Smart Science Needs Linked Open Data with a Dash of Large Language Model and Extended Relations

Hasan M. Jamil
University of Idaho, USA
jamil@uidaho.edu

ABSTRACT

Quality scientific inquiries depend on access to data distributed over the entire globe. Linked open data (LOD) and FAIRness play major roles in ensuring access to data scientists need to answer interesting questions. However, a data model and a query language to compute the responses to complex scientific inquiries remain outstanding. As the recent emergence of large language models (LLM) reshape how we interact with machines, an intriguing prospect of posing scientific inquiries to smart machines suddenly appears realizable in which a natural language ChatBot is empowered with a LOD knowledgebase as its data source. In this paper, we introduce an LLM interpreter, called *ProAb*, that aims to answer natural language scientific queries using a structured query language called *Needle* in which the LOD is viewed as a set of tables. We discuss the contours of ProAb, present its preliminary design, and highlight its salient features using an illustrative example.

CCS CONCEPTS

• **Information systems → Data extraction and integration**; **Information retrieval query processing**; *Search interfaces*; • **Human-centered computing → Human computer interaction (HCI)**; **Natural language interfaces**; • **Computing methodologies → Artificial intelligence**; **Machine translation**; • **Applied computing → Life and medical sciences**.

KEYWORDS

Large Language Model, Intelligent User Interface, Extended Relational Model, Structured Query Language, Query Processing.

ACM Reference Format:

Hasan M. Jamil. 2024. Smart Science Needs Linked Open Data with a Dash of Large Language Model and Extended Relations. In *The 2024 ACM SIGMOD/PODS International Conference on Management of Data (SIGMOD '24), June 9–15, 2024, Santiago, Chile.* ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/XXXXXXXXXXXXXXX

1 INTRODUCTION

Some of the major barriers to using linked open data (LOD) include their global distribution, administrative autonomy, heterogeneity, volume, number, volatility, and accessibility. Since the author or the owner of the data has the complete autonomy over the design of access protocol and structure, their discovery, usage and the application orchestration using them remained users' responsibility. More traditional approaches to scientific inquiries relied on application building using local replicated data stores of data from the LOD repositories as views. Their frequent curation at the source invited view maintenance hurdles [38], local storage of voluminous data needed elaborate management strategies and care [11, 39], and heterogeneity of multiple data sources posed integration hurdles [16, 33]. Architecting workflows and analytics also demanded sophisticated computational expertise that many domain scientists lacked.

Though systems such as Galaxy [2] and Taverna [47] significantly improved the state of data integration and workflow formulation to aid scientific inquiries, their usability is limited – they cover a limited class of databases meeting platform restrictions. While some systems, such as Tavaxy [1], considered extending Galaxy and Taverna's reach by fashioning their functional fusion, they still cover only a very small number of available and accessible databases. The recent push for FAIR (findability, accessibility, interoperability, and reusability) [45] compliance of LOD is also having an impact on scientific inquiries although the compliance remains pretty low [14]. Since initial coinage of the term Scientific Inquiries [17] around early eighties in the context of online searching, it has been primarily explored in education as a pedagogy for science learning. However, a few recent research has expanded its reach into scientific computation as well.

We believe that the practice of traditional scientific computing could be enlarged to include online scientific inquiries by bringing all the accessible online LODs as a scientific knowledgebase in ways similar to intelligent personal assistants for learning [34]. In fact, a recent system, called the BioNursery [23, 24], we have introduced a similar experimental digital assistant for scientific inquiries for serious life sciences research. In BioNursery, we have exploited the power of large language models (LLM) for the generation of a computational plan using online resources. The suggested plan is then mapped to a workflow involving online databases for real time computation of the response of the inquiry.

The approach in BioNursery brings both novelty and significant challenges into focus. It is novel because it frees users from a fixed traditional GUI like interface that only could compute preconceived science, and cannot really support open ended inquiries. BioNuresery also conceptually is able to query any and all databases on the LOD network. However, the challenges it faces are many and they are enormous in several instances. The good news is that these challenges are well-known, well understood, and solutions for many are also available at varying levels of sophistication that can

SIGMOD '24, June 9–15, 2024, Santiago, Chile

ACM ISBN XXXXXXXXXXXXXXXXX…$15.00
https://doi.org/XXXXXXXXXXXXXXX

be leveraged in an implementation. Our goal in this paper is to introduce a theoretical model, called ProAb (stands for programming in the abstract), for a possible implementation of an engine similar to BioNursery, and discuss possible solutions for the theoretical components that are still outstanding.

2 RELATED RESEARCH

Scientific inquiry has been a major focus in education and learning [8, 9, 28, 40] for a while. In contrast, without the banner of scientific inquiry, scientists have been conducting scientific inquiries in various forms [12, 13, 26]. The difference perhaps is that scientific inquiries of the latter forms are designed meticulously with significant efforts and substantial resources that sets them apart from spontaneous inquiries we are familiar with.

Spontaneous scientific inquiries is difficult because they usually need specific or relevant data sets, tools, and processes that are based on sound scientific knowledge or hypotheses often unavailable to traditional conversational search engines [10, 30, 35], or more recently, to chatbots [3, 41, 49]. But the emergence of LLMs such as ChatGPT is transforming science learning, and scientific inquiries [18, 36]. The sophistication with which LLMs respond to inquiries are also changing. It is expected that they will soon integrate computations, AI and machine learning to respond to queries. In BioNursery [23, 24], LLM was used to generate hypothesis and computational strategies, and then the strategies were mapped to executable workflow codes in a language called Needle [25]. The process was augmented with additional knowledge about internet resources and their access methods.

In this paper, we design a higher level conversational inquiry framework that is able to shoulder the responsibilities of generating the resource descriptions needed to access the online databases and resources autonomously and construct the computational pipelines without human assistance and interact in ways ChatGPT does. The challenges are huge and not all the technologies are at hand. Thus the goal remains to discuss the contours of the scientific inquiry system ProAb and identify opportunities for research and discuss the progress so far.

3 NATURAL SCIENTIFIC INQUIRIES

We envision ProAb as a natural language conversational question answering system that is similar to a chatbot in which we expect users to ask questions such as

> *Q_1: Find all human genes implicated for both obesity and teratozoospermia (male infertility).*

or,

> *Q_2: Is it possible that the gene PLP1 could be involved in demyelination in central nervous system severe enough to cause Parkinson's disease?*

While question Q_1 could be answered by looking up information in a suitable database, question Q_2 is more challenging and will most likely require application of analysis tools on multiple data sets retrieved from relevant databases. However, in both instances, especially for Q_1, the simplicity is deceptive and the process involved are truly complex as discussed below.

3.1 Overall Query Answering Process

Computing a response to questions such as the ones above involves three principal steps. First, each query needs to be understood in terms of a computational process – a finite set of conceptual steps must be identified that could logically be computed over a set of databases and potential use of a set of analysis tools. Second, the exact set of databases and tools needs to be selected and computational pipeline needs to be determined. Third, a precise computational code needs to be generated as an implementation of the hypothesized workflow. Finally, the generated code will be executed to return a response. Additionally, if the response needs to be presented as a summarized text using audio-visual enhancements, an auxiliary process may follow to do so.

The four broad steps outlined above are distilled from a logical set of actions a human expert will follow to solve problems of this class of queries. These steps are agnostic about the domain of application, and the specificity of an inquiry. However, the process expects the availability or discovery of the knowledge of the intended semantics of each query, databases, and analysis tools. The process also expects that it may fail at any of these steps and return no response. Even when an executable code could be generated, it is entirely possible that the code may not execute properly due to syntax or semantic errors, or gaps in the knowledge behind the generation of the code.

3.2 Computational Pipeline Construction

Computational pipelines need to respond to the queries Q_1 and Q_2 introduced in Sec 3 may take multiple different forms depending on available knowledge about the underlying resources and how they are represented. For example, consider the query Q_1 that essentially intended to find out genes that are implicated in both obesity and male infertility caused by teratozoospermia. Among the many possibilities, there are three solutions a potential workflow generator may consider.

(1) Use the MiKDB infertility database to retrieve genes by selecting these two phenotypes individually to collect the corresponding gene lists, and then taking an intersection of these lists to find the genes that are implicated for both diseases.
(2) We could use a process similar to MiKDB to retrieve the genes from DisGeNET database.
(3) We could also retrieve the teratozoospermia implicated genes from MiKDB and use the retrieved gene lists to find their disease gene association with obesity in DisGeNET to retain the ones that are implicated for both.

The suggestion of one these workflows will depend on the intelligence and other relevant operational considerations such as resource utilization, efficiency, availability of information related to computability, etc. Alternately, all these workflows could be suggested as possible alternatives to allow a query processor and optimizer to choose from to execute to maximize a processing goal.

3.3 Computability and Pipeline Breakdown

The reality is that the internet information overall are not complete, curated, sanitized, or standardized and this is also true for most of the online biological databases and tools, even when they could

found easily. The incomplete, unstructured and brittle nature of resource descriptions make it difficult to determine what can be accessed from an online database or how. For example, the widely used DisGeNET database is a repository that contains known disease association for all human genes. This database allows access to its contents using various APIs, and search interfaces. Fig 1 shows the Python code segment generated by ChatGPT 3.5 to access DisGeNET using its gene-disease association API.

```python
import requests

def get_gene_disease_associations(gene_symbol):
    base_url = "http://www.disgenet.org/api/"
    endpoint = "gene/{}/diseaseAssociation".format(gene_symbol)

    url = base_url + endpoint
    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()
        return data
    else:
        print("Error: Unable to retrieve data.")
        return None

# Example: Get gene-disease associations for the gene A1CF
gene_symbol = "A1CF"
associations = get_gene_disease_associations(gene_symbol)

if associations:
    print("Gene-Disease Associations for {}: ".format(gene_symbol))
    for association in associations:
        print("Disease: {}, Score: {}".format(association['disease_name
else:
    print("No data retrieved.")
```

Figure 1: Python code generated by ChatGPT to access DisGeNET database.

The code segment generated by the GPT engine has multiple fatal flaws. Assuming that we have a Google Colab back-end that will be able to execute this code segment, the presented code cannot be directly executed. Because the script while conceptually correct, failed to address the necessary authentication process, rendering it ineffective for accessing the protected resources of the DisGeNET API. In other words, the GPT model did not recognize the necessity of acquiring an API key for access – a process that involves registering an account with DisGeNET and subsequently authenticating via a POST request to `"https://www.disgenet.org/api"` with credentials. This procedure yields a token that serves as the requisite API key. Furthermore, the AI-generated script contained inaccuracies in the endpoint specification, erroneously suggesting `"gene/{}/diseaseAssociation". format(gene_symbol)` instead of the correct expression `"gda/gene/{}".format(gene_symbol)`. Additionally, it overlooked the inclusion of authentication headers in the request, essential for accessing the endpoint. The correct implementation thus necessitates modifying the request to include headers as: `headers = {"Authorization": "Bearer " + bearer_token}`, where the `bearer_token` corresponds to the acquired API key. Such serious oversights underscores the need for computationally savvy human expert involvement which we hoped to avoid in the first place.

Code breakage or non-availability of codes are not the only reason why a workflow may fail. A workflow may fail because schema incompatibility, semantic errors, or semantic drift[1] of the data sources. Information obsolescence, software inadequacy or algorithmic deficiency, and inaccurate or incomplete resource annotations usually are the underlying causes of these failures. It is easy to see that these types of failures are technically more challenging to detect and fix autonomously.

3.4 Crowd based Failure Remediation

The diagnosis and troubleshooting difficulty posed by workflow failures can be addressed by manual inspection by qualified users. In this approach, the failures could be trapped, and qualified crowds could be requested to help isolate and fix the errors. Helping the crowd with smart diagnostic and code synthesis tools will go a long way to motivate them to volunteer their time. If the crowd could be selected from the group of users who would eventually benefit from the failure remediation, they could have a skin in the game. The recent focus on developing knowledge ecosystems [6, 19, 37] for better science [43] could be leveraged to develop a community knowledge ecosystem as part of a scientific inquiry processing system. A more detailed discussion in Sec 4.2.1 follows.

4 A MODEL FOR SCIENTIFIC INQUIRIES

The vision and the contours of ProAb laid out in Sec 3.1 is abstract and conceptual. In this section, we develop a data model for ProAb using which scientific inquiries, i.e., the queries of the form Q_1 and Q_2s, could be processed. The model we introduce is an extended relational model that retains all the features and languages of traditional relational model and extends it with specialized features. In other words, Codd's relational model is a special case of ProAb.

4.1 ProAb Data Model

The ProAb data model $\mathcal{P}$ is a tuple of the form $\langle R, U, O, W, \tau, \omega, \mu, \varphi, \Sigma \rangle$, where R is a set of Codd's relations, U is a set of URLs, O is the set of extended relational operators, W is a set of wrapper rules, τ is a transformation function acting as an additional relational operator, ω is a wrapper generation function, μ is a schema matching function, φ is a form filler, and finally, Σ is a FAIR function that finds, and determines the capabilities of online resources. In the sections to follow, we elaborate on some of these components of $\mathcal{P}$.

4.1.1 The Transform Operator τ. ProAb relational model includes all the relational algebra operators, including extended relational algebra operators. Additionally, it includes the transform operator[2] borrowed from the DQL language [22]. We, however, discuss its basic principle below to clarify its functionality.

The corresponding declarative query language incarnation of the transform operator is the extract statement below. In statement,

[1] A semantic drift occurs when either the structure, content or the meaning are no longer aligned with the sources' previous view due to modifications.
[2] We refer the readers to [22] for a formal definition of the transform operator τ.

the deep web database (or a tool) at a URL φ is treated as a black box, to which an input relation r_i is sent and in return an output relation r_o is expected over the schemes S_i and S_o. In the extract statement in Fig 2, the output scheme S_o of r_o is $A_1, A_2, \ldots, A_n$ and the input scheme S_i is the scheme of r.

extract $A_1, A_2, \ldots, A_n$
using wrapper W, matcher M, filler F
at φ
submit r

Figure 2: The extract **statement for deep web database access.**

The most enabling feature of the extract statement perhaps is its using clause which specifies the set of access tools to be used that hide the details explicitly specified in the Python code segment in Fig 1. The wrapper W essentially captures all necessary access details encoded in a wrapper rule, including converting the returned data by database d at φ into a tabular form. However, the schema correspondence necessary between scheme $S_i = R = \{B_1, B_2, \ldots, B_m\}$ and with $S_o = \{A_1, A_2, \ldots, A_n\}$ is established by the mapper M. Finally, the form filler F helps construction of the endpoints needed to process each element in the input set.

Deep Web Database Abstraction. In DQL, and thus in ProAb, we view deep web databases or analysis tools as abstract relations, which is generated (or exposed or surfaced) when the transform operator τ is applied (equivalently, the extract statement is executed on the deep web). To illustrate the idea, consider the DisGeNET database from which we would like to extract gene-disease association for gene symbols A1CF, LEP, MC4R and ARMC2. Among these genes, only MC4R is known to be linked with obesity and not teratozoospermia while LEP is know to be linked with both, and the others are linked only to teratozoospermia. Fig 3 shows the process how the transform operator views the DisGeNET deep web database as a function to find the gene-disease association of a set of genes.

The main function of the DisGeNET search interface in Fig 3(a) is to accept a gene or a set of genes (separated by a double colon ::) and return for each gene a set of gene-disease associations along with other related information as a table as shown in Fig 3(c). However, it does so by taking the users through an intermediate step as shown in Fig 3(b) where the user could choose from a set of four association types. Regardless, the table displayed as the end result (i.e., Fig 3(c)), has a scheme and the rows spanning multiple pages. These individual pages are of users' interest, and they can be scraped and converted into Codd's relations, and a wrapper could be built to do so. One way to abstract this entire functionality of submitting a set of genes to to retrieve the set of gene-disease association pairs can be conceptualized as shown in Fig 4, more specifically as shown in the modified relational operator in Fig 4(b).

In a traditional relational operation β, a relation r is transformed into r'. In contrast, in the above example, the operation β (the DisGeNET database) in Fig 4(b) accepts a relation r (the set of gene symbols) and a function α (a wrapper function), called a modifier of the behavior of β, returns a table r' (the gene-disease association table). The interesting details missing is that how the two functions, β and α, are conceived and behave. The function β can be considered as a black-box and need not concern us how it transforms the table r and into what kind of table except that the table returned is some function of a set of table including r, and is meaningful.

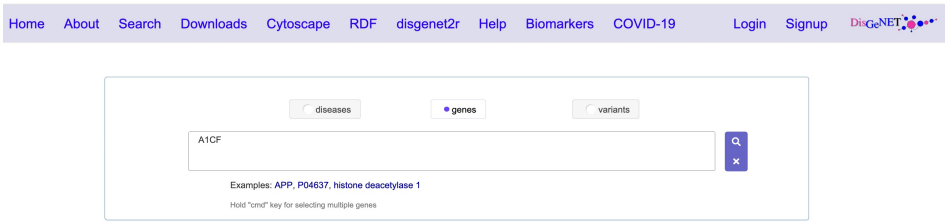
Basis for the Transform Operator. To accommodate deep web databases as online resources in LOD network, only a few modest adjustments to the traditional query model in which queries are considered as functions that transform a set of objects into another set of objects is required. In relational model, queries expressed in a higher level language such as SQL, are mapped to and processed as relational algebra expressions involving algebraic operators. Each of these operators are modeled as basic functions as shown in figure 4(a) in which a relation r is transformed into r' by a function β. All relational operators are closed because they transform relations to produce relations, making it convenient to build arbitrarily complex and nested queries involving operators and expressions.

Such a query model continues to serve us well in numerous applications. However, the functionalities discussed in Sec 3.1 includes two features that disrupt this view in particular. The first is the need to view an online database, or computational tool, as a local database or function. Secondly, we now need to recognize each external database access and tool application as an explicit step in a series of steps organized into an algorithm like process structure. The current approach to the former is to either process each online resource as an off line application that collects the results into a table for onward processing in a local application or a database query, or write glue codes to send needed information to the online resource and receive its response as part of a database query sequence. Such an approach is inherently manual, case specific, expensive, and sluggish in addition to the time needed to develop such applications through programming. Perhaps the most salient downside of this approach is its detrimental effect on spontaneity and ad hoc nature of querying, because it acts as serious economic deterrent to investigative pursuit and exploration[3].

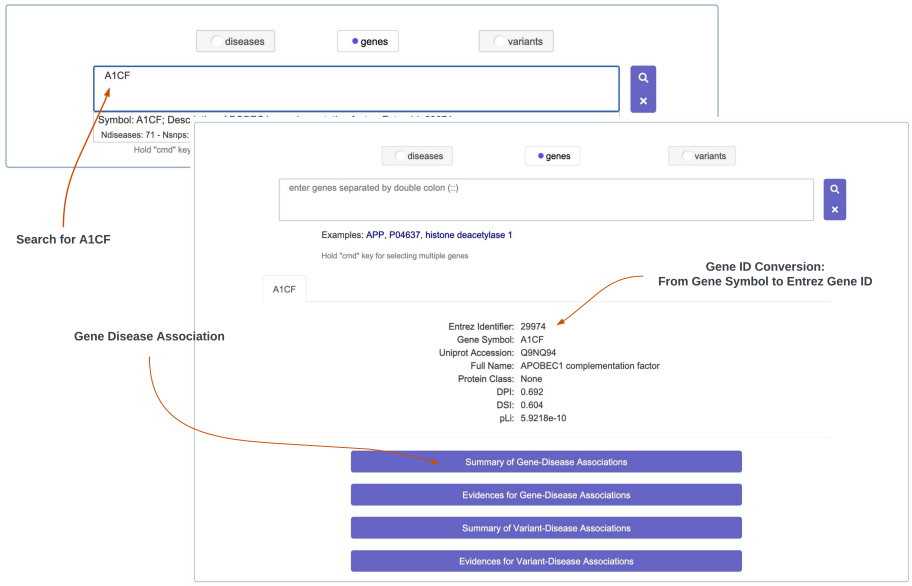
The latter issue of computational pipeline support for life sciences applications also has been advocated mainly as workflow systems that includes local or online databases as resources, and rarely as language support within database machineries. Successful systems such as Galaxy and Taverna are among the several prominent workflow systems in existence. Despite their strengths and usefulness, these systems do not empower database users with workflow primitives to explore ad hoc analysis using arbitrary resources. They certainly do not empower users to fashion scientific inquiries on the internet as discussed in Sec 3.1. Although there have been many attempts at developing generic workflow languages such as WOOL [20] and BPEL4WS [46] based languages [44], they are not nearly as relevant for the kind of biological applications we foresee.

4.1.2 Removing the Impedance Mismatch. We take the position that conventional relational model and the query language SQL already have necessary machineries to support applications involving local databases that do not require computational pipeline support other than a sequence of queries of steps. In order to include online resources, databases or application tools, we believe that casting them

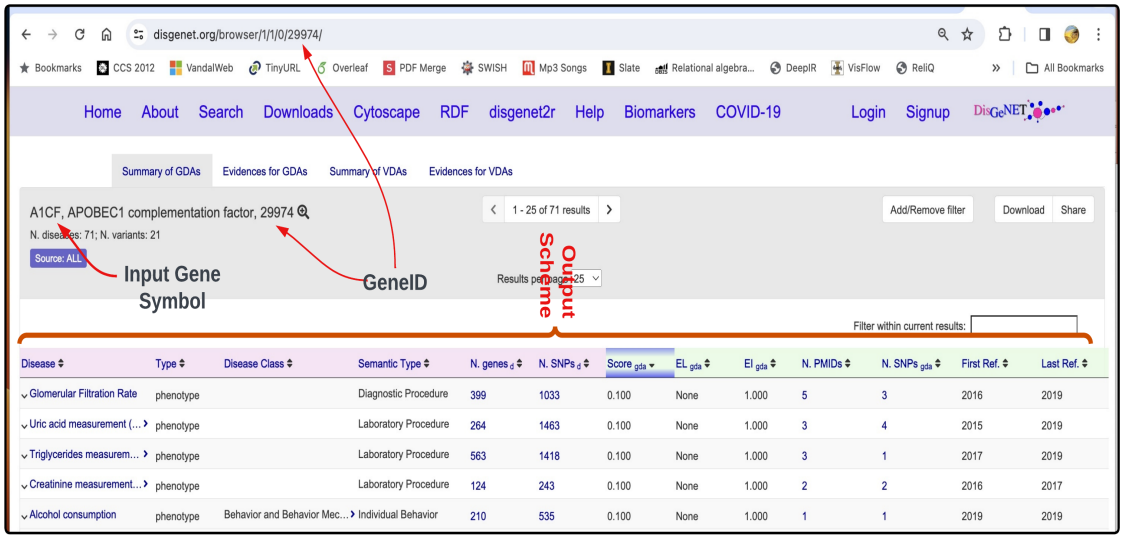
[3]Developing standard BioPerl scripts to access information from popular public databases and making them available has been one way researchers explored to help elevate the bottleneck, but at the end it does not address the basic issue of ad hoc access and integration of online databases.



(a) Search interface for gene-disease association discovery.

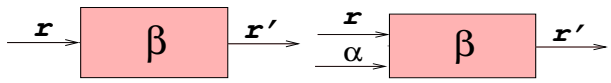


(b) Association type selection interface.



(c) Gene-disease association table.

Figure 3: DisGeNET browser access to gene-disease association information as a table.

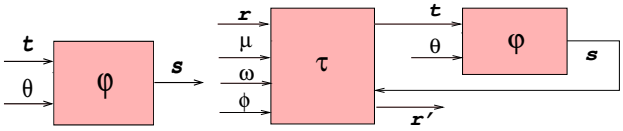


(a) Relational operator model. (b) New relational operator model.

Figure 4: Basis for a new relational model for data integration and computational pipeline queries.

as functions and blending them appropriately into the algebraic machinery will preserve the conceptual integrity and continuity of the model without any disruption. By that we mean that we would like to stay within the relational model, yet be able to support the inclusion of external databases as just another set of tables that are queryable using conventional relational operations.

However, in most cases, the only way to access life sciences databases is through web interfaces that upon submission of input values, return a set of responses in conventional HTML format. Apart from the technological disparity, this dichotomy in representing data objects, relations versus semi-structured documents, creates an impedance mismatch [5, 21] that is quite difficult to address mathematically from a data model standpoint[4]. There is also no simple mechanism in relational model to view online databases as a relation in any application. Currently, the only option is to call a procedure to submit required input to a form, extract the response as a table, and use it. To remove this hurdle, we propose to view each web form as a function that returns a table when provided with a set of parameters as shown in figure 5(a). In this view, the form function φ given a tuple and a condition θ, generates a set of objects s in a form other than a table. We can iterate over the set of tuples in a relation r and union the responses s for each tuple into S, and then extract the set of tuples using a wrapper ω as a set of tuples r' as a final response as shown in figure 5(b). This innocuous view of web forms essentially changes the whole picture of data integration as we explain in the next sections.



(a) Functional abstraction of online data resources. (b) Transform operator.

Figure 5: Abstract view of web resources as relations.

The issue now is how we blend in the table a web form function generates into a relational database and use the relation in an algebraic expression. The answer lies in blending together a recent but less explored research on *Data Mappers* introduced by Carreira [4] in which he showed that relational operators can be meaningfully parameterized to accept functions to alter relational aggregate operator behavior to map the input data differently for different queries by maintaining first-order semantics. The general idea of his approach to operator parameterization is schematically shown in figure 4(b) for unary operators with only one function parameter. We generalize Carreira's approach to operator parameterizations in our proposed model and extend to all relational operators to support ad hoc data integration over heterogeneous schemes.

[4]Adopting a completely semi-structured data model though remains as a choice for eliminating this impedance mismatch. Though there are many XML based standards in life sciences and many databases still have textual data representations, relational databases are actually the platform of choice for almost all.

4.2 Semantics of the extract Statement

The extract statement in Fig 2 actually casts the abstract view of the transform operator and makes it executable as part of a relational engine, such as ProAb. The wrapper, matcher and the filler functions cooperate with each other to help submit the rows in the table r by appropriately constructing the endpoints and normalizing the schemes of r the deep web database at φ and the scheme listed in the extract statement. In this section, we discuss the features we need to have in these three functions in particular.

4.2.1 Wrapper CroW. The example in Fig 3 illustrates the complexity and difficulty of generating a wrapper for deep web databases such as DisGeNET. The traditional approaches to wrapper generation is largely for just scrapping from a single web page [42, 48] regardless of their application, complexity or sophistication. Even when they are for scientific applications [29] developed by experts, or are developed by crowd [7] for generic applications. In contrast, we do aim to scrape the contents, but the wrappers in ProAb do more than just scrapping.

We require that wrappers encapsulate the entire data extraction or retrieval process into a single function, i.e., sending the list of values the deep web database needs to be submitted in a form to return the table that we need to scrape. We therefore need to understand how to create the HTTP endpoint to retrieve the table. To this end, we are developing a smart wrapper generation system, called CroW, for user initiated wrapper induction. In this system, users are able to bring in the sites such as DisGeNET, and follow the three steps as shown in Fig 3, mark the tables, pagination, and other related details in a simple interactive interface, and push a button to create the wrapper. The wrapper is then saved as an extraction rule and can be called in the extract statement to retrieve the data.

4.2.2 Levels of Abstraction. Unlike most wrapper generation systems, CroW supports three types of wrapper generation to access data from online resources as discussed below.

Page Level Wrapper - Level Zero. Though more elaborate and more powerful, CroW is able to access multi-page tables through an endpoint URL. It is able to detect patterns in multiple pages to find the schemes and the rows, isolate a table defined using HTML table structures, or extract a target table from a set of tables in a page. A brief description of CroW's functionality can be found in Sec 4.2.3, and a more detailed discussion in [31].

Form Level Wrapper - Level One. CroW also supports deep web access through web form submissions. For example, the MiKDB database supports a form based search engine in addition to browser level data access in different categories that can be scraped or downloaded as a level zero wrapper. For example, downloading 17,754 genes at http://mik.bicnirrh.res.in/browse_genes.php.

However, developing a wrapper for a form based access is more complicated. The code segment in Fig 6 employs Selenium and BeautifulSoup to simulate the submission of the keyword "teratozoospermia" into the MiKDB search form at http://mik.bicnirrh.res.in/searchbox2.html and retrieve the gene information for this phenotype.

As discussed in Sec 4.2.3, the form filler function Seed must generate a new wrapper for each disease phenotype by replacing the keyword "teratozoospermia" in the code segment search_field.send_keys ("teratozoospermia") if and when needed. The wrapper could be named and saved for use in the extract statement and parameterized with the help of the form filler.

API Level Wrapper - Level Two. Finally, CroW also supports API level wrappers as shown in Fig 7. In this wrapper too, the gene list in the code segment gene_symbol = "A1CF" will have to be replaced by genes from the table *genes* in the extract statement by the form filler one by one, or by a gene list. As before, the wrapper could be saved and used in the extract statement alongside the form filler Seed as appropriate.

4.2.3 Form Filler Seed. While generating a fixed wrapper is simpler, parameterizing a wrapper for a vector of changing values require a dynamic construction of a wrapper from a base definition. Consider the query Q_1 discussed in Sec 4.1.1 to retrieve the gene-disease associations for the genes A1CF, LEP, MC4R and ARMC2. These genes could be supplied as the table *genes* as follows

GeneSymbol
A1CF
LEP
MC4R
ARMC2

in the extract statement below.

extract Gene, Phenotype
using wrapper *DisWrap*, matcher *Cupid*, filler *Seed*
at https://www.disgenet.org/
submit *genes*

The actual endpoint needed to extract the gene-disease association for the gene symbol A1CF is https://www.disgenet.org/browser/1/1/0/29974/. In this endpoint expression, 29974 is the Entrez gene ID for the gene symbol A1CF. The form filler function Seed must know that wrapper DisWrap generated using CroW needs to wrap the table at https://www.disgenet.org/browser/1/1/0/29974/ for A1CF, and at https://www.disgenet.org/browser/1/1/0/3952/ for the gene LEP, and so on. However, if the wrapper was for an API based access as opposed the browser based access, Seed must adjust the access strategy accordingly. While the wrappers are designed for individual sites or databases, the form filler is generic and it is an algorithm that uses descriptions generated by the wrapper to function.

4.2.4 Schema Matcher. As shown in Fig 3(c), the scheme of the returned DisGeNET table consists of many attributes, and the most interesting column is the called *Disease*. However, the query scheme in the extract statement is S_i={*GeneSymbol*}, and S_o={*Gene, Phenotype*}, which do not match with the query and table schemes. It is the *Cupid* schema matcher's job to resolve the heterogeneity to help identify the right columns and to return the table below as the extracted table. In this case, Cupid maps *Genes* to *GeneSymbol*, and *Phenotype* to *Disease*. For now, we are using Cupid [27] and S-Match [15] as two generic schema matchers for ProAb, which serve our purpose in general in most part. A more application and biology specific generic schema matching system could potentially improve the matching accuracy, and remains in our future plans.

Gene	Phenotype
A1CF	Gout
A1CF	Schizophrenia
A1CF	Obesity
LEP	Obesity
LEP	Hyperinsulinism
...	...

5 PROAB QUERY PROCESSOR

The ProAb query processor is an integration of several functional components and implemented by stitching them in a logical workflow as detailed in Alg 1. ProAb uses a resource capability discovery system, called FAIRyfier [32], for online resources or deep web databases that help find the databases using a vector database generated from PubMed abstracts. Using information from FAIRYfier, CroW is able to generate a wrapper for the resource. These wrapper information are then used to develop resource and process descriptions in Needle [24]. A smart analyzer is then used to generate a consistent workflow script in Needle that can be executed. A Needle script is a sequence of SQL and extract statements that ProAb is able to execute.

Algorithm 1: ProAb Query Processor

Input: A text query Q
Output: A table T as the answer to query Q
S = Submit Q to ChatGPT;
for *each database D in S* **do**
  Determine input and output schemes using FAIRyfier;
  Develop wrapper using CroW and add to wrapper bank;
Map each step in S to a Needle expression;
Construct intermediate workflow sequence from the Needle expression;
Develop an admissible workflow in Needle;
Execute;
return table T;

5.1 Resource Description in ProAb using Needle

In ProAb, the machine readable version of the resource description R_d for the DisGeNET database is described as the pair $R_d = \langle p_d, r_d \rangle$, where p_d is the lower level detail of actual process used in DisGeNET, and r_d is encapsulates p_d with additional maintenance related information ProAb uses. The Needle statement below helps understand the functionalities of the browser-based access of gene-disease association (GDA) in DisGeNET and captures the three necessary components, i.e., the process identifier, the resource address and its features, and the input and output table schemes as shown in Fig 3(c).

```
def process_mikdb():
    url = "http://mik.bicnirrh.res.in/searchbox2.html"
    service = Service(executable_path=ChromeDriverManager().install())
    options = webdriver.ChromeOptions()
    options.add_argument("start-maximized")
    options.add_experimental_option("excludeSwitches", ["enable-automation"])
    options.add_experimental_option('excludeSwitches', ['enable-logging'])
    options.add_experimental_option('useAutomationExtension', False)
    options.add_argument('--disable-blink-features=AutomationControlled')
    options.add_argument("--headless")
  options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
     Chrome/87.0.4280.88 Safari/537.36")
    driver = webdriver.Chrome(service=service, options=options)
    driver.maximize_window()
    driver.get(url)
    search_field = driver.find_element(By.NAME, "search")
    search_field.send_keys("teratozoospermia")
    submit_button = driver.find_element(By.NAME, "submit")
    submit_button.click()
    WebDriverWait(driver, 10).until(EC.url_to_be("http://mik.bicnirrh.res.in/search_result4.php"))
    the_soup = BeautifulSoup(driver.page_source, 'html.parser')
    news_main_block = the_soup.find('div', id='left')
    table = news_main_block.find('table')
    rows = table.find_all('tr')
    second_td_values = []
    for row in rows[2:]:
        cells = row.find_all('td')
        if len(cells) >= 2:
            second_td_text = cells[1].get_text(strip=True)
            second_td_values.append(second_td_text)
    driver.close()
    second_td_values = list(set(second_td_values))
    return second_td_values
```

Figure 6: Level one wrapper for disease specific gene list retrieval from MiKDB.

```
% process identifier
create process DisGeNETb
% access protocol
    at https://www.disgenet.org/
    access browser
    postfix /browser/1/1/0/$Genes
% input table scheme
    accepts table (Genes EntrezID)
% output table scheme
    returns table (
    DisGeNETKey EntrezID primary key,
    Disease DisID,
    Type string,
    Disease_class string,
    Score_gda decimal (4,2), . . . );
```

From this statement, Seed form filler is able to reconstruct the URLs[5] of GDA descriptions for a given gene. The $ sign in the postfix clause indicates substitutions available in the accepts clause.

[5]Such as `https://www.disgenet.org/browser/1/1/0/29974/` for gene A1CF using its Entrez ID 29974.

A resource description ρ for DisGeNET is constructed from its process description to complete the knowledge about this internet resource in the following format. The primary function of the resource description is to help the workflow generation algorithm find, assemble and construct executable workflows as the implementation of an admissible workflow graph by selecting the most credible components from its knowledgebase.

```
% resource identifier
create resource DisGeNETb (
% resource narrative for machine consumption
    narrative "Browser access accepts an Entrez
       gene ID and returns its disease association",
% process contributors
    contributors {Alex, Abebi},
% applicable data integration tools
    meta:
      matcher {Cupid, OntoMatch},
      wrapper {FastWrap},
      mapping {Determination: Semantic},
```

```
import requests
def get_gene_disease_associations(gene_symbol):
    base_url = "http://www.disgenet.org/api/"
    endpoint = "gda/gene/{}".format(gene_symbol)
    api_key = 'af7e0409649a4a46caadf14c45a7ee7920f5fb16'
    headers = {"Authorization": "Bearer {}".format(api_key)}
    url = base_url + endpoint
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        data = response.json()
        return data
    else:
        print("Error: Unable to retrieve data.")
        return None
# Example: Get gene-disease associations for the gene A1CF
    gene_symbol = "A1CF"
    associations = get_gene_disease_associations(gene_symbol)
    if associations:
        print("Gene-Disease Associations for {}: ".format(gene_symbol))
    for association in associations:
        print("Disease: {}, Score: {}".format(association['disease_name'], association['score']))
    else:
        print("No data retrieved.")
```

Figure 7: Level two wrapper for gene specific gene-disease association list retrieval from DisGeNET.

% process testers and validators
validators *{Alex, Maya}*);

The resource description above is linked with the process description *DisGeNETb* and captures other essential information required for the construction of a credible workflow. In particular, it helps mapping by including the list of effective schema matchers, and wrappers. It also lists term mapping exceptions under *mappings* clause that must be used and thus overrides any mapping decision by any schema mapping algorithm, e.g., the pair *Determination: Semantic* under mapping is one such mapping. One of the meta entries also lists the users who validated the accuracy of this resource description.

5.2 Mapping to Needle Workflow

Two of the major components of ProAb are the workflow graph mapping function Π and graph to program mapping function Ω. The workflow mapper Π generates the graph in Fig 8(a) from the steps generated by the query Q to query plan mapping function τ using ChatGPT. At this stage, the plan is just basic and raw, describing the tentative workflow. The program mapper Ω then uses the resource descriptions of MiKDB and DisGeNET below and develops a concrete workflow plan by inserting the MapBase data conversion step in between MiKDB and DisGeNET as shown in Fig 8(b). At this point ProAb confirms that this workflow is executable only if gene symbols returned by MiKDB is converted to Entrez gene IDs as DisGeNET can only process gene IDs, as described in its resource description. The program mapper Ω then converts the workflow in Fig 8(b) into the Needle program below that ProAb is able to execute.

select *Type, N_genes, Score_gda, EL_gda, N_PMIDs, First_Ref*
from (with *mapbase* as
extract[1] *GeneID*
using matcher *Cupid* wrapper *CroW*
from *https://www.mapbase.org*
submit (extract[2] *Symbol*
using matcher *Cupid* wrapper *CroW*
from *http://mik.bicnirrh.res.in/mip.php*
submit *("teratozoospermia")*)
extract[3] *Disease, Type, N_genes, Score_gda, EL_gda, N_PMIDs, First_Ref*
using matcher *Cupid* wrapper *CroW*
from *https://www.disgenet.org/browser/1/1/0/*
submit mapbase)
where *Disease = 'obesity'* and *Score_gda > 0.01*

We note that the Needle query above is executable in ProAb. In the above query, the MiKDB view expression in the second extract statement (extract[2]) was generated from its resource description below.

create process *MiKDB*
at *http://mik.bicnirrh.res.in*
access webform
postfix /mip.php/
accepts filter (*Phenotype* String)
returns table (
Symbol GeneSymbol primary key,
ChrLoc string,
Disease string);

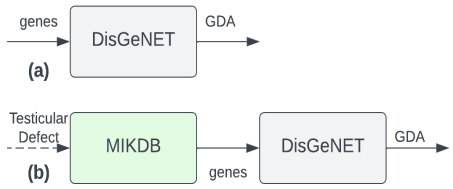


Figure 8: (a) DisGeNET as a resource node and (B) workflow suggested by Ω.

Figure 9: Workflow graph generated by Π.

6 CONCLUSION

The goal in this paper was to demonstrate that a smart LLM based ad hoc scientific inquiry processing system over LOD is possible. In particular, it was not our goal to claim that the ProAb is a completely functional engine that currently works. Rather, we have introduced the components we need and how they could be assembled to realize ProAb. The most exciting aspect of this on going research is the concept of the transform operator and how this extended operator helps abstract the access mechanism of web resources. We believe we have demonstrated that ProAb is a promising approach to query scientific databases by extending existing technologies in synergistic ways. We hope that more research to strengthen ProAb and several of its component technologies will soon materialize and make it possible to witness a fully functional system such as ProAb for scientific inquiries.

REFERENCES

[1] Mohamed Abouelhoda, Shadi Issa, and Moustafa Ghanem. 2012. Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. *BMC Bioinform.* 13 (2012), 77.

[2] Enis Afgan and The Team. 2022. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2022 update. *Nucleic Acids Res.* 50, W1 (2022), 345–351.

[3] Daniel Carlander-Reuterfelt, Álvaro Carrera, Carlos Angel Iglesias, Oscar Araque, J. Fernando Sánchez-Rada, and Sergio Muñoz. 2020. JAICOB: A Data Science Chatbot. *IEEE Access* 8 (2020), 180672–180680.

[4] Paulo Carreira, Helena Galhardas, Antónia Lopes, and João Pereira. 2007. One-to-many data transformations through data mappers. *Data Knowl. Eng.* 62, 3 (2007), 483–503.

[5] Jian Chen and Qiming Huang. 1995. Eliminating the Impedance Mismatch Between Relational Systems and Object-Oriented Programming Languages. In *6th International Hong Kong Database Workshop.*

[6] Anamika Chhabra, S. R. S. Iyengar, Poonam Saini, and Rajesh Shreedhar Bhat. 2015. Presence of an Ecosystem: a Catalyst in the Knowledge Building Process in Crowdsourced Annotation Environments. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2015, Paris, France, August 25 - 28, 2015*, Jian Pei, Fabrizio Silvestri, and Jie Tang (Eds.). ACM, 286–291.

[7] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. 2013. Wrapper Generation Supervised by a Noisy Crowd. In *Proceedings of the First VLDB Workshop on Databases and Crowdsourcing, DBCrowd 2013, Riva del Garda, Trento, Italy, August 26, 2013 (CEUR Workshop Proceedings, Vol. 1025)*, Reynold Cheng, Anish Das Sarma, Silviu Maniu, and Pierre Senellart (Eds.). CEUR-WS.org, 8–13.

[8] Sugat Dabholkar, Gabriella Anton, and Uri Wilensky. 2018. GenEvo - An emergent systems microworld for model-based scientific inquiry in the context of genetics and evolution. In *Rethinking learning in the digital age: Making the Learning Sciences count - Proceedings of the 13th International Conference of the Learning Sciences, ICLS 2018, London, UK, June 23-27, 2018*, Manolis Mavrikis and Kaska Porayska-Pomsta (Eds.). International Society of the Learning Sciences.

[9] George Datseris, Jonas Isensee, Sebastian Pech, and Tamás Gál. 2020. DrWatson: the perfect sidekick for your scientific inquiries. *J. Open Source Softw.* 5, 54 (2020), 2673.

[10] Jan de Wit and Anouck Braggaar. 2023. Tilbot: A Visual Design Platform to Facilitate Open Science Research into Conversational User Interfaces. In *Proceedings of the 5th International Conference on Conversational User Interfaces, CUI 2023, Eindhoven, The Netherlands, July 19-21, 2023*, Minha Lee, Cosmin Munteanu, Martin Porcheron, Johanne Trippas, and Sarah Theres Völkel (Eds.). ACM, 55:1–55:5.

[11] Ranjeet Devarakonda, Yaxing Wei, and Michele Thornton. 2016. Accessing and distributing large volumes of NetCDF data. In *2016 IEEE International Conference on Big Data (IEEE BigData 2016), Washington DC, USA, December 5-8, 2016*, James Joshi, George Karypis, Ling Liu, Xiaohua Hu, Ronay Ak, Yinglong Xia, Weijia Xu, Aki-Hiro Sato, Sudarsan Rachuri, Lyle H. Ungar, Philip S. Yu, Rama Govindaraju, and Toyotaro Suzumura (Eds.). IEEE Computer Society, 3966–3967.

[12] Martin Drechsler. 2020. Model-based integration of ecology and socio-economics for the management of biodiversity and ecosystem services: State of the art, diversity and current trends. *Environ. Model. Softw.* 134 (2020), 104892.

[13] Nuno Fachada. 2022. A Computational Pipeline for Modeling and Predicting Wildfire Behavior. In *Proceedings of the 7th International Conference on Complexity, Future Information Systems and Risk, COMPLEXIS 2022, Online Streaming, April 23-24, 2022*, Min Xie, Reinhold Behringer, and Victor Chang (Eds.). SCITEPRESS, 79–84.

[14] Matheus Pedra Puime Feijoó, Rodrigo Jardim, Sergio Manuel Serra da Cruz, and Maria Luiza Machado Campos. 2022. GAP: Enhancing Semantic Interoperability of Genomic Datasets and Provenance Through Nanopublications. In *Metadata and Semantic Research*, Emmanouel Garoufallou, María-Antonia Ovalle-Perandones, and Andreas Vlachidis (Eds.). Springer International Publishing, Cham, 336–348.

[15] Fausto Giunchiglia, Aliaksandr Autayeu, and Juan Pane. 2012. S-Match: An open source framework for matching lightweight ontologies. *Semantic Web* 3, 3 (2012), 307–317.

[16] David Gomez-Cabrero, Imad Abugessaisa, Dieter Maier, Andrew E. Teschendorff, Matthias Merkenschlager, Andreas Gisel, Esteban Ballestar, Erik Bongcam-Rudloff, Ana Conesa, and Jesper Tegnér. 2014. Data integration in the era of omics: current and future challenges. *BMC Syst. Biol.* 8, S-2 (2014), I1.

[17] Stephen P. Harter. 1984. Scientific inquiry: A model for online searching. *J. Am. Soc. Inf. Sci.* 35, 2 (1984), 110–117.

[18] Hossein Hassani and Emmanuel Sirimal Silva. 2023. The Role of ChatGPT in Data Science: How AI-Assisted Conversational Interfaces Are Revolutionizing the Field. *Big Data Cogn. Comput.* 7, 2 (2023), 62.

[19] Janna Hastings. 2021. Scientific Ontologies, Digital Curation and the Learning Knowledge Ecosystem. In *Proceedings of the Conference on Digital Curation Technologies (Qurator 2021), Berlin, Germany, February 8th - to - 12th, 2021 (CEUR Workshop Proceedings, Vol. 2836)*, Adrian Paschke, Georg Rehm, Jamal Al Qundus, Clemens Neudecker, and Lydia Pintscher (Eds.). CEUR-WS.org.

[20] Geoffrey C. Hulette, Matthew J. Sottile, and Allen D. Malony. 2008. WOOL: A Workflow Programming Language. In *eScience*. 71–78.

[21] Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh. 2009. A Classification of Object-Relational Impedance Mismatch. In *DBKDA*. 36–43.

[22] Hasan M. Jamil and H. V. Jagadish. 2015. A Structured Query Model for the Deep Relational Web. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, James Bailey, Alistair Moffat, Charu C. Aggarwal, Maarten de Rijke, Ravi Kumar, Vanessa Murdock, Timos K. Sellis, and Jeffrey Xu Yu (Eds.). ACM, 1679–1682.

[23] Hasan M. Jamil, Stephen A Krawetz, and Alexander Gow. 2023. Leveraging Large Language Models for Automatic Hypotheses Testing over Heterogeneous Biological Databases. In *Proceedings of the 14th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, BCB 2023, Houston, TX, USA, September 3-6, 2023*, May D. Wang and Byung-Jun Yoon (Eds.). ACM, 80:1–80:2.

[24] Hasan M. Jamil, Stephen A Krawetz, and Alexander Gow. 2024. Knowledge Synthesis using Large Language Models for a Computational Biology Workflow Ecosystem. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, SAC 2024, Avila, Spain, April 8–12, 2024*. ACM.

[25] Hasan M. Jamil and Kallol Naha. 2023. Mapping Strategies for Declarative Queries over Online Heterogeneous Biological Databases for Intelligent Responses. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, SAC 2023, Tallinn, Estonia, March 27-31, 2023*, Jiman Hong, Maart Lanperne, Juw Won Park, Tomás Cerný, and Hossain Shahriar (Eds.). ACM, 567–574.

[26] Kai Jing, Yewen Xu, Yang Yang, Pengfei Yin, Duo Ning, Guangyu Huang, Yuqing Deng, Gengzhan Chen, Guoliang Li, Simon Zhongyuan Tian, and Meizhen Zheng.

2023. ScSmOP: a universal computational pipeline for single-cell single-molecule multiomics data analysis. *Briefings Bioinform.* 24, 6 (2023).

[27] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. 2001. Generic Schema Matching with Cupid. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass (Eds.). Morgan Kaufmann, 49–58.

[28] Benjamin Maraza-Quispe, Nicolas Esleyder Caytuiro-Silva, Erick Abel Arizaca Machaca, Jorge Luis Torres-Loayza, Gustavo Alonso Ccama Marron, and Walter Cornelio Fernandez Gambarini. 2023. Promoting Scientific Inquiry in Physics Students through the Use of Simulators. In *The 15th International Conference on Education Technology and Computers, ICETC 2023, Barcelona, Spain, September 26-28, 2023*. ACM, 101–106.

[29] Saqib Mir, Steffen Staab, and Isabel Rojas. 2009. Web-Prospector - An Automatic, Site-Wide Wrapper Induction Approach for Scientific Deep-Web Databases. In *Datenbanksysteme in Business, Technologie und Web (BTW 2009), 13. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Proceedings, 2.-6. März 2009, Münster, Germany (LNI, Vol. P-144)*, Johann Christoph Freytag, Thomas Ruf, Wolfgang Lehner, and Gottfried Vossen (Eds.). GI, 87–106.

[30] Milad Mirbabaie, Jonas Rieskamp, Lennart Hofeditz, and Stefan Stieglitz. 2024. Breaking Down Barriers: How Conversational Agents Facilitate Open Science and Data Sharing. In *57th Hawaii International Conference on System Sciences, HICSS 2024, Hilton Hawaiian Village Waikiki Beach Resort, Hawaii, USA, January 3-6, 2024*, Tung X. Bui (Ed.). ScholarSpace, 672–681.

[31] Kallol Naha and Hasan M Jamil. 2024. *Crowd Inspired Wrapper Generation as a Community Resource.* Technical Report. Department of Computer Science, University of Idaho. Under review with CIKM 2024.

[32] Kallol Naha and Hasan M Jamil. 2024. *Ensuring Findability, Accessibility and Interoperability of unFAIR Biological Databases.* Technical Report. Department of Computer Science, University of Idaho. Under review with SIGIR 2024.

[33] Fatemeh Nargesian, Abolfazl Asudeh, and H. V. Jagadish. 2022. Responsible Data Integration: Next-generation Challenges. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 2458–2464.

[34] Nurfaradilla Mohamad Nasri, Nurfarahin Nasri, Nur Faraliyana Nasri, and Mohamad Asyraf Abd Talib. 2023. The Impact of Integrating an Intelligent Personal Assistant (IPA) on Secondary School Physics Students' Scientific Inquiry Skills. *IEEE Trans. Learn. Technol.* 16, 2 (2023), 232–242.

[35] Sara Pidò, Pietro Pinoli, Pietro Crovari, Francesca Ieva, Franca Garzotto, and Stefano Ceri. 2023. Ask Your Data - Supporting Data Science Processes by Combining AutoML and Conversational Interfaces. *IEEE Access* 11 (2023), 45972–45988.

[36] Jaakko Rajala, Jenni Hukkanen, Maria Hartikainen, and Pia Niemelä. 2023. "\"Call me Kiran\" - ChatGPT as a Tutoring Chatbot in a Computer Science Course". In *Proceedings of the 26th International Academic Mindtrek Conference, Mindtrek 2023, Tampere, Finland, October 3-6, 2023*. ACM, 83–94.

[37] Toni Ruokolainen and Lea Kutvonen. 2009. Managing interoperability knowledge in open service ecosystems. In *Workshops Proceedings of the 12th IEEE International Enterprise Distributed Object Computing Conference, EDOCw 2009, 1-4 September 2009, Auckland, New Zealand*. IEEE Computer Society, 203–211.

[38] Abderrazak Sebaa and Abdelkamel Tari. 2019. Materialized View Maintenance: Issues, Classification, and Open Challenges. *Int. J. Cooperative Inf. Syst.* 28, 1 (2019), 1930001:1–1930001:59.

[39] Uthayanath Suthakar, Luca Magnoni, David Ryan Smith, Akram Khan, and Julia Andreeva. 2016. An efficient strategy for the collection and storage of large volumes of data for computation. *J. Big Data* 3 (2016), 21.

[40] Vanessa Tang, Alaeddin Nassani, Suranga Nanayakkara, and Mark Billinghurst. 2022. Kiwrious AR: Exploring AR for Scientific Inquiry and Scaffolded Learning. In *2022 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct), Singapore, Singapore, October 17-21, 2022*. IEEE, 911–912.

[41] Saeid Ashraf Vaghefi, Qian Wang, Veruska Muccione, Jingwei Ni, Mathias Kraus, Julia Anna Bingler, Tobias Schimanski, Chiara Colesanti Senni, Nicolas Webersinke, Christian Huggel, and Markus Leippold. 2023. chatClimate: Grounding Conversational AI in Climate Science. *CoRR* abs/2304.05510 (2023).

[42] Iraklis Varlamis, Nikos Tsirakis, Vassilis Poulopoulos, and Panayiotis Tsantilas. 2014. An automatic wrapper generation process for large scale crawling of news websites. In *18th Panhellenic Conference on Informatics, PCI '14, Athens, Greece, October 2-4, 2014*, Sokratis K. Katsikas, Michael Hatzopoulos, Theodoros Apostolopoulos, Dimosthenis Anagnostopoulos, Elias Carayiannis, Theodora A. Varvarigou, and Mara Nikolaidou (Eds.). ACM, 1:1–1:6.

[43] Binh Vu, Yanxin Wu, Haithem Afli, Paul Mc Kevitt, Paul Walsh, Felix Engel, Michael Fuchs, and Matthias L. Hemmje. 2019. A metagenomic content and knowledge management ecosystem platform. In *2019 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2019, San Diego, CA, USA, November 18-21, 2019*, Illhoi Yoo, Jinbo Bi, and Xiaohua Hu (Eds.). IEEE, 1–8.

[44] Dejun Wang, Linpeng Huang, and Qinglei Zhang. 2006. A Workflow-Oriented Scripting Language Based on BPEL4WS. In *APWeb Workshops*. 690–697.

[45] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3, 1 (2016), 1–9.

[46] Petia Wohed, Wil M. P. van der Aalst, Marlon Dumas, and Arthur H. M. ter Hofstede. 2003. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *ER*. 200–215.

[47] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan R. Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex R. Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, and Carole A. Goble. 2013. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Res.* 41, Webserver-Issue (2013), 557–561.

[48] Yingju Xia, Yuhang Yang, Shu Zhang, and Hao Yu. 2011. Automatic Wrapper Generation and Maintenance. In *Proceedings of the 25th Pacific Asia Conference on Language, Information and Computation, PACLIC 25, Singapore, December 16-18, 2011*, Helena Hong Gao and Minghui Dong (Eds.). Digital Enhancement of Cognitive Development, Waseda University, 90–99.

[49] Xianjun Yang, Stephen D. Wilson, and Linda R. Petzold. 2024. Quokka: An Open-source Large Language Model ChatBot for Material Science. *CoRR* abs/2401.01089 (2024).