A SPARQL in the Dark: Shining a Light on Multimodal Lifelogs with LifeGraph 5

Florian Ruosch
Zurich, Switzerland
ruosch@ifi.uzh.ch

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

Abstract

For the 8th *Lifelog Search Challenge* (*LSC'25*), we develop *LifeGraph 5*, the most recent version of a multimodal knowledge graph for lifelog retrieval. Borrowing from *LifeGraph*'s previous iterations, we push for enhanced capabilities in our knowledge graph and expand the lifelog query functionalities. As such, we extend SPARQL with concepts like similarity and implicit and derived relations. Furthermore, we design a new user interface for the query builder.

CCS Concepts

• **Information systems** → *Users and interactive retrieval*; *Specialized information retrieval*; **Multimedia and multimodal retrieval**.

Keywords

Lifelogging, Lifelog Search Challenge, Multimodal Knowledge Graphs, Graph-based Retrieval, Multimodal Retrieval

ACM Reference Format:
Florian Ruosch and Luca Rossetto. 2025. A SPARQL in the Dark: Shining a Light on Multimodal Lifelogs with LifeGraph 5. In *Proceedings of the 8th Annual ACM Workshop on the Lifelog Search Challenge (LSC '25), June 30-July 3, 2025, Chicago, IL, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3729459.3748694

1 Introduction

Lifelogs are naturally multimodal document sets that represent a variety of information. In the case of the *Lifelog Search Challenge* (*LSC*) [11], the data consists of first-person perspective images taken by a wearable camera and enriched with some metadata such as geotag, date, or time. Such data is well-suited to be represented in a knowledge graph that contains multiple modalities. Furthermore, the information can be linked to external data sources, and these relationships are leveraged for the organization of the data. With *LifeGraph 5*, we build the most recent instance of such a multimodal knowledge graph for participation in *LSC'25* [5].

Given that the challenge dataset has not changed since *LSC'22* [3, 4], we mainly build upon the earlier versions of our graph-based approach *LifeGraph* [7–10]. In last year's *LifeGraph 4*, we concentrated our efforts on enriching the knowledge graph with information generated by state-of-the-art Vision Language Models. We reuse the current knowledge graph, but we are now focusing on enhancing the query capabilities and user interaction. However, we also extend the existing multimodal knowledge graph with non-materialized relations, which can be implicit or derived.

The remainder of this paper is structured as follows: Section 2 describes *LifeGraph*'s previous versions. Section 3 then gives an overview of the knowledge graph structure and the system architecture. In Section 4, we explain the improvements for the user interface before Section 5 presents the results and Section 6 the deduced Limitations. Finally, we conclude the paper in Section 7.

2 A Brief History of LifeGraph

With *LifeGraph* [7], we proposed what would be the first iteration of a multimodal knowledge graph for lifelogs. It was designed to capture and structure lifelogging data with diverse modalities while linking to external, static knowledge bases like Wikidata.[1] We also integrated COEL (Classification of Everyday Living) [1], a taxonomy with over 5,000 elements for events, to categorize the lifelog data, enabling better organization and, hence, retrieval. *LifeGraph* represented the first approach leveraging knowledge graphs and semantic web technologies in a field dominated by other methods such as video retrieval and user interaction strategies. Our novel idea let us represent concepts or tags as nodes in the graph and then formulate queries in SPARQL for the retrieval of the connected relevant documents based on pattern matching. However, *LifeGraph* also exhibited severe shortcomings.

These were addressed with *LifeGraph 2* [8]. We moved from Blazegraph DB, a triple store, to Cottontail DB [2] to enhance the graph traversal mechanisms and improve query performance and user experience. Furthermore, we switched from centering the graph around COEL to relying directly on Wikidata classes. The addition of new graph distance measures and embeddings further enriched query formulation and search results.

We focused on hierarchical clustering for *LifeGraph 3* [9]. By leveraging clustering strategies focussed on temporal, spatial, and visual dimensions of the lifelogs, we built semantically rich clusters of the multimodal knowledge graph nodes. This enabled top-down and bottom-up queries, facilitating the interaction with the knowledge graph and enhancing retrieval capabilities. Likewise, the inclusion of the metadata and the robust graph structure allowed for efficient exploration and filtering of the lifelogs, leading to an improved user experience.

The most recent iteration, *LifeGraph 4* [10], again focused on clustering but this time emphasized event nodes. It also leveraged Vision Language Models to augment the existing data by exploiting generated image captions. Moreover, we improved the presentation

[1] https://www.wikidata.org

ACM ISBN 979-8-4007-1857-1/2025/06
https://doi.org/10.1145/3729459.3748694

of the results and added query functionalities, mainly built around semantic embeddings and vector space similarity. This enabled effective search capabilities for concepts not explicitly annotated. Ultimately, we demonstrated improvements in the retrieval performance of known-item searches and ad-hoc queries.

3 System Architecture

In this section, we lay out the system architecture of the *LifeGraph 5* system, starting with its namesake multimodal knowledge graph at its heart. Then, we describe the extensions to the querying for non-materialized relations and embedding-based similarity.

3.1 LifeGraph

LifeGraph is a sophisticated multimodal knowledge graph for the retrieval of lifelogging data. It organizes its data around images but also integrates other modalities, such as text and metadata. Each image in the graph is linked to associated features and, thus, results in a semantically rich network of relationships.

Just like the past few iterations, we use our custom *MediaGraph Store*, or *MeGraS*[2] for short, to power the graph and its querying. Written in Kotlin, it provides the backend to store and query multimodal knowledge graphs, with several storage options available for flexibility. The graph may be held in memory for smaller graphs and testing, with the triples being dumped periodically to a single file. *MeGraS* also leverages Cottontail DB [2] for all data types, which includes vectors, suited for graphs with the number of nodes in the six- or seven-digit range. Finally, for larger graphs, PostgresSQL[3] is available.

For interaction with the multimodal knowledge graph, *MeGraS* provides a built-in command line interface to enable basic data management tasks such as importing graph triples or media files as nodes. Furthermore, a RESTful API in the OpenAPI specification is available for graph manipulation and querying.

In contrast to previous iterations, we forego the use of the dedicated API in favor of a single SPARQL query endpoint, similar to the approach in the first version of LifeGraph [7]. The primary difference between this initial and the current version is that the underlying graph store is our custom implementation in the form of *MeGraS*. We, therefore, add the relevant extensions directly to the SPARQL runtime, as outlined in the following sections.

For the graphs structure, we rely on and build upon the status quo from our submission to last year's *LSC*, *LifeGraph 4* [10]. With it, we enable the clustering of similar images based on diverse dimensions and with varying degrees of resolution. For example, data points may be grouped by temporal (e.g., days, months, years) or spatial (e.g., cities, countries, continents) features or by similarity (e.g., semantical, visual, embedding). The intersection of such filters allows for the efficient retrieval of specific images: the ultimate goal of the *Lifelog Search Challenge*.

We abstain from pruning the graph as we did in previous years and, instead, leverage the full amount of information available, not only of the original dataset but also what we produced in the earlier iterations of *LifeGraph*. Rather, we focus on refining retrieval by enhancing and extending the query functionalities.

[2]https://github.com/lucaro/MeGraS
[3]https://www.postgresql.org

3.2 Graph Pattern Matching for Non-Materialized Relations

The main addition to the querying capabilities of *LifeGraph* comes in the form of pattern matching relations that are not precomputed or stored (i.e., materialized). As shown in Table 1, we distinguish three types of such relations: inferred, implicit, and derived. Inferred relations are logically entailed by the data, exist in theory, and can be derived by a reasoner. Meanwhile, implicit relations are defined as between two graph nodes. Finally, derived relations result from predefined functions that can be evaluated at query time.

3.2.1 Inferred Relations. Inferred relations are not necessarily present in the data but can be inferred with rules, as the name suggests. Hence, they are entailed by the semantics of the RDF data.

Simple rules, such as a calendar ontology distinguishing weekdays, can be used to simplify queries by foregoing having to include the whole list of workdays. Hence, if we want to find all images from certain days of the week, we can use the query shown in Listing 1, which leverages our custom SPARQL function `megras:DAYOFWEEK`.

Listing 1: Detecting the inferred relation images taken on Mondays (which corresponds to"1").

```
SELECT ?img
WHERE {
    ?img lsc:day ?day .
    BIND(xsd:date(STRAFTER(STR(?day), "#")) AS ?dayDate)
    FILTER (megras:DAYOFWEEK(?dayDate) IN (1))
}
```

3.2.2 Implicit Relations. We define implicit relations to only occur between two nodes in the graph, e.g., between life-log entries in the form of images. Listing 2 shows the first example of an implicit relation, whereby the near duplicates of a given image are retrieved based on a specific relation, the similarity of the CLIP embeddings, in this case. The relation `implicit:clipNearDuplicate` is not materialized in the graph, and a regular query containing this relation would not return any results. However, MeGraS has a *handler* registered for this relation, which is called at query time to produce the relevant graph triples without them ever being part of any persistent representation of the graph explicitly.

Listing 2: Detecting the implicit relation of near duplicates based on visual features.

```
SELECT ?img
WHERE {
  lsc:img implicit:clipNearDuplicate ?img .
}
```

Similarly, the k-nearest neighbors (KNN) can be retrieved for a given node in the graph, based on their CLIP embeddings. Contrary to a conventional SPARQL approach where a LIMIT would have to be set on the query, we instead push this functionality down into our backend and only use the SPARQL syntax to have it execute the corresponding function using another *handler*. The KNN relation, opposed to the near duplicate, is not a fixed-name relation but

Table 1: Overview for the three different relation types we consider.

Relation Type	Definition	Examples
Inferred	logically entailed by the data, can be derived by a reasoner	workday, subclasses of a tag
Implicit	between two graph nodes	find near duplicates, after
Derived	results of predefined functions evaluated at query time	OCR in image, embedding vectors

rather matched with a regular expression to allow for an arbitrary k in the function. Hence, a query to find the 16-nearest neighbor of a given image would look like the one shown in Listing 3.

Listing 3: Detecting the implicit relation of k-nearest neighbors.

```
SELECT ?img
WHERE {
  lsc:img implicit:clip16nn ?img .
}
```

3.2.3 Derived Relations. Similarly to implicit relations, derived relations have associated function handlers that can be called at query time. In contrast to implicit relations, derived relations are always between a graph node and a literal, whereas implicit relations are defined as only existing between two graph nodes. Also, derived relations are materialized upon evaluation of the handler to avoid repeated computation. Handlers are only evaluated when the corresponding triple is not already part of the graph.

Derived relations allow for more complex operations, such as shown in Listing 4. The relation `derived:ocr` is associated with a handler that performs Optical Character Recognition (OCR) on an image and returns all extracted text. Since the evaluation of the text extraction can be performed at query time, we also include images for which no extracted text exists ex ante. Compared to previous iterations, where we only filtered on the previously available OCR text, this is a novel approach.

Listing 4: Detecting the derived relation of (not necessarily previously extracted) OCR text in the image with the example of a restaurant bill.

```
SELECT ?img
WHERE {
  ?img derived:ocr ?extractedText.
  FILTER (CONTAINS(?extractedText, "Guest Bill"))
}
```

3.2.4 Custom Functions and Filters. The various types of relations described above can also be combined with arbitrary custom functions, substantially expanding the expressiveness of queries. Listing 5 illustrates, for example, how two custom functions – `megras:CLIP_TEXT` and `megras:COSINE_SIM` – can be used to express a nearest neighbor similarity query within SPARQL. Since `derived:clipEmbedding` is a derived relation, this query can even be executed without any prior feature extraction, although that would negatively impact the query time.

Listing 5: Finding the five images for which the CLIP [6] embeddings are the most similar to the embedding of the textual description.

```
SELECT ?img
WHERE {
  BIND (megras:CLIP_TEXT("description") as ?textVec)
  ?img derived:clipEmbedding ?clipVec .
  BIND (megras:COSINE_SIM(?textVec, ?clipVec) as ?cosSim)
}
ORDER BY DESC(?cosSim)
LIMIT 5
```

4 User Interaction

We also improve and build up the way users interact with the graph. By switching from a general API to a SPARQL query endpoint, we reduce to a single point of contact, and all interaction happens through SPARQL queries. This also simplifies the user interface to only two main frames: the query builder and the results display. While keeping the latter largely the same, the new paradigm for the interaction with the knowledge graph necessitates a significant overhaul for how the criteria for retrieving lifelog events are created. A screenshot of the user interface is shown in Figure 1.

The new query builder itself is split into two parts: the creator and the corresponding generated SPARQL code. The creator relies on various methods to build the query, which is then displayed side by side with it, ready for execution with a "Query" button to submit it. The whole process of retrieval is, hence, based on a user feedback loop and iterations to interactively refine the query.

The creation of the SPARQL code rests on predefined templates that can be completed based on user inputs. It can be viewed as a fill-in-the-blanks text, where the user sets the desired attributes. Different methods for their capture are employed depending on the properties to be queried for. Multiple such criteria can be combined into a single query.

A configurable, collapsible sidebar provides selectors for tags, categories, countries, cities, locations, dates (year, month, weekday), and times. Similarly, filters for the captions, OCR, and CLIP similarity are available. Each filter supports multi-select, search, and reset/clear actions.

Results are shown in a responsive grid, grouped by day if selected. Clicking an image opens an overlay with navigation and context action to view temporally neighboring images in a dedicated overlay, with navigation and submission options. Advanced filtering can also be done with the KNN and near-duplicate search for a given image.

Finally, we also provide local logging, whereby queries and their results are saved locally and can be viewed, downloaded, or cleared

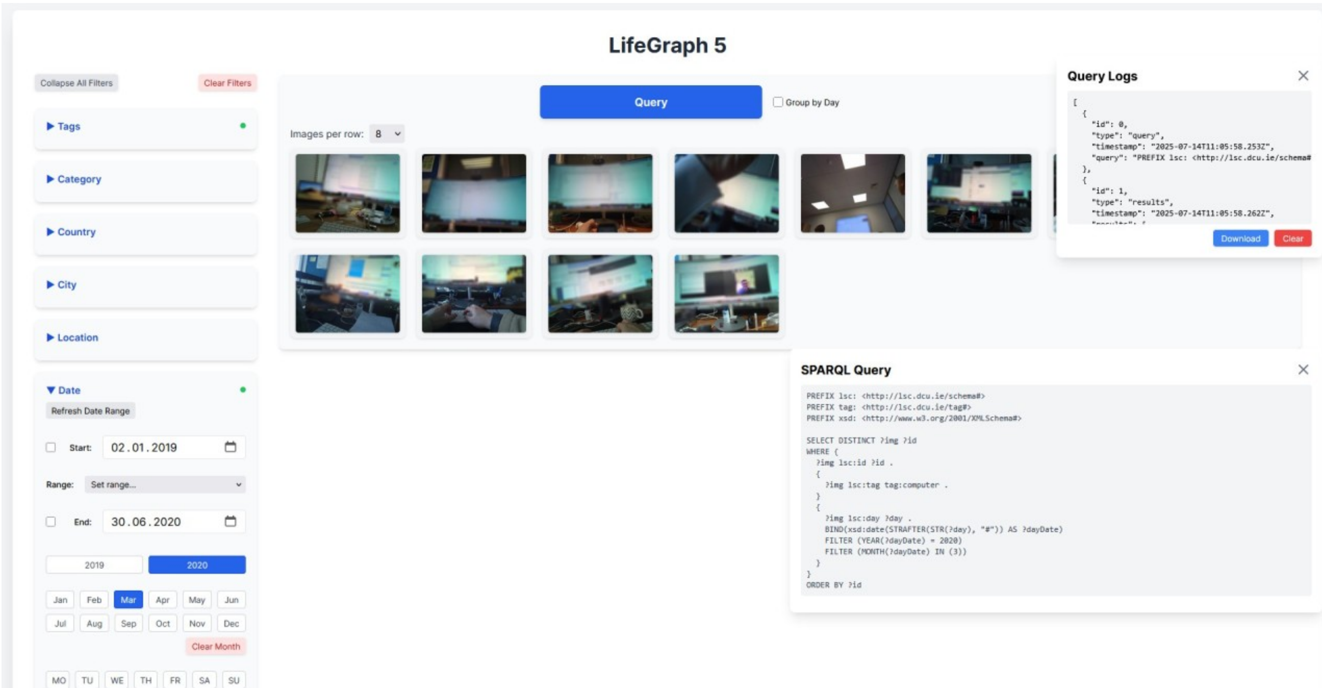

Figure 1: A screenshot of the *LifeGraph 5* user interface in action.

via a floating button. The overall focus is on a user-friendly, highly interactive search and filtering interface for image data, with real-time query feedback and advanced exploration features.

5 Results

Figure 2 shows the results for the two operators of the *LifeGraph 5* system in chronological order per task type: QA (question answering), ADHOC (ad-hoc event search), and KIS (known-item search). The system's performance profile, and consequently, the scores, varied substantially for the task type as well as between the two users.

While LifeGraph1 was in the middle of the pack after the QA tasks, it exhibited a sharp fall-off for the two subsequent task types. This was due to technical limitations, as the query execution time slowed down considerably, mostly not returning results within the allotted five-minute time frame per task. More details are explained in the subsequent Section 6.

LifeGraph2 had to deal with similar issues but managed to pre-empt them substantially better and showed its best score in the KIS tasks. Combined with a solid score from the QA and some points from the ADHOC block, it combines for the better result of the two operators and, thus, will represent *LifeGraph 5* in the final standings. However, these results are not competitive when compared to the top systems.

6 Limitations

The execution time for the queries suffered significantly from the way Jena's SPARQL engine interacted with the multimodal knowledge graph stored in MeGraS. This was mainly due to two reasons: the behavior of Apache Jena's query engine and the triple management of the chosen persistence store. While we managed to leverage the advantages of SPARQL and Apache Jena's query engine, the latter also severely inhibited the query execution time in two ways:

First, the way Jena interacted with the knowledge graph was by handing it individual triples for pattern matching instead of sets, which necessitated a large number of separate queries for their resolution. This limitation stems from the way Jena allows for the object store to be wrapped. To address this, the implementation of a custom query engine base is necessary, which was out of scope for this year's challenge.

Second, Jena forced repeated execution for custom functions with a static value, even if they were used in a `BIND` statement. This resulted, again, in an unnecessarily large number of queries to the underlying store of the knowledge graph. While adding a custom cache to these functions showed a speedup, the sheer amount of queries slowed down the entire system.

The final limitation of MeGraS was the chosen persistence store for the knowledge graph: Postgres. While we expected it to perform well when it came to the execution time of SQL queries, the nature of the resulting queries slowed it down. We addressed this by loading the entirety of the knowledge graph into memory, resulting in a startup time of 40 to 60 minutes. While this made a reboot during the challenge impossible, it shortened query execution into a time frame that would make the system usable. Also, this is not an ideal long-term solution, and efforts have to be invested in optimizing the lower-layer triple management in the persistence store, especially Postgres.

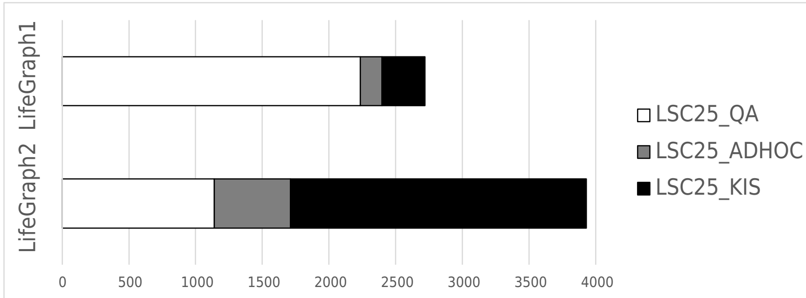

Figure 2: Aggregated performance scores per task type for the two operators of *LifeGraph 5*. The length of each colored segment indicates the sum of challenge scores obtained by each task type.

7 Conclusion

This paper presented an overview of our retrieval approach for the 2025 Lifelog Search Challenge. Based on our previous iterations of *LifeGraph* [7–10], we continue the path of a multimodal knowledge graph to represent lifelog data. We extend the query capabilities, allowing for additional patterns to be matched. Furthermore, we improve our user interface to interact with the knowledge graph through SPARQL queries.

LifeGraph 5 goes back to interaction patterns of the initial *LifeGraph* by exclusively relying on SPARQL queries. To overcome the limitations of that initial approach, we extend the capabilities of our custom multimodal graph storage engine *MeGras* to increase the expressiveness of SPARQL queries in the context of retrieval in multimodal graphs.

While *LifeGraph 5* was not able to compete at the top of the challenge, we nevertheless showed the feasibility of the knowledge-graph-based approach to exploring lifelogging data by successfully completing 9 out of the 18 tasks across the two participants, with at least two tasks per type. The main limitations were of a technical nature, which will necessitate investigation and improvements in the complex backend solution we employed, such as implementing a custom SPARQL query engine and adapting management of the persisted data.

Acknowledgments

This work was partially funded by the Swiss National Science Foundation through Project "MediaGraph" (Grant Number 202125).

References

[1] Paul Bruton, Joss Langford, Matthew Reed, and David Snelling. 2019. Classification of Everyday Living Version 1.0. https://docs.oasis-open.org/coel/COEL/v1.0/os/COEL-v1.0-os.html Last updated 23 January 2019.

[2] 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*, Chang Wen Chen, Rita Cucchiara, Xian-Sheng Hua, Guo-Jun Qi, Elisa Ricci, Zhengyou Zhang, and Roger Zimmermann (Eds.). ACM, 4465–4468. doi:10.1145/3394171.3414538

[3] Cathal Gurrin, Björn Þór Jónsson, Duc-Tien Dang-Nguyen, Graham Healy, Jakub Lokoc, Liting Zhou, Luca Rossetto, Minh-Triet Tran, Wolfgang Hürst, Werner Bailer, and Klaus Schoeffmann. 2023. Introduction to the Sixth Annual Lifelog Search Challenge, LSC'23. In *Proceedings of the 2023 ACM International Conference on Multimedia Retrieval, ICMR 2023, Thessaloniki, Greece, June 12-15, 2023*. ACM, 678–679. doi:10.1145/3591106.3592304

[4] Cathal Gurrin, Liting Zhou, Graham Healy, Werner Bailer, Duc-Tien Dang-Nguyen, Steve Hodges, Björn Þór, Jakub Lokoc, Luca Rossetto, Minh-Triet Tran, and Klaus Schoeffmann. 2024. Introduction to the Seventh Annual Lifelog Search Challenge, LSC'24. In *Proceedings of the 2024 International Conference on Multimedia Retrieval* (Phuket, Thailand) *(ICMR '24)*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3652583.3658891

[5] Cathal Gurrin, Liting Zhou, Graham Healy, Allie Tran, Luca Rossetto, Werner Bailer, Duc-Tien Dang-Nguyen, Steve Hodges, Björn Þór Jónsson, Minh-Triet Tran, and Klaus Schöffmann. 2025. Introduction to the 8th Annual Lifelog Search Challenge, LSC'25. In *Proceedings of the 2025 International Conference on Multimedia Retrieval* (Chicago, IL, USA) *(ICMR '25)*. Association for Computing Machinery, New York, NY, USA, 2143–2144. doi:10.1145/3731715.3734579

[6] 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)*, Marina Meila and Tong Zhang (Eds.). PMLR, 8748–8763. http://proceedings.mlr.press/v139/radford21a.html

[7] Luca Rossetto, Matthias Baumgartner, Narges Ashena, Florian Ruosch, Romana Pernischová, and Abraham Bernstein. 2020. LifeGraph: A Knowledge Graph for Lifelogs. In *Proceedings of the Third Annual Workshop on Lifelog Search Challenge* (Dublin, Ireland) *(LSC '20)*. Association for Computing Machinery, New York, NY, USA, 13–17. doi:10.1145/3379172.3391717

[8] Luca Rossetto, Matthias Baumgartner, Ralph Gasser, Lucien Heitz, Ruijie Wang, and Abraham Bernstein. 2021. Exploring Graph-querying approaches in LifeGraph. In *Proceedings of the 4th Annual on Lifelog Search Challenge* (Taipei, Taiwan) *(LSC '21)*. Association for Computing Machinery, New York, NY, USA, 7–10. doi:10.1145/3463948.3469068

[9] Luca Rossetto, Oana Inel, Svenja Lange, Florian Ruosch, Ruijie Wang, and Abraham Bernstein. 2023. Multi-Mode Clustering for Graph-Based Lifelog Retrieval. In *Proceedings of the 6th Annual ACM Lifelog Search Challenge, LSC 2023, Thessaloniki, Greece, June 12-15, 2023*. ACM, 36–40. doi:10.1145/3592573.3593102

[10] Luca Rossetto, Athina Kyriakou, Svenja Lange, Florian Ruosch, Ruijie Wang, Kathrin Wardatzky, and Abraham Bernstein. 2024. LifeGraph 4 - Lifelog Retrieval using Multimodal Knowledge Graphs and Vision-Language Models. 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, 88–92. doi:10.1145/3643489.3661127

[11] Allie Tran, Werner Bailer, Duc-Tien Dang-Nguyen, Graham Healy, Steve Hodges, Björn Þór Jónsson, Luca Rossetto, Klaus Schoeffmann, Minh-Triet Tran, Lucia Vadicamo, and Cathal Gurrin. 2025. The State-of-the-Art in Lifelog Retrieval: A Review of Progress at the ACM Lifelog Search Challenge Workshop 2022-24. *CoRR* abs/2506.06743 (2025). doi:10.48550/ARXIV.2506.06743 arXiv:2506.06743