ICEAGE: Intelligent Contextual Exploration and Answer Generation Engine for Scientific Data Discovery

Chenxu Niu
Texas Tech University
Lubbock, TX, USA
Chenxu.Niu@ttu.edu

Wei Zhang
Lawrence Berkeley National Laboratory
Columbus, OH, USA
zhangwei217245@lbl.gov

Mert Side
Texas Tech University
Computer Science
Lubbock, TX, USA
mert.side@ttu.edu

Yong Chen
Texas Tech University
Lubbock, TX, USA
yong.chen@ttu.edu

Abstract

More and more scientific applications store datasets in scientific data formats such as HDF5 and netCDF. However, existing search methods for scientific data formats generally require researchers to be familiar with the formats and metadata structure, resulting in a steep learning curve. Therefore, researchers need a natural language query method to query scientific data. In this paper, we propose ICEAGE, a novel Intelligent Contextual Exploration and Answer Generation Engine that bridges the gap between natural language querying and scientific data and metadata retrieval. Based on a retrieval-augmented generation framework, ICEAGE generates reliable and human-readable responses without requiring extensive domain-specific fine-tuning by applying unique indexing method for scientific datasets. Our experimental results demonstrate that ICEAGE significantly outperforms existing methods in terms of accuracy, throughput in both CPU-GPU and CPU-only environments.

ACM Reference Format:
Chenxu Niu, Wei Zhang, Mert Side, and Yong Chen. 2025. ICEAGE: Intelligent Contextual Exploration and Answer Generation Engine for Scientific Data Discovery. In *The International Conference on Scalable Scientific Data Management 2025 (SSDBM 2025), June 23–25, 2025, Columbus, OH, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3733723.3733731

1 Introduction

For data scientists and researchers, the ability to query and retrieve scientific data is essential for knowledge discovery and data analysis. However, as scientific data grows rapidly in complexity [12], both in terms of structure and volume, traditional search methods are struggling to meet the demands. Researchers now face difficulty in dealing with diverse data formats and complex metadata, which complicates the process of finding relevant information efficiently.

Traditional scientific data search methods have been developed and proposed to address the querying problem, such as BIMM [16], EMPRESS [17], SoMeta [36], JAMO [13], the SPOT suite [8], MIQS [42], PSQS [26] and IDIOMS [43]. These methods rely on databases that are queried through formal query languages like SQL or SPARQL or build metadata indexes to answer queries. Users should design their queries through supported database languages, and the returned results are typically highly reliable. Furthermore, it benefits from efficient query optimization of a database, ensuring good performance and efficient retrieval of results. However, these methods require researchers to have a steep learning curve, requiring them to master precise syntax and have a good understanding of the data schema. This makes traditional data search methods less intuitive, particularly for researchers without background knowledge of database and scientific data. Additionally, these methods have limited flexibility in query design, in which users are constrained by the rigid syntax and often struggle to capture the semantic meanings of their intended queries. Traditional data search methods also lack the ability to "understand" semantic meaning, making it difficult to retrieve relevant information when queries are not strictly structured.

Recent progress in natural language processing [23, 28] and large language models (LLMs), such as GPT-3 [4], LLaMA [37], BERT [18], CodeLLaMA [33], T5 [30], PaLM [7] shows the potential to process and query scientific data for researchers. Text-to-SQL [40, 44], code generation [38] methods, and retrieval-augmented generation (RAG) [19] methods have shown much better results in using human language to generate related structured data. However, it is not applicable to directly use these methods in scientific data domain. Text-to-SQL methods typically translate natural language into SQL statements but the quality of query results rely heavily on pre-defined database schemas and table relationships. But for scientific data, especially hierarchical data formats like HDF5, text-to-SQL approaches often struggle to produce accurate results since these methods can not find valid relationships. Code generation methods are designed to generate executable codes based on query input to retrieve the requested data. But in scientific data domain, these methods can create invalid or inefficient scripts, especially when dealing with complicated domain-specific structures. Moreover, code-generation methods often require extensive tuning or large-scale training datasets, which may not always be feasible. The third approaches is related to RAG, combines LLMs with external knowledge sources. In RAG, a retriever component locates relevant textual documents from a knowledge base, then an LLM uses those

SSDBM 2025, Columbus, OH, USA

ACM ISBN 979-8-4007-1462-7/25/06
https://doi.org/10.1145/3733723.3733731

retrieved passages to generate answers. While RAG has demonstrated potentials in open-domain question answering, its reliance on unstructured text retrieval can be problematic for scientific data, which often contains hierarchical relationships, multidimensional variables, and extensive metadata. Standard RAG pipelines typically focus on finding and summarizing textual information rather than interpreting complex metadata structures. As a result, they may not capture the organization and multi-level indexing that are common in scientific datasets.

To address the drawbacks mentioned above, we propose our solution ICEAGE, a Intelligent Contextual Exploration and Answer Generation Engine to provide natural language query capabilities for scientific data discovery. To build a robust solution for scientific data formats, ICEAGE begins by integrating a unique and hybrid indexing mechanism from scientific data and then tackles the complexity of query languages and data schemas by leveraging an RAG framework to enable natural language processing and generating queries that align with the structured metadata. Our method delivers reliable search results without requiring extensive fine-tuning.

Our major contributions with this work are:

- **A novel query engine:** ICEAGE provides a novel query engine to take natural languages as query inputs and generate natural languages as outputs.
- **Tailored for Scientific Data Discovery:** ICEAGE uses a unique indexing method that deals with the hierarchical structure and diverse data types of scientific metadata. The experimental results show that ICEAGE significantly improves the query correctness for scientific data.
- **Scalable and Efficient Architecture:** Unlike other LLM-based methods, we only apply inferences when generating answers. The evaluations demonstrate that ICEAGE delivers better query accuracy and throughput while maintaining low GPU dependency, making it scalable and cost-effective even in CPU-only environments.

We have performed extensive evaluations of ICEAGE over representative real-world scientific files, demonstrating both improved accuracy and higher throughput in query responses in CPU-only and CPU-GPU environments.

The rest of the paper is organized as follows. In section 2, we review the traditional metadata search solutions, large language model based solutions, and retrieval-augmented generated (RAG) based methods. We then propose our solution ICEAGE and show technical details in section 3. After demonstrating our comprehensive experiment results in section 4, we conclude in section 5.

2 Background and Motivation

In this section, we provide an overview of existing methods of searching scientific data, including traditional metadata-based search solutions and large language model (LLM) and retrieval-augmented generation (RAG) based methods.

2.1 Traditional Metadata Search Solutions

Many scientific applications using high-performance computing (HPC) systems nowadays need to deal with massive amounts of experimental, observational, and simulation datasets in scientific data, such as HDF5 [10], ASDF [11],netCDF [31], PnetCDF [32], Zarr [3], N5 [34], ROOT [5], and ADIOS-BP [21]. In the past decades, many scientific data management and search solutions have been proposed. Scientific data discovery methods can be broadly classified into three categories based on how metadata is stored and queried: pure dataset-scanning [16], database-powered [8, 13, 17], and index-based methods [25, 26, 42]. In the early ages, pure dataset-scanning methods scan scientific files and stored metadata in simple key-value patterns. They are straightforward but often inefficient for large-scale scientific datasets, as they require scanning entire datasets and data structure for each query. After that, many database-powered methods have been developed. In these approaches, metadata is always stored in relational database management systems (RDBMS), such as PostgreSQL [29] and SQLite [35], or document database, MongoDB [24] in particular. Querying in these solutions leverages the built-in functionality of databases. The metadata is first transformed to follow the pattern of the database's data model, tabular structures for relational databases or BSON for document databases, which enables query capabilities of the external databases. But the premise of leveraging these methods for researchers is to understand the base data schema and query languages of related databases. Modern index-based methods build different types of metadata-based indexes on key-value pairs and allow researchers to perform queries directly on these metadata indexes. Index-based methods are proven to be faster and more scalable than traditional pure dataset-scanning and database-powered methods, especially for large-scale scientific data formats.

Although these methods are widely used in scientific domain, they are still limited in handling the growing complexity and diversity of scientific datasets. These methods lack the ability to capture and understand semantic relationships, and they require technical proficiency, making them less accessible to non-expert researchers. That is why a more intuitive method is needed in this domain.

2.2 Large Language Model Based Solutions

Natural language processing, AI and Large Language Models (LLMs) have been developed for several years, especially after Transformer architecture [39] have been proposed. Typical examples are GPT-series (GPT-3 [4]), OPT [41], BERT [9], LLaMA-series [37], Mixtral [14], and Falcon [2]. The powerful tools for understanding and generating human language, offering a more intuitive way to process complex queries. LLMs can handle natural language query inputs and translate user questions into actionable instructions. In this process, researchers only need to design their queries with human languages. It is very ideal for scientific data discovery. Two primary approaches leveraging LLMs for querying scientific data have been proposed: text-to-SQL methods and code-generation based methods.

Text-to-SQL methods provide a way for researchers to issue queries in natural language, which are automatically converted into SQL queries for SQL-based databases. Typical examples are SQLNet [40] and Seq2SQL [44]. These methods provide a more intuitive way for researchers to perform queries without extensive database expertise. These models take human languages from researchers as inputs, map them to the query language schema, and generate valid SQL queries for researchers. **Code-generation based methods** take this automation further by translating user

queries into executable code, including C language, Python scripts and other domain-specific languages. For instance, *CodeLLaMA*[33] improves accuracy on coding tasks based on the natural language understanding capability based on LLaMA2[37],. CodeLLaMA also includes language-specific variants like *CodeLLaMA-Python*, which is designed specialized for Python writing use cases. With these methods, researchers do not need to learn the syntax of query language and complex structure of scientific datasets. They only need to issue queries with human language, and these methods can producing scripts to pre-process data and retrieve scientific data automatically for researchers.

However, both text-to-SQL and code-generation based methods have significant limitations when applied to scientific data discovery. First of all, most of these methods are only designed for one data format or rely on one special database, it is difficult to be applied directly to all heterogeneous or hierarchical scientific datasets. Second, these methods have better accuracy on simpler queries than complex queries. In other words, they have difficulty handling complex or nested queries that require a deep semantic understanding or the synthesis of multiple attributes, due to the simple mappings typically used between natural language and query structures. Third, code generation is highly dependent on the prompts from researchers. The generated results form code-generation based methods are not always accurate. Verifying, debugging, and maintaining codes can be time-consuming for researchers, and it is very challenging to deal with codes for large-scale scientific datasets or specialized scientific domains. To address these limitations, we need a more adaptable, user-friendly methods for researchers that can deal with scientific data without requiring extensive expertise in query languages or major code revisions.

2.3 Retrieval-Augmented Generation Methods

Most LLMs, such as CodeLLaMA, GPT-3 [4] or BERT [9], generate responses only based on patterns learned during training, which can become outdated or lack domain-specific knowledge. If we directly use these models on our scientific domain with some domain-specific terminologies, these models often generate some plausible-sounding but actually incorrect outputs, especially when queries fall outside the model's training scope.

Therefore, Retrieval-Augmented Generation (RAG) methods [19] have been proposed to address the limitation of dependence on static, pre-trained data. By retrieving contextually relevant information in real time, RAG improves accuracy and relevance, bridging the gap between generic pre-trained knowledge in training phrase and domain-specific knowledge or up-to-date requirements. As the name suggests, RAG has two phases: retrieval and generation. In the retrieval phase, RAG framework search for and retrieve snippets of information relevant to the prompt or question from researcher. It is based not only on the knowledge from the training step, but also on the external knowledge you define and feed to the framework. This assortment of external knowledge is appended to the prompt and passed to the language model. In the generative phase, the framework takes the augmented prompt and its internal representation of its training data to generate an engaging answer in human language tailored to the user in that instant. Typical frameworks like LangChain [6] and LlamaIndex [20] follow the phases.

Although RAG methods generally improve model performance by grounding outputs in external knowledge, they also introduce several challenges when applied to scientific datasets. First, RAG pipelines typically focus on unstructured text retrieval, making it challenging to capture the *hierarchical structures* and *complex data types* common in scientific data, for instance, nested attributes or multi-dimensional arrays. Second, even when relevant text or metadata is retrieved, it may fail to reflect the complex relationships among different levels of the dataset in many scientific domains. Finally, the volume and diversity of scientific data often demand specialized indexing strategies that go beyond simple text-based retrieval. As a result, while RAG methods improve general LLMs in terms of factual grounding and context utilization, additional capabilities are needed to accommodate the structural complexity and complex data types for scientific data discovery.

3 Methodology

In this section, we present the design and implementation of ICEAGE. As shown in Figure 1, ICEAGE consists of two parts: hybrid indexing construction and query process. In index construction step, ICEAGE extracts metadata, performs metadata normalization, and stores vectors. To serve query, we integrate a retrieval-augmented generation (RAG) framework to generate answers in natural language. This architecture preserves the structural information and semantics of metadata from scientific data, enabling accurate natural language query processing.

3.1 Hybrid Indexing

In ICEAGE, we need to start by building indexes from scratch. The first step is hierarchical traversal to capture the structure of each scientific file. Then to achieve context preservation, we need to record all path of each attribute. At last, we propose a unique way to handle mixed data types and build index on these attributes with their path for scientific data.

3.1.1 Hierarchical Traversal. In general, scientific data has tree-like structures, especially in formats such as HDF5 or NetCDF. Groups, subgroups, objects and attributes organize multiple layers of information in these scientific data. ICEAGE begins by parsing the metadata associated with objects and groups in a scientific file. This step involves traversing the hierarchical structure to extract both the individual data elements and the parent-child and path-dependent relationships between them. This structured parsing serves as the backbone for subsequent vectorization and retrieval processes, ensuring that all semantic and structural information of the scientific data are preserved.

To formally represent the metadata, we model each node in the hierarchy as a tuple:

$$N = \langle P, A, C \rangle,$$

where:

- P represents the full path from the root to the node, represented as a sequence $P = [p_0, p_1, \ldots, p_l]$ where each p_i denotes a group or attribute name.
- $A = \{(k_1, v_1), (k_2, v_2), \ldots, (k_n, v_n)\}$ is a set of key-value pairs representing the attributes of nodes.
- $C = [N_1, N_2, \ldots, N_m]$ is an ordered list of child nodes.

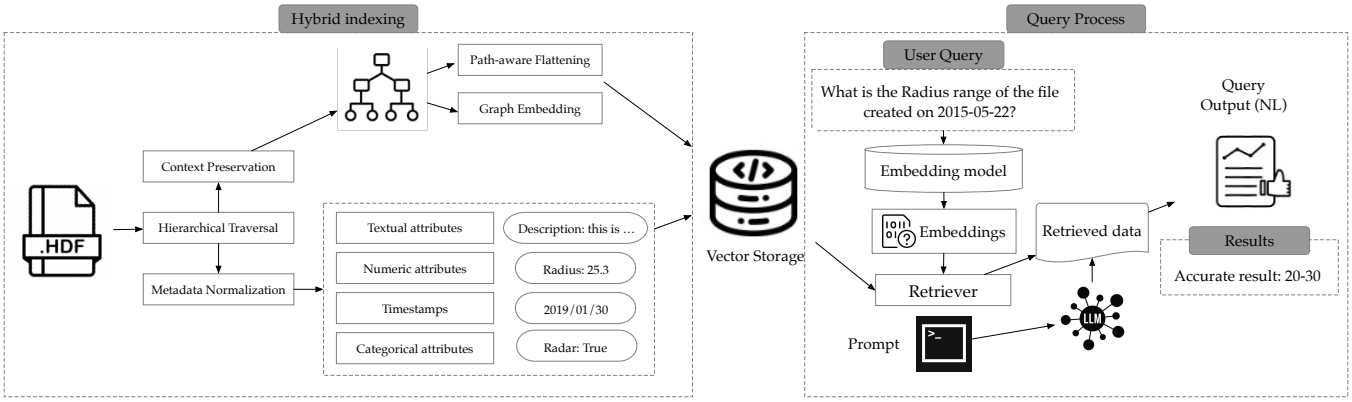


Figure 1: Overview of our ICEAGE system. It includes two important parts:hybrid indexing (data preparation) and query processing (user interaction).

Thus, the complete metadata repository M for a scientific file is defined as the set:

$$M = \{N_1, N_2, \ldots, N_T\},$$

where each N_i is constructed as described above.

To extract this structure from a scientific file, we implement the *Hierarchical Traversal* process that recursively explores every group, subgroup, each object and attribute in the scientific file. As shown in Algorithm 1, ICEAGE starts scanning the root node and traverses each child node from parent nodes in a depth-first manner. If a node is identified as a subgroup or it contains child nodes, the traversal continues; if it is a leaf attribute, its metadata tuple is recorded.

Algorithm 1 Hierarchical Traversal

```
function HIERARCHICALTRAVERSE(root)
    if root has no children then
        return { root }   ▷ Leaf node: record its metadata tuple
    end if
    nodeSet ← ∅
    for child in root.children do
        childSet ← HIERARCHICALTRAVERSE(child)
        nodeSet ← nodeSet ∪ childSet
    end for
    return nodeSet
end function
```

In scientific data formats, key-value pairs are usually used to store attributes of metadata. By traversing the hierarchical structure of the metadata, the metadata of each node is represented from any scientific data format and is defined in terms of tuples with metadata attributes. This method ensures that our approach can be applied flexibly across different implementations and systems while maintaining the integrity of the original hierarchical relationships.

3.1.2 Context Preservation. After parsing the hierarchical structure of scientific data, preserving the complete contextual information is critical for accurate query processing. Our approach retains the full path strings, for instance "`/ExperimentA/Step1/Temperature`", and metadata for each node.

First, preserving the full paths allows the use of partial queries to retrieve attributes from different levels of the hierarchy. For example, if one wants to query "`/ExperimentA/*/Temperature`", it would not be necessary to input the exact path of the attribute. After the preservation step, ICEAGE supports context-aware querying for scientific data since each attribute is linked to its parent groups or objects. Further queries can consider both local attributes and overall metadata. Keeping the hierarchical structure enables the system to skip over parts of the data that do not meet certain constraints, such as specific time ranges or domain criteria.

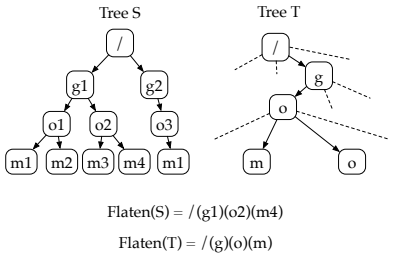


Figure 2: Overview of the an example of context preservation.

3.1.3 Handling Mixed Data Types. In general, metadata of scientific files contain a variety of data types, such as textual descriptions, numerical measurements, timestamps, temporal data, and categorical values. ICEAGE uses specialized processing methods to handle each type effectively as following.

- Textual Descriptions: Processed using state-of-the-art natural language processing models to get embeddings.
- Numerical Measurements: Normalized and incorporated into the overall feature vector, ensuring that quantitative differences are preserved.
- Categorical Data: Categorical attributes are encoded using techniques of one-hot encoding embeddings to maintain meaningful distinctions between categories.
- Timestamps/Temporal Data: Timestamps are normalized into relative temporal features, enabling the system to understand time-dependent relationships.

This encoding method ensures that ICEAGE effectively captures the diverse metadata types of scientific metadata. Some real-world data examples are shown in the following Table 1.

Data Type	Processing Method	Example
Text	Text embedding models	description="Spatial Data"
Numeric	Standardize and concatenate as features	scale=0.1 → [0.24]
Timestamps	Convert to time delta (relative to root) and normalize	timestamp=1630454400 → $\Delta t = 0.75$
Categorical	One-Hot encoding	data_type="float32" → [0,1,0]

Table 1: Data type processing methods and examples.

3.1.4 Vectorization Strategies for Hybrid Indexing. After processing the scientific data in the last step, we need to convert the parsed metadata into vector representations for further storage and querying. To process the hierarchical structure and mixed types of scientific metadata, ICEAGE applies two vectorization strategies:

Path-Aware Flattening Vectors. Unlike the traditional vectors or word embeddings, the path information is equally important for scientific data discovery. After the hierarchical traversal and context preservation process, this technique is used to record the path information of metadata attributes and convert the path information to vectors by a pre-trained text embedding model. In our implementation of ICEAGE, we choose the text-embedding-3-large model [27] from OpenAI. For example, suppose we have a metadata attached to a dataset, and it is described with a string that encodes both its attributes and its location "/Experiment1/SampleA/MeasurementX: Temperature=25°C". Then the embedding model flattens the information to a string in the right order first, and generates vectors with the string including recorded positional context. This approach preserves the contextual hierarchy so that nodes with similar attributes but different locations are distinguished.

Multi-Granularity Chunked Vectors. The previous technique is about how to deal with the path information, and this is used to record the content information of the metadata attributes. In traditional embedding methods, only words or sentences can be converted into vectors. But in scientific data, not only the content of words and attributes is valuable, the content of a subtree is also valuable. To better record the content, scientific metadata needs to be segmented into chunks of varying granularity, ranging from individual attributes to complete subtrees. That is why we need to use the multi-granularity chunked vector technique. ICEAGE applies this strategy to reflect different granularity. Each chunk is vectorized separately according to granularity, and it allows the system to retrieve relevant information at varying levels of detail and combine fine-grained and coarse-grained vectors to form a comprehensive representation.

3.1.5 Vector Storage. The generated vectors from last step are stored as index in FAISS [15]. FAISS, which stands for Facebook AI Similarity Search, is an open-source library developed by Facebook AI Research designed for efficient similarity search and clustering of dense vectors. FAISS uses hierarchical navigable small-world (HNSW) graphs [22] to store vectors of different dimensions. This storage mechanism enables FAISS of ICEAGE to efficiently retrieve the most relevant metadata vectors to support further query processing. Additionally, FAISS is suitable to be applied in HPC systems, since it supports both CPU and GPU implementations. In the GPU-accelerated environment, the storage and query performance will be boosted to support real-time queries.

3.2 Query Service

In ICEAGE, query processing is built on a Retrieval-Augmented Generation (RAG) framework to process natural language inputs and generate answers in human language.

3.2.1 User Query Interface. The user query interface in ICEAGE is designed to take any natural language as query inputs. When a user submits a query in human language, ICEAGE will process extract key elements from the inputs, like tokenization process in natural language processing. Typical key elements for scientific data discovery are words or strings, numbers, temporal data, timestamps and categorical data. These elements are then mapped onto the structured metadata and vector representations by the same LLM model and then stored in the vector storage FAISS.

3.2.2 Advanced Retrieval Mechanism. After getting and preprocessing the query, ICEAGE processes a retrieval-rugmented process, just like a typical RAG framework. In the first step, ICEAGE compares the generated query embedding against the contextual vectors from FAISS with a similarity matching method. In our case, we use cosine similarity to calculate the similarity score. Once potential matches are found, the process moves to the second step: structural retrieval. ICEAGE uses a path-aware flattening process to achieve the goal. For each candidate match, ICEAGE reconstructs the full hierarchical path associated with the metadata. This mechanism preserves the context of the location of a attribute or a subtree in the overall data structure. This step is essential for distinguishing between nodes that might have similar content but are located in different parts of the dataset.

This two-step retrieval mechanism is used to capture both context and structural information, "understand" complex queries, and generate more accurate responses.

3.2.3 Prompt Engineering for Scientific Data. It is always challenging to deal with the high complexity of scientific data. To improve the ability of ICEAGE to process scientific data, our prompt templates are tailored to guide LLM model in RAG toward producing responses. Our prompt design is described as following:

(1) **Hierarchical Context Instructions:** The first and the most important design of the prompt is to give a clear instruction to the model about the importance of hierarchical structure of scientific data. For example, the prompt should include a statement such as "You need to deal with scientific data. Consider the following hierarchical paths and metadata attributes when generating your answer." This makes the model to consider both the context and structural information for the query.

(2) **Data Type and Unit Reminders:** To ensure that numerical values and categorical data are handled correctly, the prompt includes annotations about the data types and units. For example, we should write the prompt to include "All temperature values are provided in Kelvin, and any conversions must maintain these units." The instructions help the model to "understand" data better and avoid misinterpretation of quantitative data.
(3) **Domain-Specific Terminology:** The most tricky part of scientific data is terminologies. In different domains, concepts and terminologies have completely different meanings. In general, fine-tuning technique is needed to be applied without a RAG framework. So the following prompt is used to enhance with domain-specific terminologies. For example, we can add some description information in the prompt: "Radius refers to the measured distance in millimeters, and SampleID uniquely identifies each experimental specimen."
(4) **Example-Based Guidance:** RAG is good at learning examples and do similar things for you. It would be better to includes examples that demonstrate the desired format in the response in your prompts. For example, ICEAGE will input a sample response that maps a query to specific hierarchical paths and numerical data.

After designing the prompts with these elements mentioned above, we ensure that ICEAGE is continuously reminded of the uniqueness of scientific data. This design helps ICEAGE to "understand" both query inputs and scientific data. Therefore, it will improve the quality of generated responses, and minimize the risk of hallucination.

3.2.4 Answer Generation in Natural Language. The last step of our method is to generate results in human language for users. In a typical RAG framework, a sequence-to-sequence transformer model is used to processes the query and retrieve context in natural language. But it is not applicable in scientific data domain. ICEAGE modifies the standard RAG pipeline as follows:

(1) **Context Injection**: The prompt mentioned in last step is constructed to include user-query details and the path-aware metadata attributes. So the generated tokens by the LLM in ICEAGE is verified and have better accuracy.
(2) **Inline Annotation Handling**: Since the structural and context information of metadata is annotated by users. The generated response can incorporate them directly into the final pattern in natural language.
(3) **Response Generation:** After "understanding" queries and metadata, ICEAGE utilizes LLM to generate responses, which are tightly aligned with the underlying structured data.

As a result, the final output is both *coherent* and *tightly aligned* with the metadata structure of scientific files. It offers more reliable and domain-specific responses than a purely generative approach.

Furthermore, since each query can be processed independently, our method can be easily parallelized in HPC systems. It allows multiple queries to be handled simultaneously and compared independently. As a result, it will improve the query performance and overall throughput for scientific data discovery.

3.2.5 Summary of RAG Integration. With the techniques mentioned above, ICEAGE can "understand" user queries and retrieves related metadata with its path and generate answers in natural languages. It allows the system to capture structural and semantics that are critical for advanced scientific applications.

4 Evaluation

In our evaluation, we compare ICEAGE with a representative LLM solution LLaMA3, a code-generation based method CodeLLaMA and a typical RAG framework LlamaIndex. For a fair comparison, we evaluated the query accuracy, throughput, and GPU dependency of these four methods on the same real-world scientific datasets.

4.1 Experimental setup

We conducted our evaluation of ICEAGE on the Red Raider cluster Texas Tech University's High Performance Computing Center (HPCC). The cluster consists of three distinct parts: the Nocona CPU partition with 467 CPU nodes and the Matador and Toreador GPU partitions with 10 Nvidia A100 GPU nodes. The separate partitions provide us the CPU-only and CPU-GPU environments for LLM inferencing to handle large-scale scientific data queries.

4.1.1 Datasets and Benchmarks. Our dataset contains a set of 1,987 real-world HDF5 files coming from the National Snow and Ice Data Center (NSIDC) [1]. To have a fair comparison, We designed a synthetic benchmark consisting of 350 queries based on the metadata of these files for the following evaluation. We manually annotated the ground truth for each query to further evaluate the correctness of the query.

As shown in Table 2, these queries have been categorized into two types: simple queries and nested queries.

- Simple Queries (210): These are simple, one-step queries that can get answer from one attribute directly. An example is "the author of the second file,", only one piece of metadata is directly retrieved without additional processing.
- Nested Queries (140): These require multi-step processing or building connections between different attributes in the metadata. For instance, "how many files have a Radius range less than 35-45" involves searching and counting, while "what is the Radius range of files authored by 'David Green" requires connecting multiple metadata attributes to produce the result.

Query Type	Example of Natural Language Query
Simple	Who is the author of the second file
Simple	how many files in the folder
Nested	What is the Radius of files authored by 'David Green'
Nested	how many files have a Radius range less than 35-45

Table 2: Representative examples of simple and nested queries drawn from our benchmark.

4.1.2 Baseline and Evaluation Metrics. To evaluate the performance of ICEAGE, we compare it with three baseline models: LLaMA3, CodeLLaMA, and LlamaIndex. We selected these baselines for the following reasons:

- **LLaMA3**: As one of the state-of-the-art large language models, LLaMA3 is open-sourced and now widely used to provide strong generative capabilities and serves as a representative of pure generative approaches in our scenario.
- **CodeLlama**: This model is specifically optimized for code generation based on the LLaMA family series. In HPC environment, CodeLlama can generate SQL statements or C/Python codes from natural language prompts to query scientific data.
- **LlamaIndex**: As a representative RAG framework, LlamaIndex serves natural language queries for structured data through its flexible data ingestion and structured data querying capabilities.

We evaluated the system using three key metrics, including:

- **Query Accuracy**: The quality of the returned results is the most important metrics of a query service. Based on the ground truth we annotated and the generated responses by each method, our target is to find the best method to minimize false positives and maximize relevant data retrieved.
- **Throughput**: We evaluated the number of queries processed per second (QPS) under different workloads. The average response times is used to determine the performance of ICEAGE and the baseline methods when subjected to varying query loads, highlighting the system's ability to maintain high performance across increasing numbers of processes.
- **GPU Dependency**: We assessed how each model relies on GPU resources to speedup inference and query performance. The evaluation demonstrates how ICEAGE can scale and perform well in CPU-only and CPU-GPU environments. The results provide insights into its efficiency in handling scientific data discovery in all kinds of HPC systems.

4.2 Query Accuracy

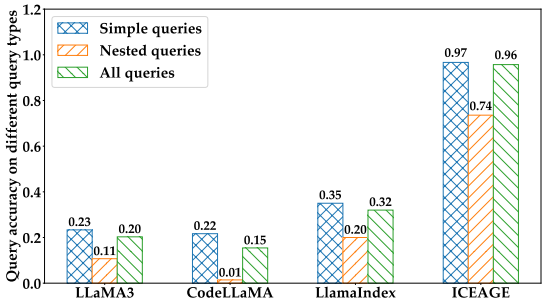


Figure 3: Comparison of query accuracy across different query types for LLaMA3, CodeLLaMA, LlamaIndex and ICEAGE. ICEAGE demonstrates significantly higher accuracy in both simple and nested queries, with an overall query accuracy of 0.96, outperforming other models, which show lower accuracy in handling nested and overall queries.

As we all know, LLMs occasionally produce inconsistent outputs because of their probabilistic nature. To address the uncertainty and variability of the responses generated by LLMs, we utilized a comprehensive evaluation strategy. To achieve this, we run the benchmark queries 100 times with four of these methods. The intensive results allowed us to capture the variability of the responses and compute more reliable accuracy metrics based on the 100-time trials. By collecting data over multiple runs, we could calculate the average accuracy across all rounds to provide a more accurate and fair assessment of the query accuracy.

4.2.1 Average Accuracy for Different Query Types. Figure 3 shows the average query accuracy for each model in three query categories: simple, nested, and overall. The bar heights represent the mean accuracy calculated over 100-time repeated query trials. As expected, all models perform better on simple queries than on nested queries, reflecting the increased difficulty of multi-step and hierarchical lookups. The results show that ICEAGE achieves the highest accuracy for both simple and nested queries, demonstrating robust performance across diverse levels of complexity. In contrast, while CodeLLaMA performs competitively on simple queries, its accuracy drops significantly on nested queries, partly due to some generated code failing to execute. Similarly, although LlamaIndex incorporates a retrieval-augmented generation framework, it struggles with nested queries because it cannot effectively handle the hierarchical structure and mixed metadata types inherent in scientific data.

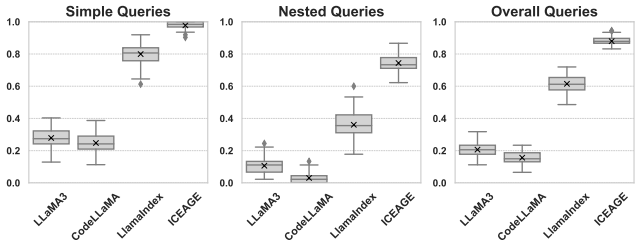


Figure 4: Box plot of query accuracy distributions for simple, nested, and overall queries across LLaMA3, CodeLLaMA, LlamaIndex, and ICEAGE. The plot is used to highlight the variability and stability of each model's performance.

4.2.2 Variability and Distribution of Query Accuracies. The variation and distribution of all query results across the 100-time queries is shown in the box plot (Figure 4). It indicates that ICEAGE demonstrates a narrower range of variability and a much higher median than all other methods. ICEAGE also achieves high scores and average values tightly clustered around high scores. It means that ICEAGE achieves higher accuracy and maintains consistent performance over all query types of the benchmark than all other methods.

In contrast, Figure 4 shows that LLaMA3, CodeLLaMA and LlamaIndex have wider ranges of variability. The median of the results for these models is lower than ICEAGE and with larger interquartile ranges. The results are even worse on nested queries. The presence of outliers emphasizes the uncertainty and the potential for errors when using these methods for scientific data queries. Overall, the

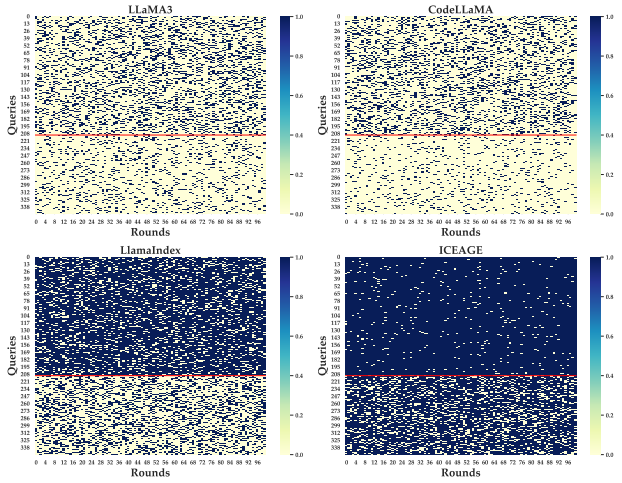
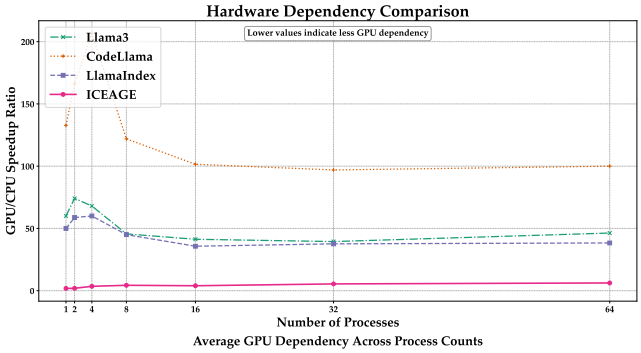


Figure 5: Heat map of per-query accuracies over 100 repeated runs for each model. Darker colors indicate higher accuracy, revealing both the stability and variability of model performance across individual queries.



Average GPU Dependency Across Process Counts

Model	Average GPU/CPU Speedup
Llama3	53.54x
CodeLlama	132.27x
LlamaIndex	46.49x
ICEAGE	3.88x

Figure 6: GPU dependency analysis comparing GPU/CPU speedup ratios for LLaMA3, CodeLLaMA, LlamaIndex, and ICEAGE in parallel computing across various processes. Lower speedup numbers indicate less dependency on GPU acceleration. ICEAGE shows an average speedup of only 3.88×, significantly lower than the baselines, demonstrating its lower dependency on GPU and better efficiency in CPU-only environments.

results show that the generated responses of all other three methods are more prone to fluctuation, highlighting the instability in their ability to handle complex queries.

4.2.3 Per-Run Accuracy Stability. To test the stability, we run the benchmark 100 times for each method. Figure 5 presents a heatmap that visualizes the accuracy of each query over 100 repeated rounds. In this figure, the vertical axis represents individual queries, while the horizontal axis shows the number of rounds. When it matches, we used darker color to mark. So darker area indicate higher accuracy, and lighter cells denote lower or more inconsistent results. The top half of the heatmap records the results of 240 simple queries, which are generally easier to answer correctly. This indicates that these queries are consistently answered correctly across the repeated trials. In contrast, the bottom half represents the nested queries, where the accuracy is lower and more variable, as shown by the lighter coloration.

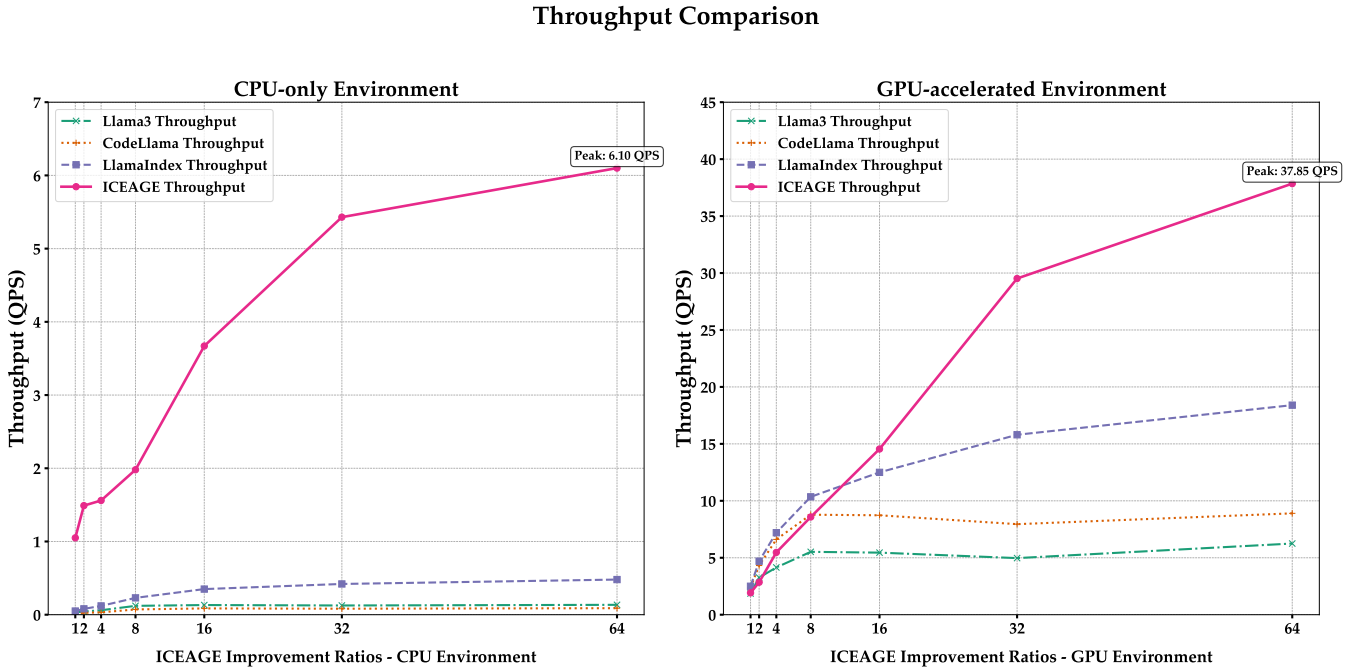
Overall, the heatmap figure clearly demonstrates that ICEAGE maintains high stability and reliability on both simple queries and nested queries. These findings show that ICEAGE is more robust and accurate in handling both simple and nested queries, and the generated results by our engine are more reliable.

4.2.4 Summary of Query Accuracy. In summary, these three figures demonstrate that ICEAGE consistently outperforms the baseline models in query accuracy and stability on our benchmark. Figure 3 shows that all models perform better on simple queries than on nested queries. But ICEAGE still maintains the highest average accuracy across both query types of the benchmark. The experiment further proves that ICEAGE has a higher median accuracy with low variability over 100 repeated trials, but the baselines show greater fluctuations. The heat map shows that ICEAGE achieves more stable and accuracy results in each round, while the baseline models display more scattered results both in simple and nested queries. Overall, these experimental results show that ICEAGE has the best reliability and performance in handling all kinds of queries over scientific data.

4.3 GPU Dependency

In some HPC systems, GPU nodes and resources are very limited. Researchers may need to wait for a long time or even have no access to use GPU for acceleration. Evaluating GPU dependency is important for researchers to understand the improvement by GPU acceleration and the scalability and efficiency of query systems in CPU-only environment. Figure 6 compares the GPU/CPU speedup ratios for LLaMA3, CodeLLaMA, LlamaIndex, and ICEAGE in CPU-only and CPU-GPU environments. The speedup ratio is calculated as the ratio of throughput with GPU acceleration to throughput in a CPU-only environment. The result is to see how each model benefits from additional GPU resources. Lower speedup values indicate less reliance on GPU acceleration.

Actually, the index construction step and the vector storage step of ICEAGE is done without relying on GPUs. In theory, only the inference step of the large language model for generating outputs in human language is GPU-dependent. This design is different from the other methods. They all rely extensively on GPU acceleration to build indexes and LLM inference. Consequently, ICEAGE shows a substantially lower average GPU/CPU speedup of 3.88 times. The results show that ICEAGE maintains strong performance even in CPU-only environments. As the number of processes increases, ICEAGE continues to deliver stable performance, but the other three methods benefit more dramatically from GPU acceleration. These findings highlight the versatility and suitability of ICEAGE for a wide range of computational HPC environments.



Processes	ICEAGE/L3	ICEAGE/CL	ICEAGE/LI
1	33.87x	58.33x	21.00x
2	33.86x	57.31x	18.62x
4	25.57x	48.75x	13.00x
8	16.36x	27.50x	8.61x
16	27.80x	42.67x	10.49x
32	43.10x	66.22x	12.93x
64	45.19x	68.54x	12.71x

Processes	ICEAGE/L3	ICEAGE/CL	ICEAGE/LI
1	1.03x	0.80x	0.77x
2	0.87x	0.66x	0.61x
4	1.32x	0.83x	0.76x
8	1.55x	0.98x	0.83x
16	2.67x	1.67x	1.16x
32	5.94x	3.71x	1.87x
64	6.06x	4.25x	2.06x

Figure 7: Throughput comparison of LLaMA3, CodeLLaMA, LlamaIndex, and ICEAGE under CPU-only and GPU-accelerated environments. L3: LLaMA3, CL: CodeLLaMA, LI: LlamaIndex.

4.4 Throughput

Throughput is another critical metric for assessing the overall performance of query engines, particularly when dealing with scientific data and complex queries in scientific applications. Figure 7 shows the query processing throughput comparison of all the methods measured in both CPU-only and GPU-accelerated HPC environments.

The left half of Figure 7 shows the results in the CPU-only environment. We tested 1, 2, 4, 8 and 16 processes for all these four methods and the result shows that ICEAGE consistently improves throughput compared to all baseline models in each process setting. The peak throughput achieved by ICEAGE is 6.10 QPS with 64 processes and is 6.06 times faster compared to LLaMA3. The right half of Figure 7 shows the results in the GPU-accelerated environment. GPU is more powerful to serve the inference of LLM, so in GPU environment, the peak throughput of ICEAGE has been boosted to 37.85 QPS. These results show the significant performance gains through GPU utilization. Similarly to the results in CPU-only environment, the results show that ICEAGE consistently improves throughput compared to all baseline models again for 16, 32, and 64 processes settings.

Overall, the experimental results demonstrate that ICEAGE is more efficient in processing queries under high-load conditions in both CPU-only and CPU-GPU environments. The scalability of ICEAGE is best compared to other methods, especially with GPU acceleration. ICEAGE delivers better throughput than all baseline methods in a significant ratio.

4.5 Overall Evaluation Summary

In summary, our comprehensive experimental results demonstrate that ICEAGE outperforms all baseline models in query accuracy, throughput, and GPU efficiency. ICEAGE consistently achieved higher accuracy for both simple and nested query types than other baseline methods on the benchmark. The results generated by our methods are more reliable compared to LLaMA3, CodeLLaMA, and LlamaIndex. The performance experiment shows that ICEAGE achieves higher throughput in parallel environments with GPU acceleration. Considering that some HPC systems only have limited GPU resources, we conducted the experiments again in a CPU-only environment. The experimental results show that ICEAGE has a lower GPU/CPU speedup ratio compared to other methods, and it indicates that ICEAGE relies less on GPU resources. In summary,

these results prove that ICEAGE is a scalable and more efficient solution for scientific data discovery.

5 Conclusion and Future Work

In this paper, we present a novel intelligent context exploration and answer generation engine (ICEAGE) for scientific data discovery. It integrates a RAG framework and build indexes based on context preservation and metadata normalization. Our extensive evaluations demonstrate that ICEAGE outperforms existing LLM-based methods in query accuracy, throughput, and GPU dependency for scientific data. To our knowledge, this is the first study to offer natural language query service for scientific data discovery and has the potential to enhance data management functionality and improve the efficiency of searching and discovering scientific data in the HPC community.

References

[1] [n.d.]. National Snow and Ice Data Center, a part of CIRES at the University of Colorado Boulder. https://nsidc.org/.

[2] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. 2023. The falcon series of open language models. *arXiv preprint arXiv:2311.16867* (2023).

[3] Francesc Alted, Martin Durant, Stephan Hoyer, John Kirkham, Alistair Miles, Mamy Ratsimbazafy, Matthew Rocklin, Vincent Schut, Anthony Scopatz, and Prakhar Goel. 2018. Zarr. https://zarr.readthedocs.io.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] R. Brun and F. Rademakers. 1997. ROOT: An Object Oriented Data Analysis Framework. *Nucl. Instrum. Meth.* A389 (1997), 81–86. https://doi.org/10.1016/S0168-9002(97)00048-X

[6] Harrison Chase. 2022. *LangChain.* https://github.com/langchain-ai/langchain

[7] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[8] Tull Craig E., Essiari Abdelilah, Gunter Dan, et al. 2013. SPOT Suite. http://spot.nersc.gov/.

[9] 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, volume 1 (long and short papers)*. 4171–4186.

[10] Mike Folk, Albert Cheng, and Kim Yates. 1999. HDF5: A File Format and I/O Library for High Performance Computing Applications. In *Proceedings of supercomputing*, Vol. 99. 5–33.

[11] P Greenfield, M Droettboom, and E Bray. 2015. ASDF: A New Data Format for Astronomy. *Astronomy and Computing* 12 (2015), 240–251.

[12] The HDF Group. 2018. The HDF Group. https://www.hdfgroup.org.

[13] Joint Genome Institute. 2013. The JGI Archive and Metadata Organizer(JAMO). http://cs.lbl.gov/news-media/news/2013/new-metadata-organizer-streamlines-jgi-data-management.

[14] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[15] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[16] Daniel Korenblum, Daniel L. Rubin, Sandy Napel, Cesar Rodriguez, and Christopher F. Beaulieu. 2011. Managing Biomedical Image Metadata for Search and Retrieval of Similar Images. *J. Digital Imaging* 24, 4 (2011), 739–748.

[17] Margaret Lawson and Jay Lofstead. 2018. Using a Robust Metadata Management System to Accelerate Scientific Discovery at Extreme Scales. In *Proceedings of the 2nd PDSW-DISCS '18.* https://doi.org/10.1109/PDSW-DISCS.2018.00005

[18] M Lewis. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).

[19] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[20] Jerry Liu. 2022. *LlamaIndex.* https://doi.org/10.5281/zenodo.1234

[21] Jay F Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. 2008. Flexible I/O and Integration for Scientific Codes through the Adaptable I/O System (ADIOS). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments.* ACM, 15–24.

[22] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781* (2013).

[24] MongoDB. 2018. MongoDB. https://www.mongodb.com.

[25] Chenxu Niu, Wei Zhang, Suren Byna, and Yong Chen. 2022. Kv2vec: A distributed representation method for key-value pairs from metadata attributes. In *2022 IEEE High Performance Extreme Computing Conference (HPEC).* IEEE, 1–7.

[26] Chenxu Niu, Wei Zhang, Suren Byna, and Yong Chen. 2023. PSQS: Parallel Semantic Querying Service for Self-describing File Formats. In *2023 IEEE International Conference on Big Data (BigData).* IEEE, 536–541.

[27] OpenAI. 2023. .

[28] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP).* 1532–1543.

[29] PostgreSQL. 2018. PostgreSQL. https://www.postgresql.org.

[30] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[31] Russ Rew and Glenn Davis. 1990. NetCDF: An Interface For Scientific Data Access. *IEEE computer graphics and applications* 10, 4 (1990), 76–82.

[32] Rob Latham, Rob Ross, Rajeev Thakur, Kui Gao, Alok Choudhary, Wei-keng Liao, Jianwei Li and Bill Gropp. 2010. Parallel NetCDF. http://trac.mcs.anl.gov/projects/parallel-netcdf.

[33] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[34] S Saalfeld. 2017. N5: Not HDF5. https://github.com/saalfeldlab/n5.

[35] sqlite.org. 2017. SQLite. https://sqlite.org.

[36] Houjun Tang, Suren Byna, Bin Dong, Jialin Liu, and Quincey Koziol. 2017. SoMeta: Scalable Object-centric Metadata Management for High Performance Computing. In *2017 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE, 359–369.

[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[38] Immanuel Trummer. 2022. CodexDB: Synthesizing code for query processing from natural language instructions using GPT-3 Codex. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2921–2928.

[39] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[40] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436* (2017).

[41] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).

[42] Wei Zhang, Suren Byna, Houjun Tang, Brody Williams, and Yong Chen. 2019. MIQS: Metadata Indexing and Querying Service for Self-describing File Formats. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* 1–24.

[43] Wei Zhang, Houjun Tang, and Suren Byna. [n.d.]. IDIOMS: Index-powered Distributed Object-centric Metadata Search for Scientific Data Management. ([n. d.]).

[44] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103* (2017).