# SYMPHONY: Towards Natural Language Query Answering over Multi-modal Data Lakes


Zui Chen♣
Tsinghua University
China
chenzui19@mails.tsinghua.edu.cn

Zihui Gu♣
Renmin University of China
China
guzh@ruc.edu.cn

Lei Cao
MIT CSAIL/University of Arizona
USA
lcao@csail.mit.edu

Ju Fan
Renmin University of China
China
fanj@ruc.edu.cn

Sam Madden
MIT CSAIL
USA
madden@csail.mit.edu

Nan Tang
QCRI, HBKU
Qatar
ntang@hbku.edu.qa



## ABSTRACT

Wouldn't it be great if we could query large, diverse data lakes of tables, text, and databases as easily as using Siri or Alexa? The problem is hard from two perspectives: integrating data lakes requires data normalization/transformation, schema matching, and entity resolution and is notoriously hard, with high human cost. Even if successful, such integration efforts typically do not support arbitrary SQL queries over the integrated data set.

In this paper, we propose SYMPHONY, a novel system that enables users to easily query complex, multi-modal data lakes *without* performing upfront integration. For ease of use, SYMPHONY adopts a natural language (NL) interface. To avoid integration, it employs a unified representation for multi-modal datasets, called cross-modality representation learning. When a user poses an NL query, SYMPHONY discovers which tables or textual data should be retrieved based on the learned cross-modal representations, decomposes a complicated NL query into NL sub-queries on-demand, evaluates each sub-query on one data source and combines the results from these sub-queries. A preliminary evaluation shows that the resulting system is able to effectively answer questions over tables and text extracted from Wikipedia.


## 1 INTRODUCTION

Modern organizations often need to manage a huge volume of data with different modalities, including relational databases, spreadsheets, documentation, messages (*e.g.*, email, Slack), log files, and domain-specific sources such as sensor data, web pages, or knowledge graphs. We call such collections of diverse types of data **multi-modal data lakes**.

Data lakes are mostly intermediate repositories for data [5]. Traditional wisdom for data lake management requires to go through a *civilization* process [3, 13] including extraction, transformation, integration, linking, and so on, as depicted in Fig. 1(a). However, the data civilization problem is notoriously hard and labor intensive.

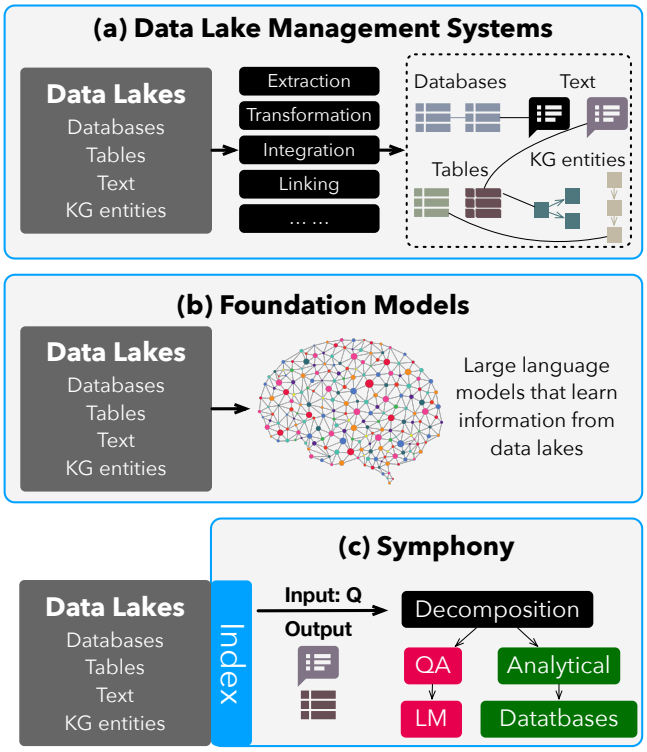


**Figure 1: The Vision of SYMPHONY.**

An oft-cited statistic is that data scientists spend 80% of their time on it. Not many enterprises can afford such costs.

Recently, giant language models (or foundation models) have been used to learn from data lakes and encode everything they learnt in their model parameters (see Fig. 1(b)), *e.g.*, GPT-3 [1]. Although they have shown exciting results for many natural language tasks and image generation, they are inefficient to reason about complicated (*e.g.*, analytical) queries over data lakes.

**The SYMPHONY Vision.** We propose SYMPHONY, a novel system that enables non-expert users to query multi-modal data lakes.

♣ This work was done when Zui visited MIT CSAIL. Zui and Zihui contribute equally to this work.



13th Annual Conference on Innovative Data Systems Research (CIDR '23). January 8-11, 2023, Amsterdam, The Netherlands.

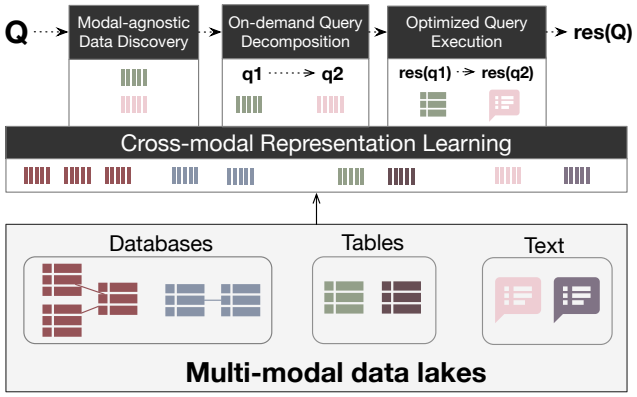


**Figure 2: An Overview of Symphony.**

Symphony simply indexes data lakes without any civilization operations, as shown in Fig. 1(c). At query time, Symphony will discover datasets-on-demand relevant to a given $Q$, and then reason about how to evaluate the query $Q$ over the discovered datasets using *e.g.*, language models for natural language question answering (QA) or database engines for analytical queries over tables or databases.

More specifically, Symphony allows the users to pose natural language (NL) queries, such as query **Q** in Fig. 3. Unlike SQL queries, users do not have to specify the datasets on which the NL queries run. Instead, Symphony will *discover* the datasets – which could be in different files or even modalities – and then decompose the NL query into NL sub-queries (only when needed) that can be effectively executed on individual datasets.

Symphony consists of four key components (see Fig. 2): cross-modal representation learning, modal-agnostic data discovery, on-demand query decomposition, and optimized query execution. We use **item** to refer to one table or a textual document. We illustrate each component with a running example shown in Fig. 3

- ***Cross-modal Representations Learning.*** Symphony learns a cross-modal representation model that encodes the items in the data lake, whatever their modality might be, into the same high-dimensional embedding space. The key idea is to first convert the items in different modalities into sequences of words and pre-train a Transformer-based language model (LM) to encode them. Rather than use the existing pre-training methods to train the model, we propose a new pre-training method customized to data lakes. It ensures that the embedding with respect to each item well preserves its vital features. Symphony then uses the learned representations in data discovery and query execution. (Section 3)
- ***Modal-agnostic Data Discovery.*** Given an NL query, Symphony discovers the datasets that should be used to answer this query. Unlike the traditional data discovery tools that typically search over the table title, column names, or metadata through keyword search or pattern matching (*e.g.*, regex), Symphony directly takes the content of the datasets into consideration, thus more effective. It achieves so by first using the learned cross-modal representation model to transform each upcoming NL query into an embedding vector, and then discovering the relevant data items based on their similarity to the query embedding. Moreover, searching over unified embedding space makes data discovery modal agnostic. Therefore, Symphony is able to discover datasets in different modalities to answer one query, *e.g.*, the text data **P1** and table **T1** in Fig. 3 (Section 4)

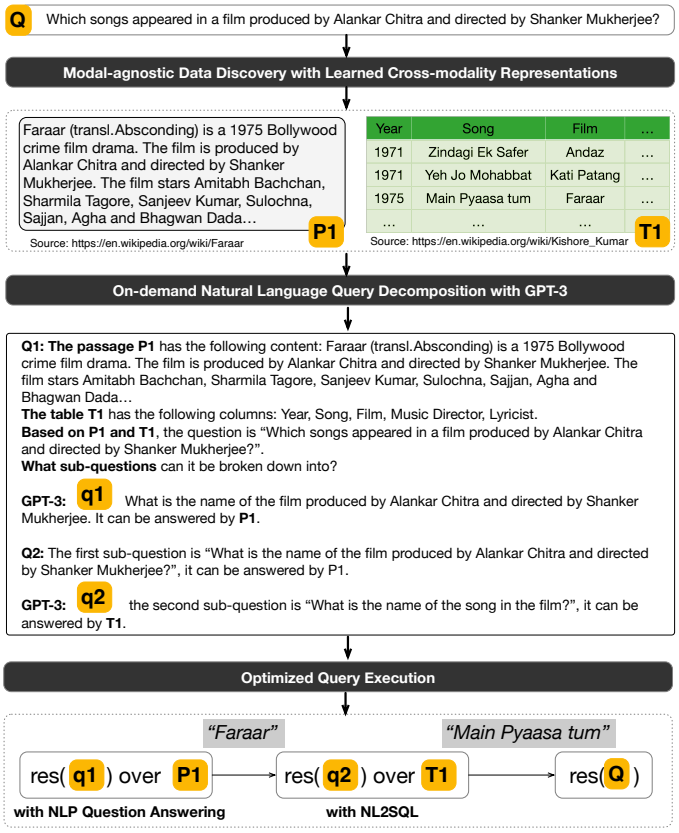


**Figure 3: A Running Example of Symphony.**

- ***On-demand Query Decomposition.*** Symphony decides whether and how to decompose an NL query into a set of NL sub-queries, such that each sub-query can be executed on one data source[1]. In this way, without having to go through the painful data integration process, Symphony uses multiple data sources in different modalities to answer one single query. For example, in Fig. 3, the query **Q** is decomposed into two sub-queries **q1** and **q2** using GPT-3, based on the text **P1** and table **T1**. (Section 5)
- ***Optimized Query Execution.*** Symphony features a multi-objective optimizer that takes both the accuracy and efficiency objectives into consideration, automatically selects the optimal execution plan for each (sub-)query, and aggregates the partial results in an appropriate order to produce the final answer. As shown in the bottom of Fig. 3, Symphony first evaluates **q1** on text item **P1** and gets the result "Faraar", which is then used as input to evaluate **q2** on table item **T1**. Finally Symphony gets the result of **Q** as "Main Pyaasa tum". (Section 6)

Different from the existing efforts that leverage large language models (LMs) to invent new data preparation (*i.e.*, civilization) tools

[1]A **data source** is a database of multiple tables, a silo-ed table, or a textual document.

such as Ditto [10] and RPT [15], Symphony solves the data problems in a fundamentally different direction, which for the first time allows the users to directly query a poorly maintained data lake, without needing data civilization efforts. Our initial experiments (Section 7) confirms that a preliminary implementation of Symphony already demonstrates promising results and thus validates our idea.

Enterprises can use Symphony in different scenarios. For example, the users could use the data discovery component of Symphony to better organize their data which is maintained poorly in their data lake, *e.g.,* by finding and annotating the tables that are relevant to the export business of the company at a low cost. Or they could use Symphony to understand how the sale of a specific product in this season improves compared to last season or if the labor costs of the company increase this year.

## 2 PRELIMINARIES

**Transformer-based Encoders.** Transformer-based language models (LMs) have achieved tremendous success for textual data. Pre-trained on a massive amount of text corpora, LMs like BERT, BART, T5, GPT-3, etc., can effectively learn common sense knowledge. *Transformer-based encoders* are widely used for representation learning. They convert an input object $x$ into an embedding representation as a high-dimensional vector $\mathbf{x}$, *i.e.,* $\mathbf{x} = f(x)$, where $f(\cdot)$ is the encoding function represented by the LM encoder.

**Pre-training.** LMs are usually *pre-trained* on large corpora to learn general knowledge. One popular strategy is *self-supervised pre-training* that does not require manual annotations. For example, masked language modeling [4] creates training data by masking or replacing certain tokens (words) in the original data, and then the LMs are trained to restore the original data given the corrupted data. Another common strategy is *multi-task pre-training* that trains the LMs on multiple existing tasks or datasets with annotations.

**Fine-tuning.** *Fine-tuning* effectively adapts pre-trained LMs to downstream tasks using task-specific objective functions and datasets. For example, Ditto [10] fine-tunes pre-trained BERT using entity resolution (ER) benchmarks and well serves ER tasks.

**Prompt Learning.** Recent research has shown that the *"pre-train, fine-tune"* paradigm can not fully explore the knowledge learned in pre-training due to the gap between the pre-training objective and the fine-tuning objective. To solve this problem, the *"pre-train, prompt, predict"* paradigm (*i.e.,* prompt learning) [11] instead reformulates the downstream task into a form similar to pre-training tasks with the aid of a textual prompt. A prompt corresponds to a natural language question template submitted to the LM. For example, two prompts "`Answer the following query:`" and "`Translate the following query to SQL:`" specify two different tasks and thus expect different answers from the LM. An LM is more likely to produce the correct answer if the prompt makes it easy for the LM to answer the question. Hence, it is important to provide appropriate prompts to effectively guide LMs.

## 3 CROSS-MODAL REPRESENTATION LEARNING

The cross-modal representation learning component of Symphony encodes items in the data lake into the same high-dimensional embedding space even if they are in different modalities.

To learn a cross-modal representation model, the most intuitive way is to first convert the items in different modalities into sequences of words and then directly train an LM on these sequences using the traditional self-supervised or multi-task methods. Taking tables, for example, one way to serialize a table is to concatenate its rows cell by cell, *e.g.,* the table T1 in Fig. 3 can be serialized as:

```
Year | Song | Film | ... || 1971 | Zindagi Ek
Safer | ... || 1971 | Yeh Jo Mohabbat | ... ||
```

However, directly using the existing pre-training methods to learn an LM is not effective in encoding a multi-modal data lake. Although the self-supervised pre-training method is good at producing embedding for each token, it simply aggregates the token level embeddings to produce a sequence level embedding which does not necessarily well represent the whole input sequence. On the other hand, the embeddings produced by multi-task pre-training are tightly-coupled with the specific queries or task objectives, because the pre-trained models tend to only extract the features that are sufficient to support these queries. This results in information loss because supporting these queries usually does not need to extract all information that perfectly represents the original input. Therefore, the learned representations tend to be ineffective at supporting new queries.

**Self-supervised Information Compression.** To solve the above problems, Symphony introduces *self-supervised information compression* as a new pre-training task to produce *query-agnostic* representations that preserve as much vital information of the data items as possible. Symphony thus only needs to pre-compute the query-agnostic representations once rather than re-computing them for every incoming query $Q$.

To achieve this goal, the encoder model has to be able to generate fixed-size embeddings that are sufficient to restore the original inputs. Because AutoEncoders [14] are remarkably successful in compressing vital information for the restoration of original data, self-supervised information compression trains an LM in the AutoEncoder fashion. More specifically, in the LM, the output of the encoder is constrained to a fixed-size embedding before sending it to the decoder, while the decoder is required to regenerate the entire input relying only on this embedding. This strategy leverages the information compression ability of AutoEncoder, while still preserving the power of the Transformer architecture. After pre-training, Symphony uses the encoder as the cross-modal representation model, which is query-agnostic.

Symphony also builds an index upon the embeddings to speed up similarity search.

## 4 MODAL-AGNOSTIC DATA DISCOVERY

Given a natural language query $Q$, Symphony automatically discovers a set of data items that are most relevant to $Q$ at online query time. Combining traditional data discovery methods and the representation learning-based method, Symphony offers a systematic and unified approach to effectively and efficiently discover relevant items from multi-modal data lakes. It consists of three key

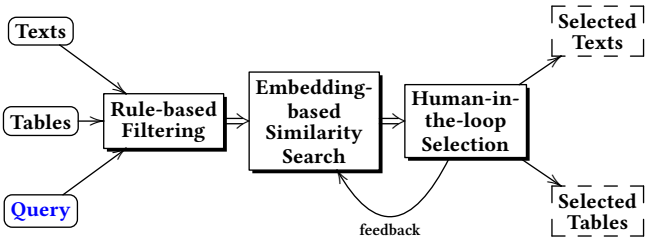


**Figure 4: Modal-agnostic Data Discovery.**

steps: rule-based filtering, embedding-based similarity search, and human-in-the-loop selection, as shown in Fig. 4.

**1. Rule-based Filtering.** Given a query $Q$, Symphony first uses traditional rule-based methods such as keyword search to quickly exclude data items that are clearly irrelevant. For example, tables can be filtered by keyword search on table names, column names, or metadata. Only a small set of candidate items will go to the next step for further examination.

**2. Embedding-based Similarity Search.** Symphony encodes the query $Q$ into the same high-dimensional space as the pre-processed multi-modal data lake representations. It then discovers the most relevant items from the candidate items based on the similarity between the query embedding and the item embeddings learned using the representation learning approach in Section 3, which can be efficiently computed via dot product when an index is available. For each modality Symphony returns the top-$k$ items whose embeddings are most similar to the query embedding.

**3. Human-in-the-loop Selection.** Finally, human interactions are introduced to select the most relevant items among the candidate items returned from the last step. Using the manual selection results as annotation, potentially Symphony can improve the cross-modal representation such that the similarity search results are better aligned with the human preference. Further, we believe Symphony can train a few-shot meta-scoring system that takes over the role of the human in the future, automatically ranks the candidates, and produces the final discovery results. The feedback of choosing from a few suggested candidates only requires a limited amount of human interactions, thus is much less labor intensive and more user-friendly than manual annotating the entire data lakes.

## 5 ON-DEMAND QUERY DECOMPOSITION

Given an NL query $Q$ and a set of discovered datasets $D = \{d_1, d_2, \ldots, d_n\}$ (each $d_i$ is either a table or a text passage), Symphony first decides whether it is necessary to decompose $Q$ (Section 5.1). If so, Symphony decomposes the query based on the objective defined in Section 5.2. We then introduce a method that uses GPT-3 to decompose an NL query in Section 5.3.

### 5.1 To Decompose or Not?

If two tables $d_i$ and $d_j$ have pre-defined PKFK relationship, *e.g.,* from the same database, we treat them as from the same data source and merge them into $d_k$. Recall that we use a *data source* to refer to either a silo-ed table, a database, or a text passage. This will convert the discovered items $D = \{d_1, d_2, \ldots, d_n\}$ into $D' = \{d'_1, d'_2, \ldots, d'_m\}$

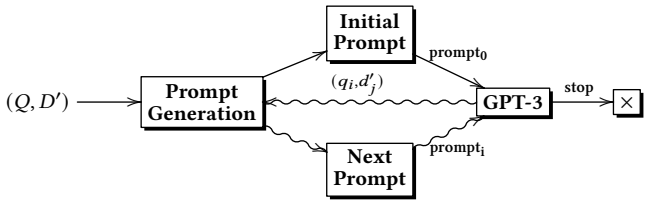


**Figure 5: Template-based Automatic Prompt Generation.**

| | |
|---|---|
| $\textbf{prompt}_0$ = | Serialize($d'_1$); Serialize($d'_2$); ... Serialize($d'_m$)<br>Based on $d'_1$, $d'_2$, ... and $d'_m$, the question is $Q$,<br>what sub-questions can it be broken down into? |
| $\textbf{prompt}_i$ = | The `[N]` sub-query is `[Q]`, it can be answered by `[D]` |

**Figure 6: Templates for Prompts.**

| Data Source | Serialization Template |
|---|---|
| **Database** | The database [DB_name] has the following tables: The first table [Tab_name] has the following columns: [Col_name], [Col_name], ..., [Col_name]; The second table [Tab_name] has the following columns: ... |
| **Table** | The table [Tab_name] has the following columns: [Col_name], [Col_name], ..., [Col_name]. |
| **Text** | The passage has the following content: **Text** |

**Figure 7: Templates for Data Source Serialization.**

where $m \leq n$, each $d_i \in D'$ represents one data source, and any two $d_i, d_j \in D'$ do not belong to the same data source.

Intuitively, if $m = 1$, there is only one data source, thus no need for query decomposition. Otherwise, Symphony uses multiple data sources to evaluate query $Q$. In this case, we choose to decompose $Q$ (see Fig. 3), instead of "integrating" these data sources.

### 5.2 Data Source-aware NL Query Decomposition

The problem of **data source-aware NL query decomposition** is to decompose an NL $Q$ into a set of NL sub-queries $\{q_1, \ldots, q_k\}$ such that each sub-query is evaluated using at most one data source $d'_i$. The sub-queries have partial order ("$\prec$"), deciding which sub-query should be executed first. In Fig. 3, the decomposed sub-queries are $\{\mathbf{q_1}, \mathbf{q_2}\}$. The sub-queries and their corresponding data sources are paired as $(\mathbf{q_1}, \mathbf{P1})$ and $(\mathbf{q_2}, \mathbf{T1})$, and the partial order is $\{\mathbf{q_1} \prec \mathbf{q_2}\}$.

To tackle the query decomposition problem in Symphony, one option is to use traditional NLP methods to convert an NLP query $Q$ into a parse tree that captures dependency of words/phrases and then devise new algorithms to decompose $Q$, taking the needed data items $D'$ into consideration.

The other option is to leverage the power of giant LMs such as GPT-3. Next, we will describe our design of prompt-based query decomposition using GPT-3 (Section 5.3), which demonstrates superior empirical result (Section 7).

### 5.3 Query Decomposition with GPT-3

**GPT-3.** The OpenAI's Generative Pre-trained Transformer version 3 (GPT-3) [1] is an autoregressive language model with 175-billion

parameters, pre-trained on a large text corpus. GPT-3 has shown to be able to adapt to new tasks based on task descriptions (*i.e.,* prompts) alone, with little to no labeled data [1, 12, 17].

**Prompts on GPT-3.** We can ask GPT-3 "any" single question (or prompt), such as **prompt1:** "who is Michael Jordan". We can also ask GPT-3 a sequence of questions in a conversational fashion, such as appending to the last question **prompt2:** "where is he living now", and **prompt3:** "what is the population there".

For Symphony, in order to decompose an NL query $Q$ *w.r.t.* the set of data sources $D'$, we need to automatically generate prompts such that GPT-3 can decompose the query $Q$ into a set of sub-queries, each of which is executed on one single data source.

**Prompt-based Query Decomposition on GPT-3 (Fig. 5).** Given $Q$ and $D' = \{d'_1, \ldots, d'_m\}$, we propose automatic template-based prompt generation as depicted in Fig. 5. Initially, we use the ***Initial Prompt*** ( $\textbf{prompt}_0$ ) function to generate the first prompt and feed it to GPT-3 which returns as the answer a sub-query and the data source on which it should run. Based on the returned sub-query and the data source, it iteratively invokes the ***Next Prompt*** ( $\textbf{prompt}_i$ , $i > 0$) function to generate the next sub-query, until GPT-3 decides to terminate, *i.e.,* no more sub-query needs to be generated. The two functions are discussed as follows.

***Initial Prompt***: Symphony generates the initial prompt using the $\textbf{prompt}_0$ template in Fig. 6. It first serializes each data source $d'_i$ in $D'$, where the methods for database/table/text serialization are provided in Fig. 7. It then adds the NL query $Q$ the user posed into the prompt and appends a fixed sentence "`what sub-queries can it be broken down into?`" at the end.

***Next Prompt***: Symphony then iteratively invokes the $\textbf{prompt}_i$ template in Fig. 6. It needs to fill three blanks `[N]`, `[Q]` and `[D]` based on the previously produced sub-queries. `[N]` indicates the order of the sub-queries (*e.g.,* `[N]` could be set to "first", indicating the first generated sub-query), `[Q]` is the previous sub-query generated by GPT-3, and `[D]` is the data source used by the previously generated sub-query.

Next, we use an example to illustrate how to generate prompts.

Example 1. *Consider the query $Q$ and two discovered datasets, text* **P1** *and table* **T1**, *as shown in Fig. 3.*

**[*Initial Prompt.*]** *After serializing the two datasets based on the rules defined in Table 7 Symphony produces the initial prompt using the prompt template in Fig. 6:*

> $\textbf{prompt}_0$ = *The passage P1 has the following content: Faraar (transl.Absconding) is .... The table T1 has the following columns: Year, Song, .... Based on P1 and T1, the question is "Which songs appeared in a film produced by Alankar Chitra and directed by Shanker Mukherjee?". What sub-questions can it be broken down into?*

*Symphony sends the prompt* $\textbf{prompt}_0$ *to GPT-3, and GPT-3 will generate a sub-query* **q1** *as well as the data source on which it should be evaluated, e.g.,* **P1**, *as shown in Fig. 3.*

**[*Next Prompt.*]** *Based the first sub-query* **q1** *and the data source* **P1**, *it will use the next prompt template to generate:*

> $\textbf{prompt}_1$ = *The first sub-query is "what is the name of the film produced by Alankar Chitra and directed by Shanker Mukherjee?", it can be answered by P1.*

*Given* $\textbf{prompt}_1$, *GPT-3 will generate sub-query* **q2** *and specify the table* **T1** *as its data source (see Fig. 3). Afterwards, GPT-3 decides to stop because it considers the original query $Q$ has been answered.*

With regards to the scalability issue that it is hard to serialize a large table or database, we sample a few rows per table to learn from big tables with Transformer-based models [18].

# 6 OPTIMIZED QUERY EXECUTION

Note that on-demand query decomposition may result in either one NL query (*i.e.,* no decomposition) on one data source or decomposed multiple NL sub-queries. The query execution engine of Symphony executes each (sub-)query on one data source with optimized efficiency and accuracy, and then combines the results from multiple (sub-)queries if needed. Symphony leverages existing techniques as well as inventing highly performant new querying techniques over text, a table, or a database.

## 6.1 Question Answering Over Text

In NLP, question answering (QA) has made significant progress, but they are still limited in supporting queries that aggregate textual facts, and problems in processing noisy data and numerical operations, the existing works are insufficient in supporting database style [16] queries which require reasoning over sets of relevant facts with operations such as filtering, aggregation, and join. An example of such a query is "Count all female database researchers who were born in the 1990s".

To solve this problem, Symphony offers techniques to support database reasoning over text. Similar to NLDBs [16], Symphony first discovers a set of relevant facts, runs neural SPJ operators on each relevant fact in parallel, and aggregates the results. The neural SPJ operator is trained as a Seq2Seq model to generate intermediate results from each relevant fact and a given query.

## 6.2 Querying a Single Table or a Database

**Table QA.** Table QA using neural models has been widely studied but existing table QA techniques are not scalable to big tables. To address this efficiency issue, we design a new technique that learns query-independent representation by introducing self-supervised information compression as a new pretraining task. In this way, all queries share the same table representation and Symphony only needs to conduct efficient inner product operation to produce answers, thus eliminating the performance bottleneck of table QA.

**NL2SQL**. Symphony offers NL2SQL as another way to support queries over a single table or a database in case the users require the *exact* answers which table QA cannot deliver. In addition to the existing NL2SQL techniques [6, 9], Symphony leverages the base foundation model, in particular GPT-3, as well as the prompting techniques to convert NL queries to SQL queries, using similar ideas we introduced in query decomposition.

## 6.3 The Query Optimizer

We propose to design a multi-objective optimizer that given a (sub-)query, automatically produces an optimal execution plan, taking both the accuracy and efficiency objectives into consideration.

Symphony optimizer combines cost models and rule-based optimizations to meet the accuracy requirement of the users, while

minimizing the execution costs. For example, given an NL (sub-)query over a table, Symphony will use NL2SQL to execute this query if this table is indexed. Otherwise, if the users do not have a strong demand in accuracy, Symphony will select the most efficient execution plan, using the cost models to determine if the table QA operator we offer is quicker than the SQL operator on this table. If the users require exact results, Symphony will execute the query with NL2SQL without worrying about the execution costs.

### 6.4 Sub-Query Aggregation

After getting the sub-answer to each sub-query, the next challenge is how to aggregate the sub-answers to produce the final answer. We propose to again leverage GPT-3 to aggregate the sub-answers. Specifically, we reformulate the problem of aggregating sub-answers as a text QA problem, which can be effectively solved by gigantic foundation models [7]. Given a set of sub-queries $\{q_1, q_2, \ldots, q_n\}$ and their corresponding sub-answers $\{a_1, a_2, \ldots, a_n\}$, Symphony concatenates the sub-queries and sub-answers sequentially as the prompt of the QA task, and then asks GPT-3 to answer the original query $Q$ based on the prompt. Note that the sub-answer $a_i$ can be a single token, a short multi-token span, or even a list containing multiple values. Example 2 shows how Symphony handles a real-world case.

Example 2. *Consider the following query that involves a sum operation: "How many representatives were elected in the 2014 United States house of representatives elections in Rhode Island and South Carolina?". We use GPT-3 to aggregate the sub-answers without any manual rules.*

**Input =** *The answer to "How many representatives were elected in the 2014 United States house of representatives elections in Rhode Island?" is 2, the answer to "How many representatives were elected in the 2014 United States house of representatives elections in South Carolina?" is 7, How many representatives were elected in the 2014 United States house of representatives elections in Rhode Island and South Carolina?*
**Output =** *9*

## 7 INITIAL EXPERIMENT RESULTS

### 7.1 Experimental Setup

We conduct experiments on a dataset consisting of two types of data items, tables and texts, extracted from Wikipedia [2]. It covers $400K$ web tables. Each table has its page title and section title. It also includes $6M$ English passages. We manually design 18 user queries to query the dataset. Each has ground truth annotations regarding the related data items to be discovered, the sub-queries to be decomposed, and the final answers to the user queries.

### 7.2 Evaluation on Data Discovery

**Initial Implementation.** As discussed in Section 4, for each query, Symphony first uses $n$-gram similarity and noun phrase similarity to generate candidates from all the items. It then uses an LM to transform each data item into a vector embedding and selects from each modality the top-$K$ items with the largest similarity to the query embedding.

| |
|---|
| Instance 1: $Q$: Did the 2014 United States house of representatives elections take place on the same day in Rhode Island, South Carolina and Louisiana? $E = \{P_1, P_2, P_3\}$<br>$\rightarrow q_1$: Did the 2014 United States House of Representatives elections take place on November 4, 2014 in Rhode Island? $e_1 = P_1$; $q_2$: Did the 2014 United States House of Representatives elections take place on November 4, 2014 in South Carolina? $e_2 = P_2$; $q_3$: Did the 2014 United States House of Representatives elections take place on November 4, 2014 in Louisiana? $e_3 = P_3$<br>★ Score: 2 |
| Instance 2: $Q$: How many representatives were elected in the 2014 United States house of representatives elections in Rhode Island and South Carolina? $E = \{P_1, P_2\}$<br>$\rightarrow q_1$: How many representatives were elected in the 2014 United States house of representatives elections in Rhode Island? $e_1 = P_1$; $q_2$: How many representatives were elected in the 2014 United States house of representatives elections in South Carolina? $e_2 = P_2$<br>★ Score: 2 |
| Instance 3: $Q$: Which number is larger, the number of living former prime ministers of Tunisia or the number of former presidents of Ghana? $E = \{T_1, P_1\}$<br>$\rightarrow q_1$: What is the number of living former prime ministers of Tunisia? $e_1 = P_1$; $q_2$: What is the number of former presidents of Ghana? $e_2 = T_1$<br>★ Score: 2 |
| Instance 4: $Q$: What year was the first German film that won the Academy Award for Best Foreign Language Film released? $E = \{T_1, P_1\}$<br>$\rightarrow q_1$: What year was the film The Tin Drum released? $e_1 = T_1$; $q_2$: What was the first German film that won the Academy Award for Best Foreign Language Film? $e_2 = T_1$; $q_3$: What year was the first Academy Award for Best Foreign Language Film given out? $e_3 = T_1$<br>★ Score: 0 |
| Instance 5: $Q$: Did The Tin Drum and Kolya both win the Academy Awards for Best Foreign Language Film? $E = \{P_1, P_2\}$<br>$\rightarrow q_1$: whether The Tin Drum has ever won the Academy Award for Best Foreign Language Film? $e_1 = P_1$; $q_2$: whether Kolya has ever won the Academy Award for Best Foreign Language Film? $e_2 = P_2$<br>★ Score: 2 |
| Instance 6: $Q$: How many years elapsed between the premiere of season 10 and the premiere of season 5 of Law & Order : Special Victims Unit? $E = \{T_1, T_2\}$<br>$\rightarrow q_1$: What is the premiere date of season 10 of Law & Order : Special Victims Unit? $e_1 = T_2$; $q_2$: What is the premiere date of season 5 of Law & Order : Special Victims Unit? $e_1 = T_1$<br>★ Score: 2 |
| Instance 7: $Q$: Did You Let Me Walk Alone participate in the Eurovision Song Contest in 1996? $E = \{T_1, P_1\}$<br>$\rightarrow q_1$: What year was the Eurovision Song Contest held in Lisbon, Portugal? $e_1 = T_1$; $q_2$: What year was You Let Me Walk Alone released? $e_2 = P_1$<br>★ Score: 1 |
| Instance 8: $Q$: Are the tallest building in the united kingdom and the tallest building in poland above 200 meters? $E = \{T_1, T_2\}$<br>$\rightarrow q_1$: What is the height of the tallest building in the United Kingdom? $e_1 = T_1$; $q_2$: What is the height of the tallest building in Poland? $e_2 = T_2$<br>★ Score: 2 |

**Table 1: Example sub-queries generated by Symphony. $q_i$ and $e_i$ represent the $i_{th}$ sub-query and its corresponding data item. $T_i$ represents a table and $P_i$ represents a text.**

**Evaluation Results.** We use the recall at $K$ ($R@K$) metric to measure our data discovery results. $R@K$ is the proportion of the relevant data items found in the top-$K$ recommendations. The experimental results show that, when $K$ = 5, 10, 15 and 20, the values of $R@K$ are 40.8%, 46.3%, 59.3% and 77.8%, respectively.

Next, we further analyze the case of $K \leq 20$, because it represents an acceptable amount of manual annotation effort. In the future, such manual inspection will be replaced by a meta-scoring system, as introduced in Sec. 4. For 12 out of the 18 queries, Symphony correctly discovers all the related items required to answer the queries. For 4 queries, it discovers part of the required items. Among a total of 38 items related to all the queries, 30 items are correctly discovered. Note we believe the results are encouraging because the current implementation is still preliminary and thus has large room to improve. This indicates that the new data discovery methodology we propose is promising.

### 7.3 Evaluation on Query Decomposition

**Implementation Details.** As introduced in Section 5, for each query, Symphony serializes the data items discovered in Section 7.2 and then combines it with prompt as the input of GPT-3. The output of GPT-3 is the generated sub-queries and the data item ID corresponding to each sub-query. We use the OpenAI API to run the experiments with GPT-3.

**Human Evaluation Results.** Similar to [8], we evaluate the quality of the query decomposition result based on two criteria: (1) if each sub-query is useful for solving the original complex query; (2) if the sub-query can be correctly answered on the selected data item. For each query, if both criteria are met, the score is 2; if only the first criterion is met, the score is 1; otherwise, the score is 0. Based on the above criteria, in all queries, 77.8% scored 2, 16.7% scored 1, and the remaining 5.5% scored 0. Table 1 shows the results of 8 instances. Symphony is able to handle different aggregation operations, such as sum (Instance 2) and comparison (Instance 3). Further, it correctly understands long sentences (Instance 1). However, Symphony has difficulty in dealing with sentences with complex syntactic structures (*e.g.,* in Instance 4, GPT-3 mistakenly generates $q_3$ because it takes the subject of "released" as "Academy Award for Best Foreign Language Film".). This is a direction worth exploring and optimizing in the future.

## 8 CONCLUSIONS

In this paper, we presented Symphony, a novel system towards supporting NL queries over multi-modal data lakes. The key difference between Symphony and existing data lake management systems is that Symphony supports on-demand query answering over massive collections of datasets with different modalities, without using the expensive data integration/cleaning operations to civilize data lakes beforehand. To support complicated NL queries Symphony discovers relevant multi-modal datasets, decomposes the query into a sequence of sub-queries, and evaluates each sub-query effectively and efficiently. In addition to presenting our vision and early achievement of Symphony, we also identify several promising research directions, such as better cross-modal representation learning, query decomposition, and optimization on top of the discovered multiple data sources.

## ACKNOWLEDGEMENTS

This work was partly supported by the NSF of China (62122090, 62072461, and U1911203).

## REFERENCES

[1] Tom B. Brown, Benjamin Mann, and et al. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165 (2020).
[2] Wenhu Chen, Ming wei Chang, Eva Schlinger, William Wang, and William Cohen. 2021. Open Question Answering over Tables and Text. *ICLR* (2021).
[3] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System. In *CIDR*.
[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[5] Michael Armbrust et al. 2020. Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. *Proc. VLDB Endow.* 13, 12 (2020), 3411–3424.
[6] Zihui Gu, Ju Fan, Nan Tang, Preslav Nakov, Xiaoman Zhao, and Xiaoyong Du. 2022. PASTA: Table-Operations Aware Fact Verification via Sentence-Table Cloze Pre-training. In *EMNLP*.
[7] Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. UnifiedQA: Crossing Format Boundaries With a Single QA System. In *EMNLP*.
[8] Tushar Khot, Daniel Khashabi, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2021. Text Modular Networks: Learning to Decompose Tasks in the Language of Existing Models. In *NAACL-HLT*.
[9] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *PVLDB* (2020).
[10] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *PVLDB* (2020).
[11] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv:2107.13586* (2021).
[12] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *CoRR* abs/2205.09911 (2022).
[13] El Kindi Rezig, Lei Cao, Michael Stonebraker, Giovanni Simonini, Wenbo Tao, Samuel Madden, Mourad Ouzzani, Nan Tang, and Ahmed K. Elmagarmid. 2019. Data Civilizer 2.0: A Holistic Framework for Data Preparation and Analytics. *Proc. VLDB Endow.* 12, 12 (2019), 1954–1957. https://doi.org/10.14778/3352063.3352108
[14] David E. Rumelhart and James L. McClelland. 1987. *Learning Internal Representations by Error Propagation.* 318–362.
[15] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Samuel Madden, and Mourad Ouzzani. 2021. RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation. *PVLDB* (2021).
[16] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Halevy. 2021. Database Reasoning Over Text. In *ACL*.
[17] Immanuel Trummer. 2022. CodexDB: Generating Code for Processing SQL Queries using GPT-3 Codex. *CoRR* abs/2204.08941 (2022).
[18] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL*.