Querying LLMs as if they were Digital Libraries

Mirco Cazzaro[1,*], Gianmaria Silvello[1]

[1]*Department of Information Engineering, University of Padova, Via Gradenigo 6/B, Padova, 35131, Italy*

Abstract

LLMs are increasingly considered as Knowledge Bases (KBs), since they encode vast amounts of factual information that could, in principle, be queried as if they were a Digital Library (DL). In this work we focus on the Cultural Heritage (CH) domain, addressing the following question: to what extent can Large Language Models (LLMs) act as a KB, and which query paradigms are better suited for the task?

In this paper, we propose a first case study on CH data using “Galois”, a recent framework for executing Structured Query Language (SQL) queries over LLMs with logical and physical optimizations tailored to the model’s behavior. We build a new benchmark grounded in the *Famous Paintings* dataset from Kaggle, a tabular collection of paintings and their artists. From this data we derive a set of information needs that reflect typical CH scenarios. For each information need we define a reference SQL query and evaluate three ways of querying the LLM: direct Natural Language (NL) questions, direct SQL prompting, and SQL execution through Galois. Our study provides an initial assessment on the feasibility of using LLMs as a KB for CH.

Keywords

Large Language Models, Query Optimization, Cultural Heritage

1. Introduction

DL institutions are increasingly deploying LLMs as an additional access layer to CH data and collections [1]. Beyond their use as conversational interfaces and reasoning agents, LLMs implicitly store large amounts of factual knowledge acquired during training. This raises a central question for the DL community: can an LLM act as a KB for querying CH data?

This paper explores that question starting from a simple intuition. If an LLM “knows” about a sufficiently large set of entities in a given domain, then it should, in principle, be possible to query this knowledge and guide the model towards answers that approach the completeness and correctness of structured queries over databases. In such a scenario, curators and researchers would use LLMs not only for explanations or summaries, but also for precise answers grounded in curated collections. We therefore ask: (i) to what extent can LLMs be used as a KB for CH data when evaluated against a curated DL, and (ii) how should they be queried to address typical CH information needs?

Several ways of using LLMs as a KB have been recently explored in the literature. A first family of approaches relies on direct natural language questions: the model is prompted with a carefully phrased query and asked to return knowledge following a specific schema. This is attractive because it requires no formal language, but it often suffers from hallucinations, incompleteness, and inconsistent formatting. A second family of approaches involves prompting the model with structured queries (e.g. SQL). This has the advantage of clear semantics and inherently provides a well-defined structure, but still relies on the capabilities of the model to provide complete and accurate results with respect to the query constraints. A third option is to iteratively query the LLM itself using SQL-like prompts, that seek a specific section of the expected dataset, expanding individual records independently, and improving accuracy by applying offline procedures that enforce query constrains. In fact, early experiments show that the way queries are phrased and decomposed has a strong impact on the quality of the answers [2]. This has motivated a line of work at the intersection of data management and LLMs, where systems are

IRCDL 2026: 22nd Conference on Information and Research Science Connecting to Digital and Library Science, February 19–20, 2026, Modena, Italy

*Corresponding author.

mirco.cazzaro@phd.unipd.it (M. Cazzaro); gianmaria.silvello@unipd.it (G. Silvello)
https://www.dei.unipd.it/~cazzaromir/ (M. Cazzaro); https://www.dei.unipd.it/~silvello/ (G. Silvello)
0009-0006-3856-7207 (M. Cazzaro); 0000-0003-4970-4554 (G. Silvello)

designed to mediate between SQL queries and the model, attempting to recover some of the discipline of relational execution while still relying on the LLM as the factual source.

Among these systems, Galois [3] has been proposed as a Database Management System (DBMS)-like framework for executing structured queries over an LLM. Rather than sending an entire query as a single prompt, Galois parses the SQL, constructs a logical plan, and instantiates it with physical operators that are either classical in-memory operators (such as joins and aggregations) or new, LLM-specific scan operators. The LLM is invoked only through these scans, which are responsible for generating candidate tuples and, when necessary, filtering or extending them. Different plans, corresponding to different ways of decomposing the original query, can be considered using simple cost models and confidence estimates derived from the model's own output. In this way, Galois treats the LLM as a noisy storage layer, and tries to organize interaction with it in a more principled and efficient manner.

Building on this idea, we ask a more specific question that has not yet been addressed in the DL realm: can a system like Galois answer structured cultural-heritage information needs by relying solely on the LLM's internal knowledge as its underlying "data catalog"?

To investigate this question, we probe the LLM's internal knowledge using a curated tabular dataset of paintings as reference. We measure to what extent Galois can reconstruct the ground-truth answers by querying the model alone, and compare its behavior with simpler baselines: direct natural language questions and direct SQL prompting without an execution engine in between. Our use case is grounded in a concrete and accessible collection that lists well-known paintings, their artists, and the museums that hold them, originally released as an exercise dataset for practicing SQL. We reinterpret this dataset as a proxy museum catalogue and derive from it information needs that mirror typical questions in cultural-heritage practice, such as retrieving all works by a given artist within a given period, identifying paintings that share stylistic or thematic characteristics, or listing museums in specific cities or countries. Each information need is specified both as a natural language query and as an SQL query, and the result of executing the SQL directly over the dataset serves as our ground truth.

Within this setting, we consider the Galois reference implementation as our target for reproducibility, without modifying its logic. Our focus is understanding how an existing SQL-over-LLM framework behaves when applied to CH data. We compare three querying strategies over the same set of information needs: direct natural language questions to the model, direct SQL prompting where the model is asked to "execute" the query, and full SQL execution through Galois. For each strategy we analyze the quality of the resulting tables, the kinds of errors that occur, and the cost in terms of interaction length and time complexity.

The contribution of this paper is twofold. Conceptually, (i) it frames the problem of "LLMs as KBs for CH" in terms of structured querying and positions different interaction paradigms along a spectrum from free-form question answering to fully managed SQL execution. Empirically, (ii) it introduces a small but focused benchmark based on famous paintings and uses it to compare natural language, SQL prompting, and Galois.

The paper is organized as follows: after reviewing related work on LLMs as KBs, on their use in DLs and CH and on SQL-over-LLM frameworks, we summarize the main ideas behind Galois. We then present our benchmark, describing the dataset, the derived information needs, and the query statistics. The experimental section details the evaluation protocol and reports the comparative results for the three querying strategies. Finally, we discuss what these findings suggest about the feasibility and limitations of using Galois as a structured access layer over LLMs for CH data, and we outline directions for future improvements.

2. Related Work

Our work lies at the intersection of three strands of research: fact-centric views of LLMs as implicit KBs, the use of LLMs as automatic judges or evaluators, and recent explorations of LLMs in CH and DL domains.

A first line of research explicitly treats pretrained language models as repositories of factual and relational knowledge [4]: what emerged is that masked and autoregressive models can answer factual queries with accuracy comparable to traditional, open-domain QA systems, paving a new research branch that aims at using LLMs as KBs. This work and its successors highlight some clear strengths of "fact-based" LLMs since they: require no schema definition, can in principle cover an open-ended set of relations, and can be updated simply by a new pretraining or providing additional context (e.g., via RAG or fine-tuning). At the same time, subsequent studies on factuality and hallucinations [5] emphasize structural weaknesses that are particularly problematic also in the CH domain: knowledge is incomplete and unevenly distributed, heavily biased toward popular entities and anglophone sources. Moreover, this knowledge is difficult to control temporally, and sources for facts are not always returned by the models and they are hard to be traced back to explicit evidence, so provenance and accountability may be opaque. Surveys on factuality in LLMs [6] show that even state-of-the-art models consistently exhibit non-negligible rates of incorrect statements, and that these errors are especially frequent for long-tail entities and specialized domains. For our purposes, this means that treating an LLM as a stand-alone catalogue for the CH domain (e.g., for artworks and artists) must be done with caution: its strengths lie in broad coverage and flexible querying, whereas its weaknesses concern exactly the kind of accurate, provenance-aware and complete facts that CH workflows rely on.

A related but distinct research thread uses LLMs not as sources of facts but as *judges* that provide numerical evaluation of different qualitative or confidence measures. Early work on "LLM-as-a-Judge" frameworks [7] reports encouraging correlations with human judgments in summarization and dialogue, and large-scale benchmarks such as MT-Bench [8] and AlpacaEval [9] have popularized LLM judges as a standard evaluation tool. However, more recent empirical studies [10] question the robustness of this practice: results show that LLMs-as-judges can be sensitive to superficial prompt changes, exhibit systematic biases, and sometimes disagree with each other and with human raters in non-trivial ways, raising concerns about reproducibility and reliability. Surveys of LLM-based evaluation methods [11] echo these concerns, noting issues such as positional bias, style over-substance preferences, and difficulties in maintaining stable scoring criteria over time. In the CH context, where cataloging decisions and interpretive judgments are often contested and historically situated, these limitations suggest that LLMs should not be treated as authoritative adjudicators, but at most as auxiliary tools whose biases must be carefully understood.

Within DLs and CH, LLMs have so far been explored mainly as instruments for metadata enrichment, description, and access support [1], rather than as primary KBs. Hence, our work builds on this gap: rather than asking how LLMs can help enrich an existing catalog, we ask to what extent they can act as a catalogue to answer structured, SQL-like questions in the CH domain, and whether a system like Galois, originally developed in a general data management setting, can mitigate some of the known weaknesses of fact-based LLMs by imposing a database-style execution discipline. To our knowledge, this combination of "LLM as implicit catalog" and SQL-over-LLM execution has not yet been systematically studied in the CH literature, and our study is intended as a first step in that direction.

3. Background: The Galois System

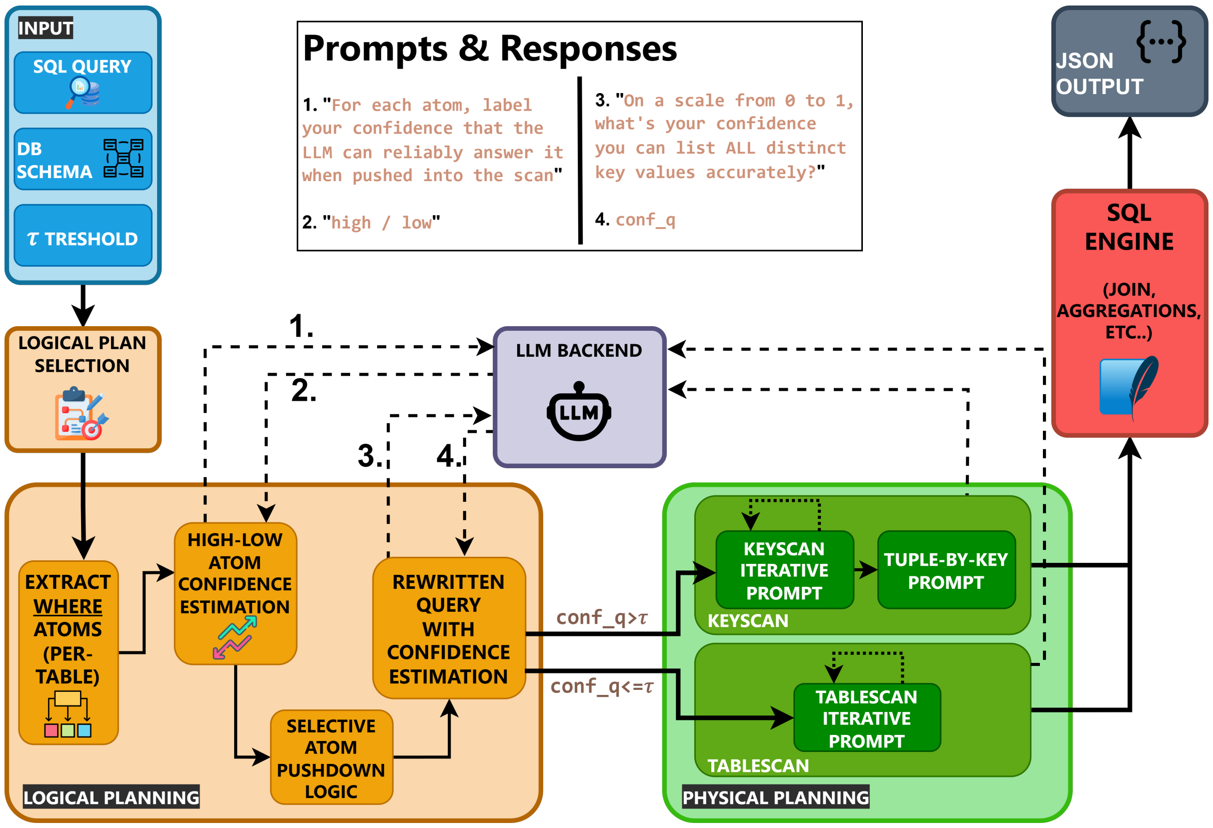
Galois[3] has been presented at ACM SIGMOD 2025 and represents the first complete system aiming to query an LLM as a structured data source. Galois has been compared to two standard baselines including querying an LLM via natural language and via SQL queries. From the experiments, we see that Galois can improve both accuracy and completeness of the answers in several experimental benchmarks, some well-established as Spider[1] and others built ad-hoc for the task.

Overall, Galois bridges two realms. On the one side, users formulate information needs as SQL queries over a virtual schema that describes the data they are interested in (for instance, tables of paintings, artists, and movements). On the other side, the actual facts are not stored in a conventional database, but are implicitly encoded in an LLM. Galois is responsible for bridging this gap: it parses SQL, turns

[1]https://spider2-sql.github.io/

it into a query plan, decides how to decompose that plan into smaller interactions with the LLM, and assembles the partial answers into a final result table.

The Galois architecture, shown in Figure 1 is a pipeline organized around a small set of core components. The entry point is the prompting interface that we can also see as a user-friendly interface to issue SQL queries. Users or client applications send SQL queries that describe the desired result: which attributes should appear in the output, which tables are involved, and which conditions must hold.


Prompts & Responses

1. "For each atom, label your confidence that the LLM can reliably answer it when pushed into the scan"

2. "high / low"

3. "On a scale from 0 to 1, what's your confidence you can list ALL distinct key values accurately?"

4. conf_q

Figure 1: Architecture representation of the Galois system. On the left, users and client applications issue SQL queries over a (possibly, self-defined) logical DB schema that describes their domain of interest. These queries enter the Galois core through the SQL front-end, which parses and analyses them to produce an initial logical plan. The logical optimizer refines this plan by deciding where query constraints should be placed, and in particular whether conditions in the WHERE clause should be pushed down into LLM-facing scan operators (FilterLLMScan) or evaluated later on materialized data (LLMScan). The confidence catalogue stores precomputed estimates of how reliably the LLM can handle different attributes and predicates, and feeds this information into the optimizer. The physical planner instantiates the chosen logical plan with concrete operators, selecting among alternative scan strategies such as TableScan (direct tuple retrieval) and KeyScan (key-first retrieval with focused follow-up prompts), while all other operators (joins, projections, aggregations) are relegated at the end of the pipe to a standard DBMS.

The SQL is then parsed and analyzed by the front-end: here, syntax and schema checks are performed, minor standardizations are applied (e.g., table aliases normalization) and an initial logical plan is produced. The logical plan is a tree of relational operators such as selection, projection, join, and aggregation, together with one or more logical scan operators to access the underlying LLM.

Once an initial plan is available, the logical optimizer explores different ways of organizing the computation, with explicit awareness of LLM characteristics. It determines which query conditions should be pushed down into LLM-facing scan operators and which should be evaluated on materialized data. This decision reflects a fundamental trade-off: plans that push more conditions into the scan reduce the number of tuples requiring processing but create more complex prompts that may be error-prone for the LLM. Plans that keep conditions outside the scan require simpler prompts but necessitate retrieving and filtering more data locally, potentially introducing incompleteness issues.

To reason about this trade-off, Galois maintains a confidence catalogue – a metadata repository that approximates, for each attribute and predicate type, how reliably the LLM can enforce given conditions. These confidence scores are obtained through calibration queries that ask the LLM to self-assess its reliability for specific filtering operations or by observing how consistently conditions are respected in small test queries. While the catalogue contains no actual data, it provides optimization signals that guide planning decisions.

Based on this metadata, the logical optimizer chooses between two primary logical access patterns: `LLMScan`, where the system requests tuples without applying selection conditions in the prompt, and `Filter-LLMScan`, where conditions are embedded in the prompt.

The physical planner instantiates the chosen logical plan with concrete execution strategies. Galois provides two physical scan operators: `TableScan`, which asks the LLM to produce complete tuples directly through iterative prompting until no new rows appear, and `KeyScan`, which operates in two stages—first requesting key values iteratively, then issuing focused follow-up prompts to retrieve remaining attributes for each key. The choice between them is guided by a cost-quality model considering the number of prompts, token usage, and expected reliability based on the confidence catalogue.

Finally, the execution engine applies deterministic relational operations (joins, projections, aggregations) to the output of physical scan operators. Galois delegates these operations to an embedded DBMS (e.g., SQLite or DuckDB [12]), ensuring correctness while focusing optimization efforts on the LLM interaction layer. Results are delivered to users in JavaScript Object Notation (JSON) format.

Galois implements multiple optimization configurations that represent different points in the cost-quality trade-off space:

- **Galois$_{\text{WO}}$** (Without Optimization): Applies no predicate pushdown; all filtering occurs locally after data retrieval via KeyScan operations. This approach generates extensive LLM interactions but provides maximal control over constraint enforcement.
- **Galois$_{\text{S}}$** (Selective): Never uses KeyScan, relying exclusively on TableScan operations. The system allows the LLM to decide which predicates from the WHERE clause to apply during tuple generation, with remaining constraints enforced locally.
- **Galois$_{\text{A}}$** (All pushdown): Pushes all WHERE clause predicates to the database layer, relegating the LLM to choosing the access path (TableScan vs. KeyScan) based on confidence scores.
- **Galois$_{\text{F}}$** (Full optimization): Employs confidence scores to selectively determine which predicates to push down to the LLM. High-confidence predicates (typically above a threshold of 0.6–0.7) are embedded in prompts, while low-confidence conditions are evaluated locally on materialized results.

From the user's perspective, these implementation details remain hidden. The system accepts SQL queries and returns structured tables, internally orchestrating when and how to interact with the LLM and combining model outputs with deterministic in-memory processing to maximize answer quality while minimizing computational costs.

3.1. A Running Example: Querying for Paintings with Galois

To make the functioning of Galois more concrete, we walk through a simple running example. The example is illustrated in Figure 2.

In NL the query is: `List the names of all works created by artists whose last name is "Mantegna"`, while the corresponding SQL query is:

```sql
SELECT w.name
FROM target.work AS w
JOIN target.artist AS a
  ON w.artist_id = a.artist_id
WHERE a.last_name = 'Mantegna';
```

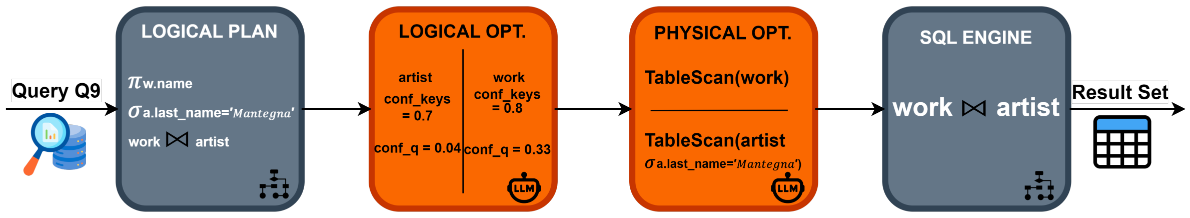


Figure 2: Running example of the Galois$_F$ pipeline for query Q9 (*works by Mantegna*) on the paintings benchmark. The SQL query is first translated into a logical plan, then the logical optimizer consults the confidence catalogue and decides to push down the predicate on `artist` only. Since both query-level confidences fall below the threshold $\tau = 0.6$, the physical optimizer selects `TableScan` for both `work` and `artist`. The resulting tables are finally joined and projected by the embedded SQL engine to produce the result set.

At the logical level, Galois parses this query, normalizes table aliases, and builds a standard relational plan of the form

$$\pi_{\texttt{w.name}}\Big(\sigma_{\texttt{a.last\_name}='Mantegna'}(\texttt{work} \bowtie_{\texttt{w.artist\_id}=\texttt{a.artist\_id}} \texttt{artist})\Big),$$

with one logical scan for `work` and one for `artist`. The only selection predicate applies to the `artist` table (`a.last_name = 'Mantegna'`); no condition constrains `work` directly instead.

In the Galois$_F$ configuration, the logical optimizer consults the confidence catalogue and decides, for each table, which predicates should be pushed into the LLM-facing scan. Since there is no predicate on `work`, the corresponding scan is configured with strategy *none*, meaning that no filter is pushed down and the table is retrieved unconditionally. For `artist`, instead, since the equality condition on `last_name` is labeled as `HIGH`, the optimizer selects the single predicate pushdown strategy.

The physical planner then instantiates both logical scans as `TableScan` operators, choosing to retrieve complete tuples directly from the LLM. This happens because, for this query, the confidence catalogue assigns `conf_keys` $= 0.8$ to `work` and `conf_keys` $= 0.7$ to `artist`, but Galois$_F$ bases its decision on the more conservative query-level scores `conf_q` ≈ 0.33 and `conf_q` ≈ 0.04, both below the threshold $\tau = 0.6$ used to enable key-based access.

At execution time, Galois first issues a `TableScan` initial prompt that asks the model to populate the table `(work_id, name, artist_id, style, museum_id)` without filters. An iteration of subsequent prompt of the form "List more values if there are more, otherwise return an empty JSON" is used to request new batches. The same procedure is repeated after this completes, for the `artist` table.

Once both scans are complete, their results are stored in the embedded DBMS, which performs the join on `artist_id` and applies the final projection on `w.name`. From the user's point of view, the system simply answers the SQL query with a table containing the names of works by Mantegna. Internally, however, Galois$_F$ has used confidence-guided optimization to decide where predicate enforcement should happen, has decomposed the query into two smaller LLM interactions, and has delegated join and projection to a conventional relational engine.

4. A Cultural Heritage oriented Benchmark for Galois

To study how a SQL-over-LLM system behaves on CH content, we build a small but focused benchmark grounded in the *Famous Paintings*[2] dataset from Kaggle, a collection originally released to support SQL practice over data about paintings, artists, and museums. We reinterpret this resource as a proxy for a curated art catalogue and derive from it both a logical schema and a controlled set of information needs. The resulting benchmark is rich enough to exhibit typical patterns of CH querying, such as filtering by nationality, period, and style, retrieving museums by location, and joining works with their authors.

[2]https://www.kaggle.com/datasets/mexwell/famous-paintings

Table	#attributes	#rows
artist	9	421
museum	9	57
work	5	14 776
subject	2	6 771
canvas_size	4	200
product_size	4	110 347
image_link	4	14 775

Table 1
Basic statistics of the paintings benchmark schema. For each table we report the number of attributes and rows after importing the Kaggle CSV files into a relational representation. Together, the tables cover 421 artists, 57 museums, 14 776 works, and their associated subjects, sizes, prices, and image links.

4.1. Data and Schema

As reported in Table 1, the original Kaggle collection consists of several Comma Separated Values (CSV) files that describe artists, works, museums, subjects, canvas sizes, product sizes, and image links. We import these files into a relational schema that we expose to Galois through a JSON description. At the core of this schema lies the `work` table, which contains 14 776 paintings identified by a numeric key and characterized by a title (`name`), a reference to the `artist` table (`artist_id`), a stylistic label, and an optional reference to the `museum` table (`museum_id`). The `artist` table lists 421 artists with their full name and a decomposition into first, middle, and last name, plus nationality, stylistic label, and birth and death years. The `museum` table comprises 57 institutions, described by name, postal address, city, state or region, postal code, country, phone, and URL.

On the descriptive side, the `subject` table links paintings to iconographic subjects. It contains 6 771 rows that associate 6 008 distinct works to one or more subject labels drawn from a controlled vocabulary of 29 values such as "Still-Life", "Horses", or "Marine Art/Maritime". Physical artifacts are modeled through two tables. The `canvas_size` table lists 200 size templates, each identified by a `size_id` and described by width, height, and a human-readable label. The `product_size` table records 110 347 combinations of work and canvas size with their corresponding sale and regular prices, reflecting the dataset's origin in an e-commerce context. Finally, the `image_link` table contains 14 775 image records that associate 14 715 distinct works with URLs pointing to the digitized artifacts.

For the purposes of this first benchmark we restrict our attention to the three core tables `artist`, `museum`, and `work`, which together capture the essential relationships between creators, their works, and the institutions that hold them. The remaining tables are still part of the schema exposed to Galois and could be targeted by more complex queries, but they are not used for this experiment. This focuses our analysis on factual knowledge about artists, stylistic movements, and museum locations, which are precisely the aspects that one might expect a general-purpose LLM to have encoded from web and reference sources, independently of this specific dataset.

4.2. Information Needs and Query Set

On top of this schema, we define a set of 14 information needs that are meant to resemble reasonable questions a curator or researcher might pose about artists and museums. Each information need is expressed both as a natural language question and as an SQL query over the benchmark schema, and the result of the SQL query serves as ground truth for evaluation.

Concretely, the natural language queries are:

Q1. List the names and cities of all museums in France.
Q2. List the full names of all French artists who were born between 1800 and 1899.
Q3. List the full names and nationalities of all French artists whose style is Baroque.
Q4. List the names and states of all museums in the USA that are located in the state of New York.

Q5. List the names and cities of all museums in the UK whose name is "National Gallery".
Q6. List the full names of all artists in the Rococo style who were born before 1700.
Q7. List the full names of all Spanish artists who died after 1950.
Q8. List the names of all museums located in the Netherlands.
Q9. List the names of all works created by artists whose last name is "Mantegna".
Q10. List the names and cities of all museums located in the USA.
Q11. List the full names of all English artists whose style is Rococo.
Q12. List the full names of all artists who were born between 1880 and 1890.
Q13. List the names and cities of all museums located in Madrid, Spain.
Q14. List the nationality and artistic style of the artist whose first name is Claude and last name is Monet.

Roughly half of these queries target museums and their locations (Q1, Q4, Q5, Q8, Q10, Q13), exercising the `museum` table by filtering on country, state, or city or by constraining on a specific institution name. The remaining queries focus on artists and their properties (Q2, Q3, Q6, Q7, Q11, Q12, Q14), probing nationality, stylistic labels, and life dates, while Q9 is deliberately included to test a join between the `artist` and `work` tables via the artist's last name.

From the perspective of query structure, this set spans a range of simple but representative patterns. Some questions involve only a single equality predicate on a categorical attribute, such as restricting museums to a given country or artists to a given nationality. Others combine multiple equality predicates, for instance by requiring that an artist satisfy both a nationality and a stylistic condition, or that a museum match both a country and a state. Some queries introduce temporal constraints in the form of ranges over birth or death years. The join-based query Q9 tests the model's ability to resolve specific individuals from their surname and then project their works. In absolute terms, the ground-truth answers to these queries are modest in size, with heterogeneous cardinalities. Over the 14 queries, result sizes span from one to 81 tuples, with an average of about 12 rows.

4.3. Benchmark Characteristics

Although compact, the benchmark exhibits several properties that are important when assessing SQL-over-LLM systems for CH. First, it embodies a clear separation between the logical schema and the LLM's knowledge. The schema and instances we use to compute ground-truth answers are derived entirely from the Kaggle dataset and are not exposed to the model at query time. From Galois' perspective, the database is empty; all facts about artists and museums must be elicited from the LLM through scan operators. This setup mirrors a scenario in which an institution wishes to prototype a database-oriented access layer over an LLM without yet committing to mirroring its catalogue into a traditional database.

Second, the dataset is intentionally projected toward relatively well-known entities, such as major museums and famous artists. This reflects the original goal of the Kaggle collection and offers a favorable test for the LLM, since famous names are more likely to be represented in its training data. At the same time, the presence of multiple artists within the same stylistic movement, as well as multiple museums within the same country or city, creates opportunities for confusion that reveal how the system handles disambiguation and fine-grained filtering.

Third, the query set isolates distinct classes of information needs. The museum-focused questions probe the model's knowledge about cultural facts and its ability to associate institutions with the correct city, state, and country. The artist-focused questions probe biographical and historical knowledge, including nationalities and artistic movements, and test whether temporal constraints are respected. Join queries function as a test of relational reasoning, where the system must first identify the correct artists and then project their works.

Finally, because all results can be computed exactly from the dataset, the benchmark supports a detailed, cell-level comparison between what Galois extracts from the LLM and what the curated tables contain. In later sections we will exploit this property by adopting the same evaluation metrics used in the original Galois work, which distinguish between correct values, missing tuples, spurious tuples, and violations of tuple-level constraints. For now, the important point is that the benchmark provides a controlled environment in which the strengths and weaknesses of using an LLM as an implicit catalogue for paintings and museums can be observed with precision, while still connecting closely to the kinds of questions that arise in everyday CH practice.

5. Experimental Setup

All experiments were run on a Ubuntu Linux Virtual Machine (VM). The VM is backed by an AMD EPYC processor with four virtual CPU cores at approximately 3.2 GHz, each with 512 KB of L2 cache and AVX2 support, and 4 GB of RAM. No GPU acceleration is used: all heavy computation for language modeling happens on the side of the external model provider, while the local machine is responsible for orchestrating queries, parsing results, and computing evaluation metrics.

We consider as LLM backend the *GPT-4o-mini* model, accessed via an Azure-hosted deployment of the OpenAI chat completion Application Programming Interface (API). In this configuration, the Galois client uses the `AzureOpenAI` Software Development Kit (SDK)[3], with the model identifier set to `gpt-4o-mini`. We use the chat-style API and enforce JSON output formatting at the prompt level, so that the responses can be parsed into lists of records. Decoding parameters are aligned with the Galois reference implementation: `temperature` is fixed to 0.0 and a fixed random seed is used whenever the backend supports it. These choices favour determinism and reproducibility over diversity of generations.

The software stack is based on Python 3.12.3, running the reproduced Galois reference implementation packaged as a small `py_galois` module. The code follows a modular design: a central `runner` class reads dataset metadata and SQL/NL queries, constructs logical and physical plans using the components described in Section 3, and executes them against the LLM interface. The implementation relies only on the Python standard library for planning and evaluation logic (e.g., `argparse`, `json`, `pathlib`) and on the external `OpenAI` SDK client for Azure/OpenAI LLM access.

On top of this infrastructure we run six querying modes over the aforementioned benchmark. In the NL mode, we send the natural language query directly to the chosen LLM, asking it to return a JSON array of objects with the appropriate attributes. In the SQL mode, we send the reference SQL query as a string and ask the model to "execute" it by producing the corresponding table in JSON form, without any intermediate planning. To ensure a fair comparison between these baselines and Galois as in the original work, we iterate until the model no longer returns new tuples. For the four *Galois* settings – i.e., Galois$_{\mathrm{WO}}$, Galois$_{\mathrm{S}}$, Galois$_{\mathrm{A}}$, Galois$_{\mathrm{F}}$ – we run the full Galois pipeline over the same SQL query: the `runner` parses the query, builds and optimizes logical plans, selects physical operators (including LLMScan vs. FilterLLMScan and TableScan vs. KeyScan), and executes the resulting plan by issuing multiple, simpler prompts to the LLM. We use the default configuration of the reference implementation, including the fixed confidence threshold $\tau = 0.6$ for deciding between key-based and table-based scans. For each combination of query and mode, we record the generated rows, total token usage reported by the SDK and latency as measured on the local VM; these quantities are used for the evaluation, based on the same six metrics used by the authors in the original Galois paper:

- **F1-Cell**: treats each table as a set of unique, normalized cell values (ignoring row/column position) and computes the F1 score between the ground-truth and predicted cell sets, thus capturing how many factual values are correctly produced versus missing or spurious ones;
- **Cardinality**: measures how close the number of predicted rows is to the ground truth, defined as $\min(|E|, |A|)/\max(|E|, |A|)$ where E and A are the sets of expected and actual rows; it is maximal when the system returns exactly the right number of tuples, regardless of content;
- **Tuple Constraint**: operates at the tuple level: after normalizing rows, it checks for each distinct ground-truth tuple whether it appears in the system output with exactly the same multiplicity, and reports the fraction of ground-truth tuples whose multiplicity is correctly matched, thereby capturing how well full rows (not just individual cells) are reconstructed;

[3]https://learn.microsoft.com/en-us/fabric/data-science/ai-services/how-to-use-openai-sdk-synapse?tabs=python0

Metric	NL	SQL	Galois$_{\mathrm{WO}}$	Galois$_{\mathrm{S}}$	Galois$_{\mathrm{A}}$	Galois$_{\mathrm{F}}$
F1-Cell	**0.947**	0.709	0.418	0.737	0.876	0.876
Cardinality	0.475	0.409	0.217	0.418	0.502	**0.504**
Tuple Constr.	0.267	0.251	0.003	0.269	0.302	**0.336**
AVG-Score	0.563	0.457	0.213	0.475	0.560	**0.572**
#Tokens (M)	0.003	0.003	0.097	0.018	0.016	0.016
Avg Time (s)	2.701	1.573	51.877	7.877	6.555	6.381

Table 2
Performance of NL, SQL, and Galois variants with `gpt-4o-mini` on the paintings benchmark (14 queries).

- **AVG-Score**: is the arithmetic mean of these three quantities and provides a single summary indicator of table quality;
- **# of Tokens (Millions)**: is the total number of tokens consumed across all the queries, allowing for cost comparisons between experiments;
- **AVG Time (s)**: is the average execution time per query.

6. Results and Discussion

In this section we compare baselines results with the four Galois configurations (Galois$_{\mathrm{WO}}$, Galois$_{\mathrm{S}}$, Galois$_{\mathrm{A}}$, Galois$_{\mathrm{F}}$) on the Famous Paintings benchmark. Table 2 reports the main quantitative results, following the same metrics of the original Galois paper. We then compare them with the original results.

Baselines: NL and SQL. A first observation is that the NL baseline is very strong in this setting. Direct natural-language answers reach an AVG-Score of 0.563 and an F1-Cell of 0.947, which means that, for most queries, the content of the returned tables is almost perfectly aligned with the ground truth at the cell level. Cardinality and tuple construction scores (0.475 and 0.267 respectively) are also solid. The average time per query remains small (about 2.7 seconds) and the token usage is minimal (0.003M), so direct NL interaction is both accurate and efficient.

The SQL baseline shows a different trade-off. It is the fastest method overall, with an average time of 1.57 seconds, and it uses the same small number of tokens as NL. However, its AVG-Score is clearly lower (0.457), and all fine-grained metrics lag behind NL: F1-Cell drops to 0.709, Cardinality to 0.409, and Tuple Constraint to 0.251. In other words, when the model is forced to reason only in SQL, without the additional reasoning scaffolding of Galois, it tends to make more mistakes in the final answers.

This behavior contrasts with the original paper, where NL was substantially weaker (AVG-Score 0.254, F1-Cell 0.237) and SQL slightly stronger (AVG-Score 0.481, F1-Cell 0.431). In our experiments, NL has improved dramatically, while SQL remains competitive but not dominant. This difference is important, because it raises the bar for Galois: the system is no longer competing with a weak NL baseline, but with an already high-performing one.

Galois with and without structural guidance. Galois$_{\mathrm{WO}}$ represents the most unconstrained variant. Here the model always uses KeyScan and Tuple-by-Key through the LLM, never pushing conditions to the database, but applying all the constraints only inside the LLM prompt. In our experiments this configuration performs poorly. Its AVG-Score is only 0.213, much worse than both NL and SQL, and the F1-Cell and Tuple Constraint scores are also very low (0.418 and 0.003 respectively). At the same time, it is by far the most expensive variant: it consumes about 0.097M tokens per query and requires more than 50 seconds on average. The combination of low accuracy and high cost makes this configuration clearly suboptimal. This pattern is consistent with the original paper, where Galois$_{\mathrm{WO}}$ also exhibited very large token and time requirements and only modest accuracy gains. In our setting, with a small number of queries, this variant looks essentially like unstructured scratchpad exploration that does not translate into better final outputs.

When we move to Galois$_S$, the picture improves significantly. This variant, in contrast with Galois$_{WO}$, never uses KeyScan: it always does a full TableScan on each table and only lets the LLM decide which conditions to push down to the database. Here, the resulting AVG-Score climbs to 0.475, slightly above SQL. F1-Cell also increases to 0.737, and the cardinality and tuple metrics (0.418 and 0.269) are consistently better than those of SQL. The computational cost remains higher than the baselines, but is much more reasonable than for Galois$_{WO}$: the model uses 0.018M tokens per query and runs in about 7.9 seconds on average. Galois$_S$ therefore shows that adding structured Galois steps can already help the system correct or refine SQL answers. However, the gains over plain SQL are moderate, and the configuration does not yet match the NL baseline.

State-of-the-art Galois: Galois$_A$ and Galois$_F$. The stronger variants, Galois$_A$ and Galois$_F$, are the most relevant for the main message of this work. They tighten the interaction between SQL and the model's internal reasoning and verification.

Galois$_A$, which always pushes all WHERE conditions down to the database, relegating the LLM to just choosing the access path, achieves an AVG-Score of 0.560, which is essentially aligned with NL (0.563) and clearly above SQL (0.457). F1-Cell increases to 0.876, and the cardinality and tuple scores (0.502 and 0.302) also improve over SQL. Importantly, the cost remains much lower than that of Galois$_{WO}$: 0.016M tokens and about 6.6 seconds per query. This configuration already shows that a carefully designed Galois pipeline can lift SQL-based interaction up to NL level in terms of answer quality.

Galois$_F$ is the best-performing configuration in our experiments. It reaches the highest AVG-Score among all methods (0.572), slightly surpassing NL despite using SQL as the primary interface. Tuple Constraint improves further to 0.336, and the Cardinality score reaches 0.504, which is higher than both NL and SQL. In fact, compared to SQL, Galois$_F$ gains 0.115 points in AVG-Score (0.572 vs. 0.457) and improves all three fine-grained metrics by noticeable margins. The cost is still moderate: token usage is the same as for Galois$_A$ (0.016M) and runtime is around 6.4 seconds per query.

These results are particularly important because they show that Galois is effective even when the underlying model already performs very well in pure natural language. The original paper demonstrated that Galois could rescue a relatively weak NL baseline and significantly outperform it. In our case, the NL baseline is already strong, yet the best Galois configuration still manages to match and slightly exceed it using only SQL interaction. This makes the result more robust: Galois is not merely compensating for an underpowered NL setup, but genuinely enhancing structured querying.

Comparison with the original work. The comparison with the original Galois results highlights two shifts. First, NL performance has improved substantially. The original work reported an AVG-Score of 0.254 and an F1-Cell of 0.237 for NL, whereas we measure 0.563 and 0.947 respectively. Second, plain SQL no longer dominates NL: the original SQL baseline achieved an AVG-Score of 0.481 and F1-Cell of 0.431, slightly higher than NL, while in our case SQL lags behind NL in all metrics.

Despite these differences, the qualitative conclusions of the original work also hold in our experiment. In fact, in both settings Galois variants that make principled use of structure (Galois$_S$, Galois$_A$, Galois$_F$) outperform plain SQL, and the strongest configuration (Galois$_F$) attains the best overall accuracy. What changes is the size of the margin over NL. The smaller improvement is expected given the stronger baseline, but the fact that Galois in its full optimizations configuration can still match and slightly surpass direct NL querying is encouraging.

7. Discussion and Final Remarks

Our case study set out to answer two questions: whether a general-purpose LLM can be treated as a usable KB for CH, and how different querying paradigms, natural language, plain SQL, and SQL mediated by Galois compare in this setting. The paintings benchmark and the results reported in Table 2 allow us to draw some preliminary conclusions.

First, in our setup direct NL interaction is already very strong, both in accuracy and in efficiency. On a relatively small but non-trivial set of CH information needs, a well-configured GPT-4o-mini backend produces tables that are, at the cell level, almost perfectly aligned with the curated dataset, with modest latency and token usage. This is a higher starting point than in the original Galois experiments and suggests that up-to-date LLMs can reliably retrieve factual information in tabular form when prompted carefully in natural language. For practitioners, this means that a simple, iterative "ask in English and get a JSON table back" pattern is already a competitive baseline for prototyping LLM-based access to collections. Second, plain SQL prompting is fast and compact but significantly less accurate than NL. Asking the model to "execute" an SQL query without any intermediate planning yields lower scores on all evaluation dimensions, despite using similar amounts of tokens and even slightly less time. This confirms that the ability of LLMs to parse and reason over SQL is still more fragile than their ability to respond to well-structured natural language instructions; this means that a CH system that relies only on SQL-to-LLM prompting would therefore leave non-trivial accuracy "on the table". Third, and most importantly for this paper, Galois, when instantiated in its more structured variants, successfully closes this gap. Configurations that simply offload work to the model without strong structural guidance (such as $\text{Galois}_{\text{WO}}$) are both expensive and inaccurate. However, once conditions are systematically pushed into the database side and the interplay between logical and physical plans is exploited (Galois_{S}, Galois_{A}, Galois_{F}), SQL-based interaction recovers most of the benefits of NL. In particular, Galois_{F} raises the performance of SQL-based querying to the level of NL and slightly beyond, while keeping token usage and latency at a moderate level. In other words, a carefully designed execution pipeline can make structured access via SQL as effective as free-form NL access. This last point is particularly relevant for scenarios where NL interfaces are unavailable or undesirable. In many DL and CH systems, the primary interaction layer is not a chat box but a set of structured search forms, APIs, or internal reporting tools that already speak SQL (or SQL-like query languages). In such contexts, relying on SQL alone would imply a measurable drop in answer quality with respect to what the LLM could in principle provide. Our experiments suggest that SQL combined with Galois can mitigate this loss: instead of asking the model to "run" arbitrary SQL strings end-to-end, the system can decompose queries into a sequence of LLM scans and deterministic relational operations, recovering much of the accuracy of NL while preserving the advantages of a structured interface and a clear separation between logical intent and physical execution. At the same time, several limitations of this study must be acknowledged. The benchmark is intentionally focused on a single, relatively clean dataset centered on well-known paintings and institutions; it does not capture the full complexity of real-world catalogs, with noisy records, multilingual descriptions, and intricate authority control. Our queries mostly involve selections and simple joins over a narrow schema, while many CH tasks require deeper joins, aggregations, and reasoning over classifications and events (e.g., provenance chains, exhibition histories). We also consider only one backend and a single configuration of decoding parameters, so the generality of the observed patterns across models and providers remains to be validated. Most importantly, we take the curated paintings dataset as ground truth and treat the LLM purely as an implicit catalog; hybrid scenarios in which LLM output is cross-checked against knowledge graphs or linked open data remain unexplored.

Future work will therefore move in two complementary directions. On the methodological side, we plan to extend the benchmark to richer schemas and to design query sets that better reflect the analytical tasks of digital humanists. On the infrastructural side, we aim at applying the same Galois-based approach to larger CH aggregators, such as a subset of Europeana [13]. This setting would stress both the factual coverage of LLMs and the scalability of SQL-over-LLM execution and enables to explore architectures in which Galois interacts not only with models but also with retrieval services.

In conclusion, this work provides an initial answer to the question of whether LLMs can be queried as structured KBs for CH. On a controlled benchmark, we have shown that modern LLMs can already perform remarkably well in direct NL interaction, that plain SQL prompting underutilises their capabilities, and that a system like Galois can largely bridge the resulting gap, making SQL-based access a viable option even when only structured interfaces are available. We see this as a promising starting point for a broader research agenda in which database-guided techniques and digital humanities concerns meet in the design of accountable, efficient, and expressive LLM-based access layers for CH collections.

Declaration on Generative AI

During the preparation of this work, the author(s) used GPT-5.1 for grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] X. Hu, Application of Large Language Models for Digital Libraries, Association for Computing Machinery, New York, NY, USA, 2025. URL: https://doi.org/10.1145/3677389.3702617.

[2] X. Gao, J. Liu, H. Xu, L. Huang, Improving llm reasoning via dependency-aware query decomposition and logic-parallel content expansion, 2025. URL: https://arxiv.org/abs/2510.24390. arXiv:2510.24390.

[3] D. Satriani, E. Veltri, D. Santoro, S. Rosato, S. Varriale, P. Papotti, Logical and physical optimizations for SQL query execution over large language models, Proc. ACM Manag. Data 3 (2025) 181:1–181:28. URL: https://doi.org/10.1145/3725411. doi:10.1145/3725411.

[4] F. Petroni, T. Rocktäschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu, A. Miller, Language models as knowledge bases?, in: K. Inui, J. Jiang, V. Ng, X. Wan (Eds.), Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Association for Computational Linguistics, Hong Kong, China, 2019, pp. 2463–2473. URL: https://aclanthology.org/D19-1250/. doi:10.18653/v1/D19-1250.

[5] C. Jiang, B. Qi, X. Hong, D. Fu, Y. Cheng, F. Meng, M. Yu, B. Zhou, J. Zhou, On large language models' hallucination with regard to known facts, 2024. URL: https://arxiv.org/abs/2403.20009. arXiv:2403.20009.

[6] C. Wang, X. Liu, Y. Yue, Q. Guo, X. Hu, X. Tang, T. Zhang, C. Jiayang, Y. Yao, X. Hu, Z. Qi, W. Gao, Y. Wang, L. Yang, J. Wang, X. Xie, Z. Zhang, Y. Zhang, Survey on factuality in large language models, ACM Comput. Surv. 58 (2025). URL: https://doi.org/10.1145/3742420. doi:10.1145/3742420.

[7] D. C. Chiang, H. Lee, Can large language models be an alternative to human evaluations?, in: A. Rogers, J. L. Boyd-Graber, N. Okazaki (Eds.), Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023, Association for Computational Linguistics, 2023, pp. 15607–15631. URL: https://doi.org/10.18653/v1/2023.acl-long.870. doi:10.18653/V1/2023.ACL-LONG.870.

[8] L. Zheng, W. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, I. Stoica, Judging llm-as-a-judge with mt-bench and chatbot arena, in: A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

[9] Y. Dubois, B. Galambosi, P. Liang, T. B. Hashimoto, Length-controlled alpacaeval: A simple way to debias automatic evaluators, CoRR abs/2404.04475 (2024). URL: https://doi.org/10.48550/arXiv.2404.04475. doi:10.48550/ARXIV.2404.04475. arXiv:2404.04475.

[10] A. Bavaresco, R. Bernardi, L. Bertolazzi, D. Elliott, R. Fernández, A. Gatt, E. Ghaleb, M. Giulianelli, M. Hanna, A. Koller, A. F. T. Martins, P. Mondorf, V. Neplenbroek, S. Pezzelle, B. Plank, D. Schlangen, A. Suglia, A. K. Surikuchi, E. Takmaz, A. Testoni, Llms instead of human judges? A large scale empirical study across 20 NLP evaluation tasks, in: W. Che, J. Nabende, E. Shutova, M. T. Pilehvar (Eds.), Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025, Association for Computational Linguistics, 2025, pp. 238–255. URL: https://doi.org/10.18653/v1/2025.acl-short.20. doi:10.18653/V1/2025.ACL-SHORT.20.

[11] H. Li, Q. Dong, J. Chen, H. Su, Y. Zhou, Q. Ai, Z. Ye, Y. Liu, Llms-as-judges: A comprehensive

survey on llm-based evaluation methods, CoRR abs/2412.05579 (2024). URL: https://doi.org/10.48550/arXiv.2412.05579. doi:10.48550/ARXIV.2412.05579. arXiv:2412.05579.

[12] M. Raasveldt, H. Mühleisen, Duckdb: an embeddable analytical database, in: P. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, T. Kraska (Eds.), Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019, ACM, 2019, pp. 1981–1984. URL: https://doi.org/10.1145/3299869.3320212. doi:10.1145/3299869.3320212.

[13] N. Aloia, C. Concordia, C. Meghini, Europeana: Towards the european digital library, in: M. Agosti, F. Esposito, C. Thanos (Eds.), Post-proceedings of the Fifth Italian Research Conference on Digital Libraries - IRCDL 2009, Padova, Italy, 29-30 January 2009, DELOS: an Association for Digital Libraries / Department of Information Engineering of the University of Padua, 2009, pp. 154–157. URL: http://www.dei.unipd.it/%7Eagosti/ircdl/atti-ircdl2009-finale.pdf.