semantics of such a transformation may differ and range from traversal of relationships to aggregation. Produced retrievables may be transient.

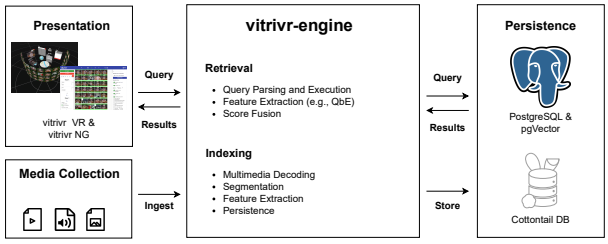

Figure 1: Architecture of the typical vitrivr deployment.

It is worth noting that this proposed list of possible query operators is not final. In theory, any type of operator semantic could be defined using our model. However, since these operators do not exist in isolation but should be thought of as a foundation for reasoning about queries and how queries can be executed efficiently, thinking in terms of categories and their common traits is a useful exercise. We will argue in Section 4 that identifying a canonical set of operators, from which complex queries can be constructed, is a necessary first step towards optimized query execution. This set of operators can then also serve as an intermediate representation, into which any future declarative query representation could be translated.

4 Parallels in Feature Extraction

In Section 3 we have described a universal query representation that can be used to model arbitrary multimedia collections and queries on these collections. For the sake of completeness, it is worth noting that the same representation can be employed during multimedia indexing, which in literature, is often treated as a separate matter that precedes querying.

Indexing often starts with a set of multimedia sources, which are being indexed and analysed for the purpose of later retrieval – in practice, we are often talking about a list of media files. The *indexing transformation* from these sources to a retrievable graph can again be modeled as a DAG, analogously to the query execution graph. The initial operator in such an indexing transformation is called *enumerator* and it generates one retrievable per source, with descriptors that contain the information relevant for accessing the multimedia data (e.g., a file path). These retrievables are then further processed by downstream operators in the DAG, some of which are somewhat specific to the indexing process:

Decoder Decoders load the media content from a source and append it to the retrievable, thus making it accessible for downstream analysis. For example, a video decoder would extract and append an image for each frame in the video.

Extractor Extractors analyse the content held by a retrievable and append a corresponding descriptor. Different extractors can be used to derive different descriptors. For example, an extractor using the CLIP model can produce a CLIP embedding vector from an image.

Other operations, such as projection, aggregation, and filtering, are equivalent to the ones described in the previous section. From a model perspective, the only difference between indexing and querying is that at some point, indexing requires access to the original content of a media object, whereas querying typically relies solely on the derived representations (i.e., the descriptors). That being said, any indexing activity might as well start from a query on an existing retrievable graph instead of an explicit list of files, thus further blurring the lines between retrieval and indexing, at least in theory. The fact that the same data and processing model can be applied to both the indexing and retrieval aspect of a multimedia retrieval system bears a certain elegance, that might provide room for interesting applications and synergies. However, we currently expect a formal model to be more relevant for the query and retrieval side of things.

5 Proof of Concept: vitrivr-engine

vitrivr[4] [20, 21] is an open-source, multimedia retrieval stack initially developed at the University of Basel. Since its inception almost ten years ago, vitrivr has seen a great deal of development at many different levels [8] and it has been successfully used in various research projects and recurring evaluation campaigns such as the Video Browser Showdown (VBS) [19] and the Lifelog Search Challenge (LSC) [18].

The vitrivr-engine [6] is the newest addition to the vitrivr ecosystem and acts as a successor to the extraction and query orchestration engine Cineast [20]. Our aim with the development of vitrivr-engine was to create an multimedia information retrieval engine that can be used more flexibly and can be adapted more easily to different applications and use cases. We therefore wanted to move beyond the very static and inflexible approach of Cineast. Most importantly, vitrivr-engine serves as both as a testbed and a feasibility study for the model outlined in Section 3. The source code for vitrivr-engine is available on GitHub under a permissive license.[5]

5.1 System overview

A typical setup of vitrivr with vitrivr-engine is illustrated in Figure 1. It exhibits a classical three-tier architecture with the following components:

Persistence Layer The persistence layer is used to store all the metadata required for describing and retrieving the multimedia objects. Using terminology from our model, it holds the persistent snapshot of the retrievable graph, that is, the retrievables themselves, the relationships, and the descriptors. Currently, we support PostgreSQL with pgVector[6] [25] or our own vector database Cottontail DB[7] [7]. However, the persistence abstraction in vitrivr-engine makes adding support for other databases straightforward.

Orchestration Layer (vitrivr-engine) The orchestration layer, implemented by vitrivr-engine, is responsible for executing the information flow DAGs for both feature extraction and querying. As such, this component is responsible for all the operations executed during multimedia retrieval and indexing. The vitrivr-engine also provides interfaces to the other layers in the stack. For example, it exposes the query functionality through an OpenAPI RESTful interface.

[4] See https://vitrivr.org
[5] See https://github.com/vitrivr/vitrivr-engine
[6] See https://github.com/pgvector/pgvector
[7] See https://github.com/vitrivr/cottontaildb

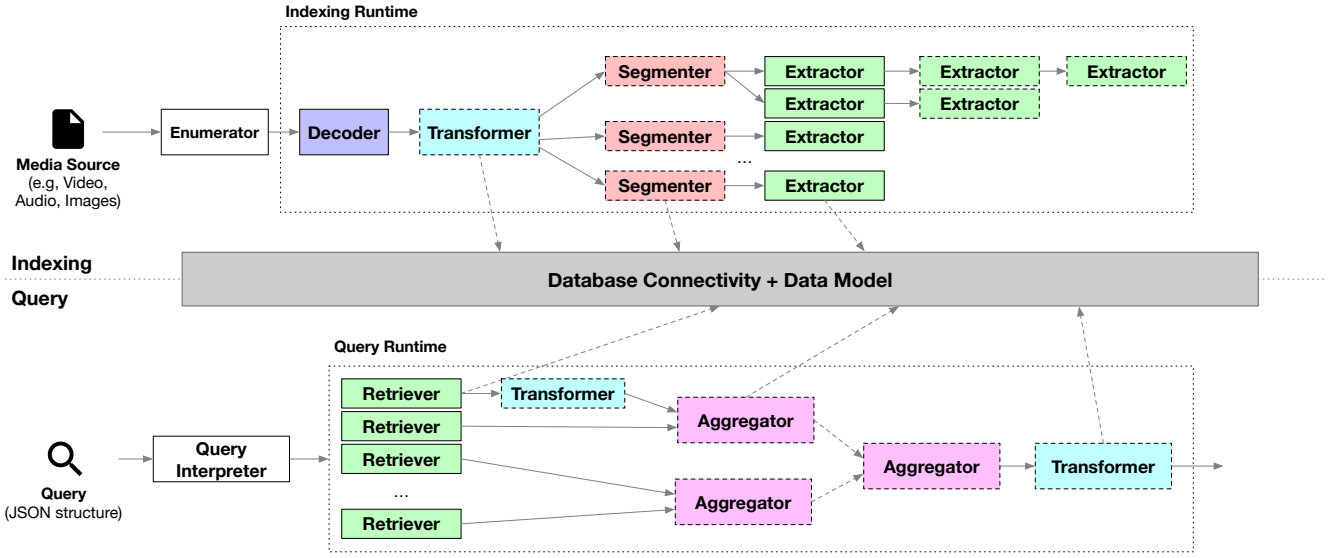

Figure 2: A high-level depiction of vitrivr-engine's internals. The runtime environments for indexing and querying are both modeled as a DAG of operators. The two parts share a common database connectivity and data model.

Presentation Layer The presentation layer is the interface between the retrieval system and its human users. It provides means to formulate queries and visualises results. Within the vitrivr ecosystem, we have developed several different such interfaces over the years, each with its own strengths and weaknesses [23, 24].

While the classical setup described herein is very common, and has already been employed in evaluation campaigns such as LSC 2024 [22], VBS 2024 [6], and VBS 2025 [17], vitrivr-engine has been designed in such a way that its components can also be used in the form of libraries to power other application. This is made possible by a high degree of modularity.

5.2 Internals

At the core of vitrivr-engine's internal design lie the different types of operators that are used to build the execution DAGs. An example of such a DAG for both indexing and retrieval is provided in Figure 2. The different operator's overall behavior and interface is determined by vitrivr-engine's core module, which also contains the data model, interfaces for shared functionality (e.g, persistence) and some very basic implementations. The core module of vitrivr-engine can be used by developers to build their own implementations and provide them as libraries. Since such libraries can be added and removed at runtime, building and using custom operators is straightforward.

All of the remaining functionality provided by vitrivr-engine follows this same logic. For example, distinct modules are provided to support different persistence layers (PostgreSQL and Cottontail DB), types of retrievers and extractors (e.g., extractors that implement a specific algorithm or that delegate extraction to external services) and fundamental logic for retrieval and indexing (e.g., video decoders, image decoders). In addition, there are modules containing the default vitrivr-engine execution runtime and web server, the latter of which exposes all the relevant functionality via a RESTful API.

The important advantage of this modular design is its flexibility. While one user of vitrivr-engine may be interested in booting-up a server instance right away and start extraction on a media collection using the provided plugins, another may decide to use selected modules as a library instead and build their own server or even execution engine. In fact, we use both approaches in our own projects activities. For example, we use vitrivr-engine components as a plugin to power the Neural Media Repository (NMR) in the XReco project.[8]

5.3 Examples

To illustrate how the proposed model could be used to formulate specific queries, the following sections describe example-flows for both extraction and querying relevant to specific use cases.

5.3.1 Image Search. Let us consider a basic image retrieval example. Images are embedded using two different feature transformations; a multimodal image-text alignment model such as CLIP [16], and an auto-regressive unimodal image model such as DINO [15]. The former can be used for text-based image search, while the latter can be used for image-based similarity queries. Additionally, we extract and index all available EXIF metadata of the images.

A representation capable of indexing a collection of images using these features is illustrated in Figure 4. It starts with an *enumerator*, which enumerates all images of a collection. A typical enumeration process would, for example, scan a sub-tree of a file system for files of a specific type, or read content from a list of sources.

The enumerator produces a stream of *sources* which are then processed by a *decoder*, which turns them into a sequence of *retrievables*. Each of these retrievables has a unique id and holds a

[8] See https://xreco.eu/technologies/technologies-nmr

Figure 3: Example of a basic Text-based Image Retrieval Query

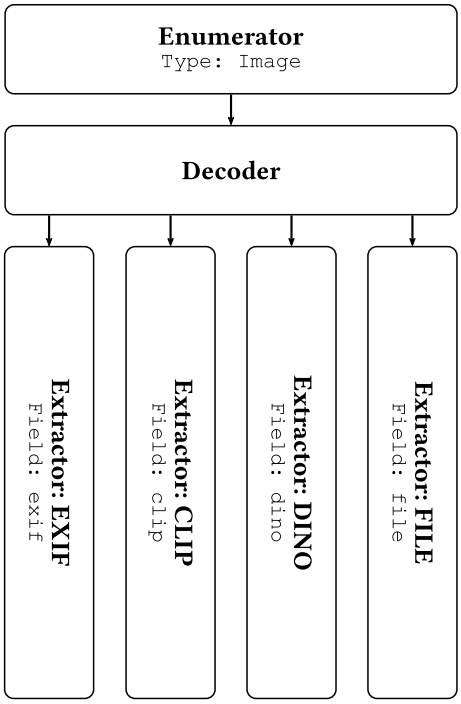

Figure 4: Example Workflow for Image Feature Extraction

reference to the original source in the form of a file or an input stream, as well as the decoded content.

These retrievables are then passed one by one into the *extractors*, which are responsible for generating one or multiple *descriptors*, i.e., the representation which is subsequently persisted. Descriptors are attached to a retrievable as it flows through the DAG. Each of these extractors has an associated *field*, which tells the persistence layer how to store the relevant information. An extractor has access to all information associated with a retrievable, including source, content, and descriptors produced upstream. Since none of the extractors in this example are dependent on one another, they can be executed in parallel.

Based on this extracted information, we can now perform several types of queries. A common use case is to find images that fit a textual description. For this, we can use the CLIP feature with a user-provided text (e.g., as entered into a search field). The relevant query DAG is a simple chain that starts with a text input which is passed to the relevant retriever. The retriever can be specified by its *field*. It produces a sequence of *retrievables* based in its internal mechanisms which are sorted by relevance to the query. In order to make the results more useful for a human user, we follow the retriever with an operator that obtains descriptors from another field for each retrievable in the sequence. In our example, we use the original file name of the input images. The final output of the DAG is a list of retrievables, each with an associated *score* attribute produced by the retriever and a *file* attribute pointing to its original location. The query representation is illustrated in Figure 3.

Next, let us assume that we want to limit the above query to images that were created on a specific day, as specified by the image's EXIF metadata. To achieve this, we can simply add a second input of type *date* to the query specification and feed it into a retriever looking at the *creation date* attribute of the *metadata* field. This retriever can be evaluated in parallel to the one already used in the previous query. The two resulting retrievable sequences can be merged using an *aggregator* performing an *intersection*, as illustrated in Figure 5.

A semantically equivalent query DAG can also be expressed as a sequential operation, as illustrated in Figure 7. Rather than using two retrievers, the creation date can be obtained for all retrievables, which can subsequently be filtered.

5.3.2 Video Search. In a video search application, the unit of retrieval is often not the entire document, but a specific temporal point or range within. This has consequences for both query formulation and feature extraction. A possible workflow to extract features from a collection of videos is illustrated in Figure 6. Analogously to the image retrieval example, the workflow starts with an enumerator, followed by a decoder. For decoders of documents with a temporal dimension, such as audio and video, we provide the option to have the decoder emit not only one retrievable with the entire decoded content attached, but to emit several retrievables with parts of the content, segmented at fixed time intervals. These temporally segmented retrievables all have a *partOf* relationship to a root retrievable which represents the entire document and holds the source property.

In our example, we set the decoder to emit one retrievable per second of video content. These retrievables are then processed by an *aggregator* that performs short segmentation. It consumes the retrievables generated by the decoder and emits new retrievables which have the same relation to the root retrievable but combine the content of several input retrievables, if they belong to the same shot.

Afterwards, these aggregated retrievables are processed by several extractors. In contrast to the image processing pipeline, here, we are using some extractors that are not independent. Specifically, the ASR feature transcribes the spoken words contained in the audio signal of each segment and attaches a text descriptor containing the transcript to the retrievable. To facilitate not only sub-string matching in these transcripts but also semantic text search, a text-embedder such as BERT [3] is used. Since this extractor is using previously generated information as an input, it needs to be downstream of the ASR extractor. The other extractors work analogously to the image example.

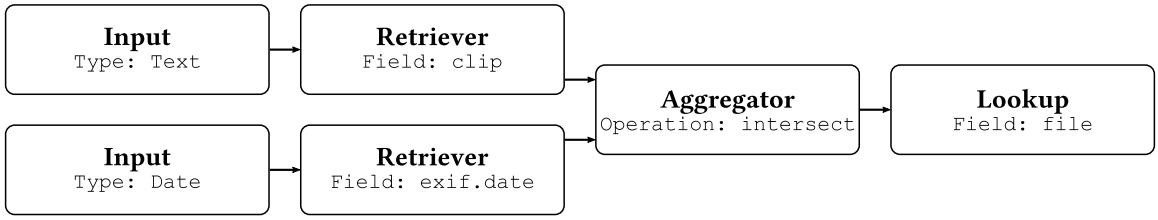

Figure 5: Example of a parallel Image Retrieval Query using multiple Inputs

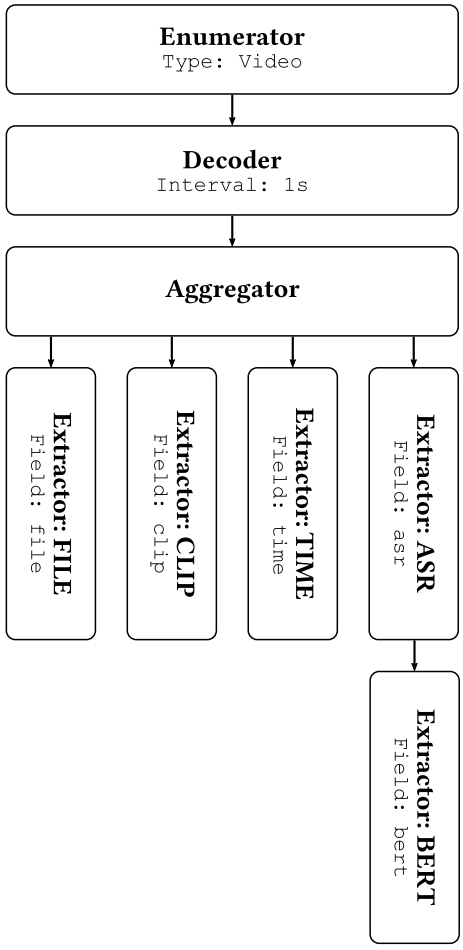

Figure 6: Example Workflow for Video Feature Extraction

Similarly to the query shown in the image retrieval example, we can query for video segments based on a textual description. The query representation is structured similarly, but requires some additional operations to deal with the fact that the retrievables described by the feature descriptors don't represent whole documents, but only parts thereof. We can, therefore, start our query in the same way, but after retrieving relevant segments, we use a *transformer* to expand the result set and obtain all retrievables that are related to the present ones using a *partOf* relation. This way, we ensure that not only the segments, but also the root retrievables representing the video documents are returned. The final result then consists not of a collection of independent retrievables, but of a collection of tree structures, associating retrieved segments with their source videos. The query representation is illustrated in Figure 8.

6 Challenges and Opportunities

Thus far, we have described our existing query model and we have demonstrated the feasibility and flexibility of our approach. We believe that even in its current state, the proposed solution opens up opportunities that go beyond what we have described in this paper. In this section, we will discuss some possible future expansion that could build upon what has been achieved.

6.1 An Algebra of Retrieval Operators

In a first step, we think it might be worthwhile to come-up with a more formal approach to the definition of the individual operators described in Section 3. Our aim here should be to build an *algebra of canonical retrieval operators*, from which all relevant types of more complex multimedia queries can be constructed.

We argue that, while of course every operator could be arbitrarily powerful, having simple operators allows for (de-)composition and reasoning about the properties of the individual operators and the resulting DAG. Both are a pre-requisite for query optimization and a declarative query language.

6.2 Optimization of Complex Queries

Currently, we implicitly considered the representation DAG of a query and the actual execution DAG to be identical. This does, however, not necessarily need to be the case. For every query graph, there can exist any number of semantically equivalent DAGs with different execution characteristics. Analogously to algebraic query optimization performed by every query planer found in a modern database management system, it is foreseeable that a set of graph transformations exist that can be applied to a query graph to increase the efficiency of its execution without changing its semantics.

Similarly, while our current implementation assumes a one-to-one correspondence between the operators (i.e., nodes in the query DAG) and the execution primitives available to the system, such a direct correspondence is not necessary. The goal of this endeavor is to define a query representation mechanism that is not specific to only one retrieval system. This will require a set of operators that is shared across all systems implementing this query representation. It does, however, not require the systems to actually implement

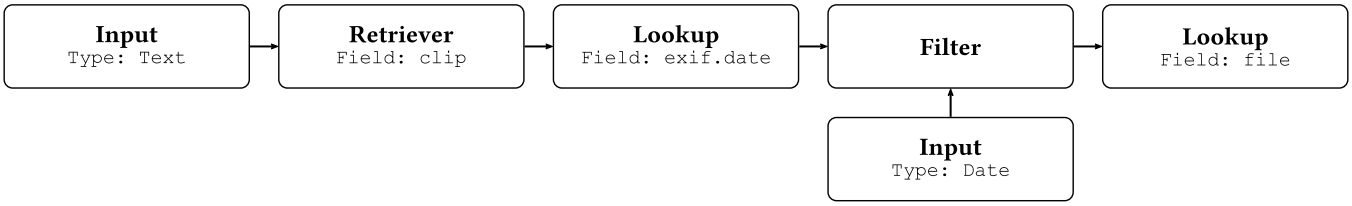

Figure 7: Example of a sequential Image Retrieval Query using multiple Inputs

Figure 8: Example of a Video Retrieval Query

these operators in the way they are described in the query specification, as long as equivalent functionality can be constructed using the available implementations. This also enables a system to optimize its execution graph, by replacing a relevant sub-graph with a single operator that is capable of performing multiple operations concurrently. An example of such a translation could be an operator that performs nearest neighbor search only on descriptors of retrievables that also fulfill a specific condition, expressed as a Boolean predicate. When using a relational vector database such as PostgreSQL with the pgVector extension, such a sub-query could be directly pushed down to the persistence layer, if the operator supports doing so. Thinking this optimization process even further, it is also thinkable to translate the query graph into a system-specific representation that can be directly compiled into machine code, similar to the procedures employed by some database management systems [11].

6.3 Declarative Query Language

So far, we have discussed our proposed query representation as an information flow graph which is, at least in principle, directly executable. The representation is therefore *imperative* in nature. This is in stark contrast to established query languages, such as SQL, SPARQL, etc., which are *declarative*, describing what the query seeks, but not how it is to be executed. One could imagine an even higher-level representation of a retrieval query, being built on top of something like our proposed approach, which does not require the explicit specification of a query execution plan. The primary challenge of defining such a declarative representation is the selection of an appropriate mental abstraction of the query. While SQL fundamentally expresses an information need as a set of Boolean conditions applied to specific columns of select tables, and SPARQL describes sub-graph patterns which are to be matched to a larger graph structure, it is currently unclear what analogous abstraction would be suitable for multimodal information retrieval. The chosen abstraction would need to be flexible enough to represent a sufficiently large number of information needs in order to be useful and choosing an insufficiently powerful abstraction can lead to limitations in query expressiveness. An example of such a limitation can be seen in SPARQL, which does not allow for certain valid graph operations, such as breath-first-search, since it cannot be expressed as a graph pattern to be matched.

Finding such an abstraction for general-purpose multimodal information retrieval is a non-trivial task. While we do not currently have a suggestion for it, a careful study of the consequences of our proposed imperative representation, especially once a minimal set of operators for query expression is established, could be an important stepping stone towards such a declarative query language.

7 Conclusion

In this paper, we presented our proposal for a universal query representation in multimedia retrieval. We demonstrated how our flow-based query representation can be used both during multimedia indexing as well as querying to represent the execution graph of the operations involved and we introduced the multimedia retrieval system vitrivr-engine, which implements the proposed model.

Based on this work, we then go on to suggest three steps towards a more universal framework for multimedia query formulation, optimization and execution:

(1) The establishment of a formal algebra of canonical retrieval operators that can be used to express complex queries.
(2) The separation of the graph of these canonical operators and the resulting execution graph with an optimization step in between, that aims to find the most efficient execution path for a given query.
(3) The conception of a declarative language, similar to SQL or SPARQL, that can be used to express multimedia queries in terms of what should be retrieved instead of how.

We argue that work in these three areas will ultimately lead us to multimedia retrieval systems that are more flexible in their use, more compatible among one another and will allow developers to focus more on what they want to achieve rather than how it should be achieved – as is the case for SQL and database development.

Acknowledgments

This work was partially supported by the Swiss National Foundation through the project 'MediaGraph' (contract no. 202125) and the Horizon Europe project XR mEdia eCOsystem (XReco), based on a grant from the Swiss State Secretariat for Education, Research and Innovation (contract no. 22.00268).

References

[1] Donald D. Chamberlin. 2024. 50 Years of Queries. *Commun. ACM* 67, 8 (2024), 110–121. doi:10.1145/3649887

[2] WB Croft. 1983. Document Retrieval System. *Information Technology* 2 (1983), 1–21.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. doi:10.18653/V1/N19-1423

[4] John D. N. Dionisio and Alfonso F. Cardenas. 1996. MQuery: A Visual Query Language for Multimedia, Timeline and Simulation Data. *Journal of Visual Languages & Computing* 7, 4 (1996), 377–401. doi:10.1006/JVLC.1996.0020

[5] Mario Döller, Ruben Tous, Matthias Gruhne, Kyoungro Yoon, Masanori Sano, and Ian S Burnett. 2008. The MPEG Query Format: Unifying access to multimedia retrieval systems. *IEEE MultiMedia* 15, 04 (2008), 82–95.

[6] Ralph Gasser, Rahel Arnold, Fynn Faber, Heiko Schuldt, Raphael Waltenspül, and Luca Rossetto. 2024. A New Retrieval Engine for Vitrivr. In *MultiMedia Modeling - 30th International Conference, MMM 2024, Amsterdam, The Netherlands, January 29 - February 2, 2024, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 14557)*. Springer, 324–331. doi:10.1007/978-3-031-53302-0_28

[7] Ralph Gasser, Luca Rossetto, Silvan Heller, and Heiko Schuldt. 2020. Cottontail DB: An Open Source Database System for Multimedia Retrieval and Analysis. In *MM '20: The 28th ACM International Conference on Multimedia, Virtual Event / Seattle, WA, USA, October 12-16, 2020*. ACM, 4465–4468. doi:10.1145/3394171.3414538

[8] Ralph Gasser, Luca Rossetto, and Heiko Schuldt. 2019. Multimodal Multimedia Retrieval with vitrivr. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*. 391–394.

[9] Nael Hirzalla and Ahmed Karmouch. 1995. A multimedia query specification language. In *Proceedings. International Workshop on Multi-Media Database Management Systems*. IEEE, 66–73.

[10] Shu-Chen Kau and JCR Tseng. 1994. MQL – A Query Language for Multimedia Database. In *5th IEEE COMSOC International Workshop on Multimedia Communications*. IEEE, 5–1.

[11] Konstantinos Krikellas, Stratis Viglas, and Marcelo Cintra. 2010. Generating code for holistic query evaluation. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*. IEEE Computer Society, 613–624. doi:10.1109/ICDE.2010.5447892

[12] Thomas Kurz, Sebastian Schaffert, Kai Schlegel, Florian Stegmaier, and Harald Kosch. 2014. SPARQL-MM - Extending SPARQL to Media Fragments. In *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 8798)*. Springer, 236–240. doi:10.1007/978-3-319-11955-7_26

[13] John Z Li, M Tamer Ozsu, Duane Szafron, and Vincent Oria. 1997. MOQL: A Multimedia Object Query Language. In *Proceedings of the 3rd International Workshop on Multimedia Information Systems*. Como Italy, 19–28.

[14] Carlo Meghini, Fabrizio Sebastiani, and Umberto Straccia. 2001. A Model of Multimedia Information Retrieval. *Journal of the ACM (JACM)* 48, 5 (2001), 909–970. doi:10.1145/502102.502103

[15] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. 2024. DINOv2: Learning Robust Visual Features without Supervision. *Transactions on Machine Learning Research Journal* 2024 (2024). doi:10.48550/arxiv.2304.07193

[16] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 8748–8763. http://proceedings.mlr.press/v139/radford21a.html

[17] Luca Rossetto and Ralph Gasser. 2025. Feature-Driven Video Segmentation and Advanced Querying with vitrivr-Engine. In *MultiMedia Modeling - 31st International Conference on Multimedia Modeling, MMM 2025, Nara, Japan, January 8-10, 2025, Proceedings, Part V (Lecture Notes in Computer Science, Vol. 15524)*. Springer, 272–277. doi:10.1007/978-981-96-2074-6_32

[18] Luca Rossetto, Ralph Gasser, Silvan Heller, Mahnaz Amiri Parian, and Heiko Schuldt. 2019. Retrieval of structured and unstructured data with vitrivr. In *Proceedings of the ACM Workshop on Lifelog Search Challenge*. 27–31.

[19] Luca Rossetto, Ivan Giangreco, Ralph Gasser, and Heiko Schuldt. 2018. Competitive Video Retrieval with vitrivr. In *MultiMedia Modeling – 24th International Conference, MMM 2018, Bangkok, Thailand, February 5-7, 2018, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10705)*. Springer, 403–406. doi:10.1007/978-3-319-73600-6_41

[20] Luca Rossetto, Ivan Giangreco, and Heiko Schuldt. 2014. Cineast: A Multi-feature Sketch-Based Video Retrieval Engine. In *2014 IEEE International Symposium on Multimedia, ISM 2014, Taichung, Taiwan, December 10-12, 2014*. IEEE Computer Society, 18–23. doi:10.1109/ISM.2014.38

[21] Luca Rossetto, Ivan Giangreco, Claudiu Tanase, and Heiko Schuldt. 2016. vitrivr: A Flexible Retrieval Stack Supporting Multiple Query Modes for Searching in Multimedia Collections. In *ACM Conference on Multimedia*. doi:10.1145/2964284.2973797

[22] Loris Sauter, Ralph Gasser, Laura Rettig, Heiko Schuldt, and Luca Rossetto. 2024. General Purpose Multimedia Retrieval with vitrivr at LSC'24. In *Proceedings of the 7th Annual ACM Workshop on the Lifelog Search Challenge* (Phuket, Thailand) *(LSC '24)*. Association for Computing Machinery, New York, NY, USA, 47–52. doi:10.1145/3643489.3661120

[23] Loris Sauter, Heiko Schuldt, Raphael Waltenspül, and Luca Rossetto. 2023. Novice-Friendly Text-based Video Search with vitrivr. In *20th International Conference on Content-based Multimedia Indexing, CBMI 2023, Orleans, France, September 20-22, 2023*. ACM, 163–167. doi:10.1145/3617233.3617262

[24] Florian Spiess, Ralph Gasser, Silvan Heller, Heiko Schuldt, and Luca Rossetto. 2023. A Comparison of Video Browsing Performance between Desktop and Virtual Reality Interfaces. In *Proceedings of the 2023 ACM International Conference on Multimedia Retrieval, ICMR 2023, Thessaloniki, Greece, June 12-15, 2023*. ACM, 535–539. doi:10.1145/3591106.3592292

[25] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. PASE: Postgresql ultra-high-dimensional approximate nearest neighbor search extension. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2241–2253.

[26] Jan Zahálka and Marcel Worring. 2014. Towards interactive, intelligent, and integrated multimedia analytics. In *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 3–12.