ELEET: Efficient Learned Query Execution over Text and Tables

[Technical Report — Extended Version of [62]]

Matthias Urban
Technical University of Darmstadt
matthias.urban@cs.tu-darmstadt.de

Carsten Binnig
Technical University of Darmstadt & DFKI
carsten.binnig@cs.tu-darmstadt.de

ABSTRACT

In this paper, we present ELEET, a novel execution engine that allows one to seamlessly query and process text as a first-class citizen along with tables. To enable such a seamless integration of text and tables, ELEET leverages learned multi-modal operators (MMOps) such as joins and unions that seamlessly combine structured with unstructured textual data. While large language models (LLM) such as GPT-4 are interesting candidates to enable such learned multi-modal operations, we deliberately do not follow this trend to enable MMOps, since it would result in high overhead at query runtime. Instead, to enable MMOps, ELEET comes with a more efficient small language model (SLM) that is targeted to extract structured data from text. Thanks to our novel architecture and pre-training procedure, the ELEET-model enables high-accuracy extraction with low overheads. In our evaluation, we compare query execution based on ELEET to baselines leveraging LLMs such as GPT-4 and show that ELEET can speed up multi-modal queries over tables and text by up to 575× without sacrificing accuracy.
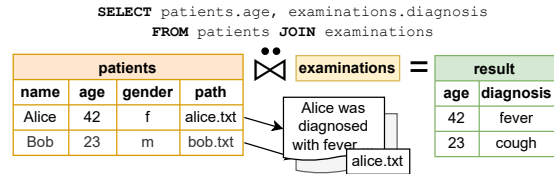
Figure 1: Example of a query that executes a multi-modal join between a patient table and examination reports. ELEET analyzes the texts and extracts values for each queried attribute, such as the diagnosis from each examination report.

1 INTRODUCTION

More than Tables. Decades of research have turned relational databases into highly optimized systems for managing tabular data. However, modern data applications need to deal with other data modalities as well that are often used in addition to tabular data, such as texts or image data [8, 23, 57]. Unfortunately, traditional relational databases are not well-equipped to handle these multi-modal scenarios. Instead, practitioners are forced to process modalities other than tables outside the database or integrate them by manually transforming such modalities into tabular form first.

Query Execution over Multi-modal Data. At the same time, rapid advancements in natural language processing and computer vision have made it easier to extract insights from texts, images, as well as other modalities. In light of these developments, we believe it is time to bring these innovations to the world of databases and enable users to seamlessly query multi-modal data. Although some extensions have been integrated into commercial database systems such as full-text search or pattern matching for textual data [22], modalities such as text do by far not allow for the same level of querying as tabular data. Our work aims to fill this gap. Hence, we propose ELEET, an approach that leverages query plans with learned operators that allow us to seamlessly process data of modalities other than tables as if they were available in tabular form.

A Simple Example. Figure 1 illustrates how we envision how ELEET can be used by applications. In the example, the database stores structured patient information (using a table) alongside textual patient reports that contain additional diagnostic information per patient. If the diagnostic information were stored in tabular form as well, a SQL query, as shown at the top of Figure 1, could easily be used to analyze the correlation between the patient's age and her diagnosis. If the information is, however, stored inside textual reports, today a data scientist would need to write many lines of code to create a data extraction pipeline that retrieves the diagnostic information from text. Only after extraction would it then be possible to query this information at a similar level to tabular data.

Learned Multi-Modal Operators. The goal of our work is to challenge this need for a "special treatment" for modalities other than tables, allowing users to query them seamlessly and declaratively as if they were tables. To enable seamless querying of multi-modal data, we propose to extend relational query plans with so-called learned multi-modal operators (MMOps). The basic idea of MMOps is that they extend the set of operators used in traditional query engines by new operators that can natively process data sources of other modalities. As shown in Figure 1, for example, a multi-modal join operator for tables and texts allows users of ELEET to join the patient table directly with the linked patient reports. As such, the data analyst can correlate the relationship between the patient's age and their diagnoses in a simple and efficient manner.
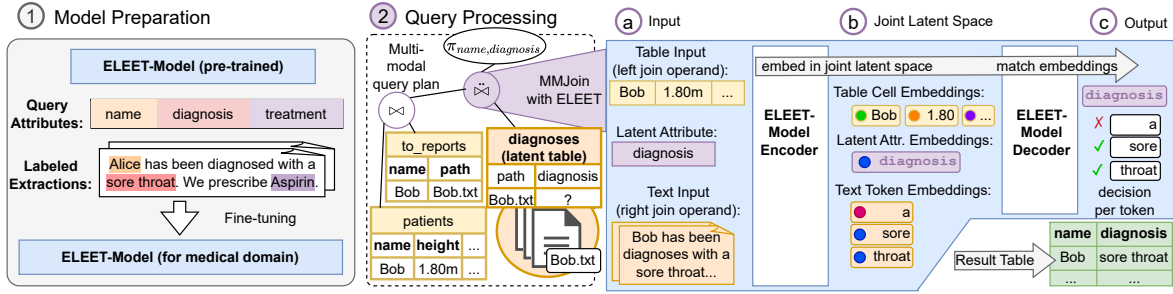
Figure 2: Overview of ELEET. In an offline phase, the ELEET-model can be fine-tuned for unseen domains ①. Fine-tuning the ELEET-model for an unseen domain is a one-time effort and requires a small sample of a few labeled texts. ② For query execution, ELEET uses multi-modal query plans that contain traditional (white) and multi-modal database operators (purple). To compute the result of a multi-modal operation such as a join over texts, the ELEET-model is used (see ⓐ to ⓒ): During the execution of a multi-modal operation, the ELEET model first computes embeddings of the query attributes, texts, and table input ⓐ, using its encoder ⓑ. Afterwards, the ELEET-model matches text token embeddings to query attribute embeddings to extract the output table from the text using its extractive decoder ⓒ, which decides which tokens qualify for a given query attribute.

Multi-Modal Query Plans. The rationale behind MMOps, such as the multi-modal join, is that they can accept data sources of other modalities as input and produce tables as an output. Thus, MMOps nicely integrate with the existing query processing capabilities of traditional databases since MMOps can be composed into query plans along with relational operators to enable complex analytical queries. For example, after the multi-modal join operator shown in Figure 1, other relational operators, such as a projection or a filter, can be applied to provide rich query functionality to users. Moreover, as we elaborate later in the paper, in addition to multi-modal joins, ELEET implements a wide spectrum of different MMOps, including multi-modal scans, unions, and aggregations.

Realizing Multi-Modal Operators using LLMs? The key idea in realizing such multi-modal operators for text and tables is to build on recent advances in the area of language models. In fact, recent language models (e.g., GPT-3 [4], GPT-4 [41], LLaMA [58, 59], PaLM [2, 9] or Gemini [56]) have shown remarkable results on a wide range of text-processing tasks. While it has been shown that recent language models can be used out-of-the-box to transform a text into a table, we argue that such language models are not readily usable for efficient query execution. In particular, due to their immense size (i.e., GPT-4 has more than a trillion parameters), they are very computationally expensive. Each call for a single text can take several seconds, leading to query runtimes of multiple minutes even for small text collections as we show in our experiments.

Small Language Models to the Rescue? Therefore, in this paper, we take a different route: instead of building ELEET on large language models (LLMs) such as GPT, we instead base multi-modal operators on a small language model (SLM) to achieve a more efficient execution of multi-modal operators. A key to this SLM is that we use a model architecture that targets table extraction from text and pre-train the model to learn the essential skills to perform MMOps. Thanks to this pre-training, the ELEET-model can provide high accuracy and efficiency at the same time. For example, our model architecture avoids using costly autoregressive decoding, which is prominent in LLMs and requires many passes through the model to construct the output rows from texts. Instead, our model uses an extractive approach using embeddings of text and query attributes, which can extract data from text in a single model pass.

More than table extraction from text. Finally, a last important property of our model is that it can incorporate additional signals from tabular data sources when extracting structured data from text, which can improve the extraction quality. For example, when executing a multi-modal join between a table and a text collection, as discussed before in our example, the multi-modal join can take structured data (e.g., the name of a patient) as input to extract the correct diagnosis from the text. This might be particularly helpful in scenarios where the text contains information about multiple patients. Moreover, providing such signals from structured data can also reduce the runtime of multi-modal joins, as we discuss later.

Contributions. To summarize, in this paper, we present three major contributions: (1) As the core contribution, we present the ELEET-model. For realizing the ELEET-model we use a novel SLM that targets table extraction from text. For pre-training the SLM, we construct a new parallel pre-training corpus of tables and texts. (2) As a second contribution, we show how MMOps can be realized using a pre-trained ELEET-model. We present a wide range of different MMOps such as a multi-modal scan that can turn text collections into tables or more complex operations like joins, unions, selections, and aggregations operating on texts and tables. (3) Finally, we provide an extensive evaluation using four data sets to challenge our approach and evaluate how accurate and efficient multi-modal query plans are executed with ELEET. The evaluation uses data sets from different domains and a wide range of multi-modal query plans. Using these workloads, we compare ELEET against strong baselines, including some that leverage LLMs such as GPT-3 or 4.

Outline. We first provide an overview of ELEET in Section 2, before we define its data model and algebra in Section 3. Then, we explain the details of the ELEET-model in Section 4 and how it can be used to realize MMOps in Section 5. Finally, we present our evaluation in Section 6, related work in Section 7 and conclude in Section 8.

2 OVERVIEW OF ELEET

In the following, we first explain the overall procedure of executing queries with ELEET. After that, we discuss its key design principles.

2.1 Overall Procedure

ELEET executes a multi-modal query plan containing traditional database operators and MMOps. Such a multi-modal query plan consumes data from tables and text collections that can be seamlessly treated as tables (called latent tables) using our ELEET-model. ELEET supports different MMOps in multi-modal query plans. For instance, the query $\pi_{\text{name,diagnosis}}((\text{patients} \bowtie \text{to\_reports}) \bowtie \text{diagnoses})$ in Figure 2 uses a multi-modal join to combine text with table data. In the following, we sketch how this join can be realized by using our ELEET-model in Figure 2 ②.

A Sketch of a Multi-Modal Join. The join in Figure 2 needs to extract the diagnosis for each patient tuple coming from the first join of the *patients* and *to_reports* table (i.e., each patient can have multiple reports). For executing this query, we feed the attributes to extract from text (i.e., *diagnosis*; called *latent attribute*) together with the patient data from the first join and the text documents to be joined into the ELEET-model. For example, for joining the patient tuple of Bob with his patient report in Figure 2 ⓐ, ELEET feeds the patient tuple (containing name, height, ...), the latent attribute *diagnosis*, and the patient report of Bob into the ELEET-model. For extracting the diagnosis, the encoder of our ELEET-model maps all inputs into a joint latent space ⓑ. Afterwards, the decoder identifies spans of texts in the report that qualify as diagnosis, such as the text span *sore throat* in Figure 2 ⓒ. Finally, the result row {name ↦ Bob, diagnosis ↦ sore throat} with the extracted values from the text is materialized. One assumption we make in ELEET is that there exists a foreign key relationship between the tuples in the relational table and the text collection, i.e., patient reports are linked to a patient tuple. In Section 3 we discuss the data model of ELEET more formally.

Fine-Tuning for unseen Domains. While the ELEET model is pre-trained to learn table extraction from text, it clearly benefits from fine-tuning on texts of domains unseen during pre-training (see ① in Figure 2). To do so, in a model preparation phase (offline), the user labels a few example texts by marking text spans that are extractions for potential query attributes. Based on a pre-trained ELEET-model, only a few fine-tuning samples are necessary. In our evaluation, we show that only a small number of labeled documents are typically sufficient to achieve high accuracy for unseen domains.

2.2 Key Design Principles

Efficiency of Extraction. The efficiency of query execution is a major concern for ELEET. We tackle this using three key design principles for the ELEET-model: (1) Firstly, regarding the ELEET-model, we use a small language model with only 140 million parameters that we optimize for performing query operations. The ELEET-model is multiple orders of magnitudes smaller than recent LLMs (e.g., GPT-3 has 175 billion and GPT-4 has 1.76 trillion parameters) and thus provides much lower inference latencies. (2) Secondly, in our model architecture as we discuss next, we avoid costly autoregressive decoding that LLMs typically use, which requires many passes through a transformer-based decoder and thus

leads to high inference times as shown in Section 6.6. Instead, the ELEET-model uses an extractive approach that computes its output in a single pass through the model. (3) Finally, beyond the model itself, optimal physical operator implementations of MMOps in ELEET can help to improve the efficiency of query execution further. For example, for a multi-modal selection that applies a filter on attributes from the texts, we leverage an index-based implementation to avoid the high scan cost of scanning the full text for all documents in the collection.

Accuracy without Regrets. Another key design principle of ELEET is that we do not trade efficiency for accuracy. Instead, as we specialize our model for the task of extracting structured data from text, it is highly accurate on this task and often even more accurate than much larger general-purpose language models such as GPT-4. We achieve this through our pre-training procedure, which teaches the model the necessary skills to perform table extraction from text. This allows ELEET to be highly accurate while being more efficient than LLMs, as we show in our evaluation.

Online and Offline Execution While this paper aims to enable online execution of multi-modal query plans, ELEET can also be used to pre-compute extractions from text offline (i.e., constructing a materialized view of a multi-modal query). However, we think there are many scenarios where the ability to execute multi-modal query plans online is crucial. For example, in a setting where text collections are continuously updated, online query execution is important to provide up-to-date query results and can avoid the high cost of view maintenance. Moreover, online query execution can also be attractive in a setting where queries only need to process a few texts, and materializing a table of a potentially huge text collection would cause high overheads. In these cases, online processing saves the additional high storage and pre-processing overheads of materialization. Finally, materialization prevents ad-hoc queries where query attributes are not known in advance.

3 DATA MODEL AND ALGEBRA DEFINITION

In this section, we explain which types of queries are supported by ELEET by formally defining its data model and ELEET's algebra.

3.1 Data Model of ELEET

The data model of ELEET builds upon the relational data model and extends it by text collections and latent tables.

Table 1: Overview of ELEET's data model.

Symbol	Name
T_A	Table with attributes A
D	Text collection
$D.LT_{LA}$	Latent table with latent attributes LA

Text collections. A text collection $D = \{d_1, \ldots, d_{|D|}\}$ is a set of text documents. Similar to how different rows in database tables follow the same schema defining a set of attributes A per table, we assume that different documents d of a text collection D also expose the same attributes. For instance, in a text collection of patient reports, each document contains information about the patient's name and the diagnosis. Furthermore, each document is uniquely

identified by its file path $path(d)$ and can contain an arbitrary number of tokens: $d = w_1 \ldots w_{|d|}$. In ELEET, text collections can be queried standalone but can also be linked to traditional tables. Traditional tables can be linked to text collections by storing their file paths as an additional attribute in the tables, essentially creating a foreign key relationship between tables and text collections. In fact, many real-world datasets today already use this model to link text documents and tables. For example, we analyzed the GitTables corpus [24], a collection of 1M tables, and found that 15k tables already contain paths to externally stored text files.

Latent Tables. In ELEET, users register latent tables over a text collection D to make it available for query processing. Similar to regular tables, the user has to define a schema $LA = \{la_1, \ldots, la_{|LA|}\}$ for a latent table $D.LT$. We assume that the user knows the data well and can specify a reasonable schema. We denote such a latent table as $D.LT_{LA}$. The schema defines the attributes that can be extracted from text (e.g., name and diagnosis of medical reports). A tuple $t \in D.LT$ thus contains a value $v = w_1^v \ldots w_{|v|}^v$ that can be extracted from text $d = w_1 \ldots w_1^v \ldots w_{|v|}^v \ldots w_n$ for each attribute in LA. Furthermore, each tuple contains the file path to the document d. Importantly, defining a latent table does not yet extract any values from the text. Instead, a latent table is merely a handle that can be used in query plans. The values are extracted on the fly during query execution as explained in Section 2.2.

Single-row and Multi-row Latent Tables. Finally, in ELEET we distinguish between *single-row* and *multi-row* latent tables. A *single-row* latent table is where the user knows that each document $d \in D$ contributes exactly one tuple t to a latent table $D.LT$ (e.g., each patient report always contains a single diagnosis and treatment). On the other hand, a *multi-row* latent table is the general case where each document can contribute an arbitrary number of tuples to a latent table. In case of a *multi-row* latent table, the user has to define a *document-level key* $la_{key} \in LA$ such that each latent tuple t coming from the same document d is uniquely defined by the values for la_{key}. For example, if a medical report contains information about multiple diagnoses, the diagnosis name would be a sensible *document-level key*. Note that defining the schema, the type of latent table (*single-row* vs. *multi-row*) and the *document-level key* needs to be done only once per latent table and not per document, thus causing the same overhead as creating a schema for a normal table.

Moreover, we found that some of the decisions mentioned above, like choosing the type of latent table or choosing the *document-level key* can be automated using LLMs. For example, to decide whether a latent table is *multi-row*, we prompt an LLM with a few example texts together with the schema of the latent table and ask it whether each text contributes one or multiple rows to the latent table. Since choosing the type of latent table is done only once per latent table, as explained before, using a costly LLM is reasonable. See our experiments in Section 6.7 for details on the prompts and our findings.

3.2 Algebra of ELEET

ELEET uses an algebra to compose multi-modal query plans. The algebra of ELEET extends the traditional relational algebra with multi-modal operators summarized in Table 2 and formally defined in Section 3.4.

Table 2: Summary of the multi-modal operators.

Name	Expression	Output Type
Scan	$\ddot{S}can(D.LT_{LA})$	$T_{LA \cup \{\text{path}\}}$
Join	$T_A \ddot{\bowtie} D.LT_{LA}$	$T_{A \cup LA}$
Union	$T_A \ddot{\cup} D.LT_{LA}$	T_A
Projection	$\ddot{\pi}_{LA' \subseteq LA}(D.LT)$	$D.LT_{LA'}$
Selection	$\ddot{\sigma}_{cond}(D.LT_{LA})$	$D.LT_{LA}$
	$\ddot{\sigma}_{cond}(T_A)$	T_A
Aggregation	$\ddot{\chi}_{F,A' \subseteq A}(T_A)$	T_A

Multi-modal Scan. The most important operator is the multi-modal scan operator. A multi-modal scan takes a latent table $D.LT_{LA}$ as input and materializes a normal table $T_{LA \cup \{\text{path}\}}$ as output by extracting values $v = w_1^v \ldots w_{|v|}^v$ for each latent attribute from all text documents. The output table of a multi-modal scan can thus be used as input to normal relational operators such as joins and filters. Important is that the multi-modal scan is sufficient for expressing all possible multi-modal queries in ELEET. As such, the other multi-modal operators in Table 2 do not enrich the expressivity of queries in ELEET but instead improve the quality of query results or the efficiency of query execution (or both). In the following, we briefly explain each multi-modal operator's interface, its role, and how it can optimize a multi-modal query plan. More details about the individual operators can be found in Section 5.

Multi-modal Join. The second operator is a multi-modal join $T \ddot{\bowtie} D.LT$, which can replace a combination of a scan with a traditional join $T \bowtie_{T.path = D.LT.path} \ddot{S}can(D.LT)$. For the multi-modal join, the table T must be linked to the document collection D by storing file $path$ as an additional attribute in T, which is used as a join key. The task of the multi-modal join is to extract values from text linked to a tuple in T by leveraging attribute values from the tuple. The multi-modal join patients $\ddot{\bowtie}$ reports.examinations, for example, extracts for each patient (stored in a table) values for the latent attributes of the latent *examinations* table (e.g., diagnosis, treatment, etc.) from the textual reports.

The multi-modal join is an optimization over using patients $\bowtie$ $\ddot{S}can(\text{reports.examinations})$ as the multi-modal join can use the data in the patients table during extraction from the textual reports. For example, the patient table containing the patient name can help extract the relevant parts of the medical report text, which is particularly helpful if the reports contain information about multiple patients.

Multi-modal Union. The third multi-modal operation is the union $T \ddot{\cup} D.LT$, which can be used to replace a traditional union and a scan $T \cup \ddot{S}can(D.LT)$. For instance, when a hospital stores its patient information in tabular form, while another stores it as reports, one could combine both with a multi-modal union. Important is that the attributes exposed by the normal and latent table are compatible in types, meaning they store the same type of information (e.g., both store patient name and diagnosis). Again, the difference to a scan is that the union can use the tabular data T as additional context for the model. In the case of a union, this additional context is example values (e.g., example names and diagnoses) for each latent attribute, which can help extract values from the text.

Multi-modal Projection & Selection. The multi-modal projection $\ddot{\pi}_{LA'}(D.LT)$ projects the columns from a latent table without materializing it. As such, it is an optimization over using traditional projection after a multi-modal scan $\pi_{LA'}(\ddot{S}can(D.LT))$, which would need to materialize values for all columns. In contrast, the multi-modal selection $\ddot{\sigma}_{cond}(D.LT)$ as shown in Table 2 (first variant) reduces the number of texts using a filter condition *cond* on a latent attribute. Important is that the text documents are filtered without materializing them as output table. As such, it is an optimization over using traditional selection after a multi-modal scan $\sigma_{cond}(\ddot{S}can(D.LT))$ which would first need to extract rows for all documents before filtering. Moreover, another feature of the multi-modal selection is that it improves selection quality since it can detect matches of text values similar to the value used by the filter condition *cond* (e.g., diagnosis=fever can also match the synonym high temperature in text). The second variant of the multi-modal selection $\ddot{\sigma}_{cond}\ddot{S}can(T)$ is different since it uses a normal table as input. However, it can still detect matches of similar values in filter predicates and table values. This selection can be used in case a selection can not be pushed down to the text. An example is a disjunctive selection on the output of a multi-modal join, which filters based on attributes of the table and the latent table (e.g., patient.age=18 OR reports.examinations.diagnosis='fever').

Multi-modal Aggregation. Finally, similar to the second selection, multi-modal aggregation $\ddot{\chi}_{F,A'}(T)$ operates on a normal table but can group similar values for attributes coming from text extractions. It replaces a traditional aggregation over a multi-modal scan or join, which is less robust if extracted values contain synonyms. Like traditional aggregation, multi-modal aggregation takes parameters F for the aggregation function over one of the attributes in T and A' for the group-by attributes.

3.3 Algebraic Rewrites in ELEET

As discussed before, interfaces of multi-modal operators are designed so they can often be used to optimize query execution of multi-modal query plans both in terms of runtime and accuracy. More specifically, we imagine a query compiler that, given a user query (e.g., in SQL), will first instantiate a query plan that only contains multi-modal scans and traditional operators. Since the scan output is a normal table, and afterwards traditional operators can be composed arbitrarily, the system supports arbitrarily complex SQL queries where data sources can be text exposed as latent tables and normal tables. However, in an optimization step, the query compiler can look for patterns in the query plan where the plan can be optimized by inserting another multi-modal operation (e.g., multi-modal joins, unions, filters, etc.). For instance, if the user inputs the SQL query from Figure 1, the query compiler would first instantiate the multi-modal query $\pi_{age,diagnosis}(\text{patients} \bowtie \text{Scan}(\text{examinations}))$, that contains a multi-modal scan on the latent table *examinations*. Afterwards, the compiler would notice that using the data in the *patients* table helps extract the diagnoses of the correct patients from the text. Moreover, materializing all columns of the *examinations* table using the scan is also unnecessary because the user is only interested in diagnoses. Hence, the compiler replaces the join and the scan with a multi-modal join that extracts values

from text while leveraging the context from the tabular operand. Afterwards, it pushes down the projection and replaces it with a multi-modal projection, preventing all columns from being materialized. Hence, the result query plan that is optimized with multi-modal operators is: $\pi_{age,diagnosis}(\text{patients} \ddot{\bowtie} \ddot{\pi}_{diagnosis}(\text{examinations}))$. Similarly, multimodal unions could be inserted when users try to union a table with a latent table, and the quality of extractions can be improved with example values; multi-modal selections could be inserted if the user filters based on values extracted from text, and so on, as explained above. This example illustrates that ELEET supports arbitrary queries, and multi-modal operators can be used in many cases for optimization. However, while this shows the potential of using multi-modal operators to enhance the accuracy and efficiency of multi-modal plans, building an optimizer for multi-modal plans that uses cost models for estimating performance (and accuracy) is beyond the scope of this paper but represents an interesting opportunity for future research on multi-modal databases.

3.4 Formal Definition of Multi-Modal Operators

So far, we have restricted the formal definitions of the operators to their input and output sets (i.e., the domain and co-domain of the underlying function). In this section, we formally define the expected result of a multi-modal operator given its input data.

Multi-modal Scan (Single-row Latent Table). For the scan of a *single-row* latent table, the scan extracts for all latent attributes $la \in LA$, the "correct" values (i.e., as given by the ground truth) from each text $d \in D$. The output table R contains one output tuple per input text document d:

$$R = \left\{ \begin{array}{l} \{\text{path} \mapsto path(d)\} \\ \cup \{la \mapsto \text{extract}(d, la) | la \in LA\} \end{array} \middle| d \in D \right\}$$

In this definition, a single output tuple $t \in R$ is extracted from each document $d \in D$ by extracting a value $v = extract(d, la)$ for all latent attributes $la \in LA$. The expression $\text{extract}(d, la)$ denotes the extraction of a value v for the latent attribute $la \in LA$ (e.g., diagnosis) from document d and is a sequence of tokens

$$v = w_1^v \ldots w_{|v|}^v$$

(e.g. v = sore throat, w_1^v = sore, w_2^v = throat). The values are extracted directly from text d, meaning the token sequence of v appears in d as explained before:

$$d = w_1 \ldots w_1^v \ldots w_{|v|}^v \ldots w_n$$

Moreover, all result tuples $t \in R$ contain the special *path* attribute that contains the file path $path(d)$ of document d. The file path of a document d indicates where d is stored in the file system and uniquely identifies each document. This allows for further operations on the scan output, such as selections based on the file path. Another important property enabled by the file path is that we can join the output with a relevant table. As explained in Section 3.1, a table T can contain links to a text collection D by explicitly storing the file paths of text documents in one of its columns. For example, a patient table could store the file path to each patient's patient report. That way, a traditional table T (e.g., the patient table) can be joined with the output of a scan on a text collection (e.g., patient reports),

by joining on the path column using a traditional join operator: $T \bowtie_{T.path=D.LT.path} \ddot{S}can(D)$. However, ELEET also offers the multi-modal join as a more accurate and efficient alternative, which we define later.

Multi-modal Scan (Multi-row Latent Table). The definition is different for a *multi-row* latent table as each document can contribute multiple tuples to the output table. To enable this, the user has to define a *document-level key* la_{key} as explained in Section 3.

The *document-level key* is a special latent attribute that allows us to define an output table that consists of arbitrarily many tuples per input text. The *document-level key* can have multiple values for a single input text, and each of those values represents a different output tuple. For example, in a document collection of patient reports, where the same report can contain the diagnosis of multiple patients, la_{key} = patient_name is a sensible *document-level key*. In this example, if three patients with patient_name Alice, Bob, and Carol were mentioned in a single text, this would result in three output tuples for that text according to the definition below:

In this case, extract(d, patient_name) = {Alice, Bob, Carol} is a set of all patients mentioned in the one document d. Moreover, since each patient has their own diagnosis, we also need to define extract(d, v_{key}, la), which is the value for latent attribute la given the value v_{key} for the *document-level key*. For instance, if the diagnosis of Bob is fever, then extract(d, Bob, diagnosis) = fever. As such, we can define the expected scan output R as:

$$R = \left\{ \begin{array}{l} \{\text{path} \mapsto path(d)\} \\ \cup \{la_{key} \mapsto v_{key}\} \\ \cup \{la \mapsto \text{extract}(d, v_{key}, la) \\ \quad | \; la \in LA \setminus \{la_{key}\}\} \end{array} \middle| \begin{array}{l} d \in D \; \wedge \\ v_{key} \in extract(d, la_{key}) \end{array} \right\}$$

Like before, as we can see in the expression left of the vertical bar, each tuple $t \in R$ contains the special *path* attribute that contains the file path $path(d)$ of the document d. Additionally, it also contains the *document-level key* la_{key} as one of its attributes (e.g., la_{key} = patient_name) and one of its potential values v_{key} (e.g., Alice or Bob or Carol from the example before) is assigned to it. Finally, the output tuple contains values for all other latent attributes $la \in LA \setminus \{la_{key}\}$ extracted from text d. However, different from before, we can see in the predicate right of the vertical bar that for a single document d, the expected output contains multiple output tuples, one for each v_{key} mentioned in the document (e.g., one tuple for each patient if multiple patients are contained in the report).

Multi-Modal Join. As described in Section 3, the multi-modal operators beyond the multi-modal scan do not make multi-modal queries more expressive. Instead, they are optimizations to improve accuracy of the extractions. For example, a multi-modal join improves the extraction accuracy by using the tabular context of the structured table in the input.

To be more formal, the expected output of a multi-modal join is defined to be equivalent to the combination of a scan and a traditional join:

$$T \ddot{\bowtie} D.LT = T \bowtie_{T.path=D.LT.path} \ddot{S}can(D.LT)$$

As such, the formal definition of the multi-modal join is based on the formal definition of the multi-modal scan and *it is not necessary to provide a separate formal definition*.

Multi-Modal Union. Similarly, the expected output of a multi-modal union is equivalent to a traditional union with the output of a scan:

$$T \ddot{\cup} D.LT = T \cup \ddot{S}can(D.LT)$$

Again, the formal definition of the multi-modal union is based on the formal definition of the multi-modal scan and thus *it is not necessary to provide a separate formal definition*.

Multi-Modal Selection and Aggregation. As explained in Sections 5.3 and 5.4, the multi-modal selection and the multi-modal aggregation operator operate on normal tables as input (and not text documents). These operators aim to make the selection and aggregation more robust than their traditional counterparts by considering semantic synonyms. For instance, if a user selects all tuples with diagnosis *fever*, we also want to return patients with diagnosis *high body temperature*. Similarly, if the user aggregates with the group-by key *diagnosis*, we want to place *fever* and *high body temperature* in the same group. As such, *the definition of the multi-modal aggregation and the multi-modal selection is equivalent to their traditional counterparts* and again does not require an additional formal definition.

4 THE ELEET-MODEL

At the core of ELEET is the ELEET-model which is used for extracting structured tuples from text. In the following, we show the model architecture and how the model is pre-trained.

4.1 Model Architecture

Our ELEET-model uses an encoder-decoder architecture as depicted in Figure 3 that follows the design principles outlined in Section 2.2. In our model, the encoder first computes embeddings from the input. The decoder then consists of several lightweight decoder heads that use these embeddings to perform the different subtasks for transforming texts to tables (e.g., value extraction, deduplication). For the sake of presentation, we explain the design of our model and how it can be used to execute multi-modal joins or unions on a *single-row* latent table. For *single-row* latent tables, for each text document, we only need to extract a single value for each latent attribute (i.e., for each patient report, we need to extract one diagnosis, one treatment, etc.). More details about all operations and *multi-row* latent tables are explained in Section 5.

Encoder. The encoder computes embeddings for the inputs required to extract values from text documents. Figure 3 ① and Algorithm 1 shows an example of a multi-modal join. The input to the encoder consists of the text tokens coming from a text document (orange) and the latent attributes LA for which values need to be extracted from text (purple), which is the diagnosis attribute in Figure 3. Additional non-latent context can be fed into the model (yellow) that comes from the tabular operand of a multi-modal join or union (this is generally optional). For the multi-modal join example, the additional context is the tuples linked to the textual report. In Figure 3, we see two tuples (e.g., Alice and Bob) and two separate reports shown as text input (i.e., one per patient) from which we aim to extract the diagnoses. For unions, we use two rows randomly picked from the tabular operand of the union. The ELEET-model can use the additional values from the table as context for extractions, as we explain below. We use the same linearization
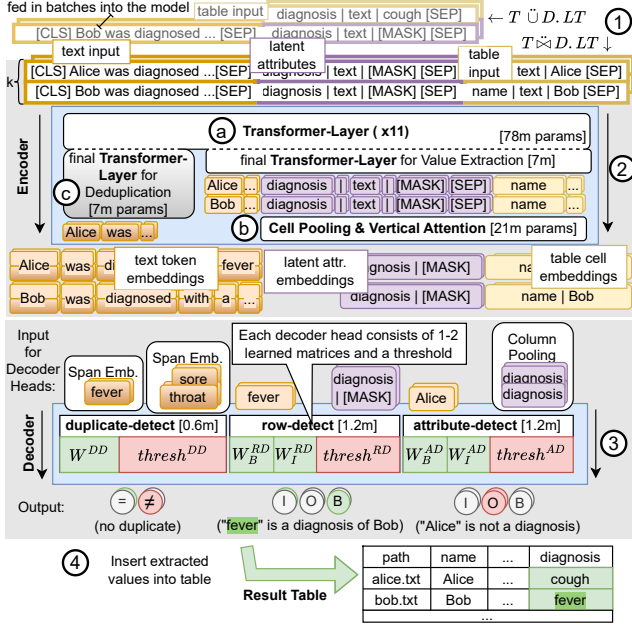
Figure 3: Model architecture. After encoding a batch of sequences that each contain an input text, the latent attributes, and traditional table values ① using 12 (11+1) transformer layers ② ⓐ, all embeddings corresponding to the same cells or latent attributes are pooled, before vertical attention lets signal flow between groups of k rows ⓑ. A separate final transformer layer computes a second set of text embeddings optimized for detecting duplicates ⓒ. The decoder ③ consists of several heads for the different sub-tasks for extracting table data from texts. For instance, the row-detect head is used to find extractions in the text. For this, it pairs the embedding of each text token with the embedding of a masked cell (i.e., the attribute to be extracted) and classifies whether the token is part of the attribute or not. The tokens that are marked to be extracted are inserted into the output table ④.

Algorithm 1 ① Encoder of ELEET for a multi-modal join. Scans and union differ in the table data that is fed in the model as explained in Section 4.

Require: texts $d_1 \ldots d_k$, latent attributes LA, table tuples $t_1 \ldots t_k$ (table tuples come from the tabular operand of the join; t_i is linked to d_i via the file path)

1: Feed the tuple-text pairs into 11 Transformer layers ⓐ

$$\hat{E} \leftarrow \text{Transformer}[\times 11](d_1 \,\|\, [SEP] \,\|\, \text{linearized}(LA) \,\|\, \text{linearized}(t_1),$$
$$\ldots$$
$$d_k \,\|\, [SEP] \,\|\, \text{linearized}(LA) \,\|\, \text{linearized}(t_k))$$

2: $\hat{E}^{DD} \leftarrow \text{Transformer}[\times 1]_{DD}(\hat{E})$ ▷ Embeddings for DD ⓒ

3: ▷ Discard non-text token embeddings

4: $(\hat{w}^{DD}_{1,1}, \ldots, w^{DD}_{k,n}), \_ \leftarrow \hat{E}^{DD}$

5: $\hat{E} \leftarrow \text{Transformer}[\times 1]_{RD,AD}(\hat{E})$ ▷ Embeddings for RD, AD

6: ▷ Separate in text and table embeddings

7: $(\hat{w}^{VE}_{1,1}, \ldots, w^{VE}_{k,n}), (\hat{e}_{1,1}, \ldots \hat{e}_{k,\bar{m}}) \leftarrow \hat{E}$

8: ▷ Take the mean of all embeddings belonging to the same cell

9: $(\hat{c}_{1,1}, \ldots \hat{c}_{k,m}) \leftarrow Cell-Pool((\hat{e}_{1,1}, \ldots \hat{e}_{k,\bar{m}}))$ ▷ ⓑ

10: $(\hat{c}_{1,1}, \ldots \hat{c}_{k,m}) \leftarrow \text{Vertical} - \text{Attention}((\hat{c}_{1,1}, \ldots \hat{c}_{k,m}))$

11: **return** $(\hat{w}^{VE}_{1,1}, \ldots, w^{VE}_{k,n}), (\hat{w}^{DD}_{1,1}, \ldots, w^{DD}_{k,n}), (\hat{c}_{1,1}, \ldots \hat{c}_{k,m})$

---

of name | type | value for the latent attributes and non-latent table values but use a MASK token for latent attributes to indicate that a value needs to be extracted for them, as shown in Figure 3 ①. As shown in Figure 3 ②, after linearization, the inputs are fed into the 12 transformer layers (see ⓐ) to compute their representations (i.e., embeddings). Multiple input sequences are fed into the model in a batch to increase efficiency. After running the input through 11 transformer layers, there are different paths for different decoder subtasks. In particular, the paths use different final transformer layers:

For *extracting values*, after running the individual rows through the transformer layers and obtaining embeddings for all input tokens, we pool all token embeddings belonging to the same table cell (i.e., the embeddings for attribute name, type, and value / MASK) into one cell embedding (cell pooling). For unions, it is important that signal flows from the rows containing example values to the embeddings of the masked cells to compute optimal embeddings

for extraction. Hence, we apply vertical self-attention [71] across different rows, i.e., across the cell embeddings of the same column (see ⓑ in Figure 3). In ELEET, we feed the sequences in groups of $k = 3$ into vertical self-attention [71]. Hence, for unions, we pair all sequences with MASK tokens and text with both sequences containing example values, meaning we extract values for a single text per vertical self-attention call.

The second path is for *detecting duplicates*, which is needed if text mentions the same extraction multiple times (e.g., fever and high body temperature for the same patient). For this, we use a final transformer layer that is separate from the one for extracting values (see ⓒ in Figure 3). This is because the two decoder subtasks benefit from embeddings of different characteristics. For value extraction, embeddings of text-tokens should be similar to those of the latent attributes for which they are a value. For detecting semantic duplicates, however, embeddings of text tokens that belong to different semantic concepts should be dissimilar, even though they are a value for the same latent attribute. Hence, this separate final transformer layer produces embeddings that the decoder can use to detect duplicate values more easily. To summarize, the encoder computes two embeddings per text token w (one for value extraction $\hat{w}^{VE}$ and one for duplicate detection $\hat{w}^{DD}$) and one embedding $\hat{c}_{i,j}$ per cell, where $1 \leq i \leq k$ and $1 \leq j \leq |LA|$.

Decoder. The decoder (Figure 3 ③) generates the output table by extracting a value for each latent attribute per text. It uses three lightweight decoder heads and the embeddings from the encoder.

For materializing the output of a join, for example, it is important to extract only values for the entities described by the tuple linked to each text. For instance, when we join a patient tuple (e.g., Bob) with a textual report of this patient, we typically only want to extract

Algorithm 2 ② Decoder of ELEET: **Row Detect**. The actual implementation computes I-O-B tags for all input sequences at once using matrix multiplication instead of a loop. **Attribute Detect** works analogously, using $\hat{a}_j = \sum_{i=1}^{k} \hat{c}_{i,j}$ instead of attribute embedding $\hat{c}$ and weights $W_{\text{tag}}^{AD}, thresh^{AD}$ instead of $W_{\text{tag}}^{RD}, thresh^{RD}$.

Require: text $d = w_1 \ldots w_n$, token embeddings $\hat{w}_1^{VE} \ldots \hat{w}_n^{VE}$, masked cell embedding $\hat{c}$

1: $(\text{tags}_1, \ldots, \text{tags}_n) \leftarrow (O, \ldots, O)$ ▷ Initialize tags
2: **for all** tokens $w_i \in w_1 \ldots w_n$ with embedding $\hat{w}_i$ **do**
3:      $\text{tags}_i \leftarrow \underset{\text{tag} \in \{I, O, B\}}{\text{argmax}} \begin{cases} \hat{w}_i^T \cdot W_{\text{tag}}^{RD} \cdot \hat{c} & \textit{if } \text{tag} \in \{B, I\} \\ thresh^{RD} & \textit{else} \end{cases}$
4: **end for**
5: **return** I-O-B-decode(d, tags) ▷ Return a tag for extraction

---

Algorithm 3 ② Decoder of ELEET: **Duplicate Detect**.

Require: Spans $V = \{v_A, v_B, \ldots\}$, token embeds. $\hat{w}_1^{DD} \ldots \hat{w}_n^{DD}$

1: $\hat{v}_A, \hat{v}_B, \cdots \leftarrow$ span_embeddings[29]$(v_A, v_B, \ldots; \hat{w}_1^{DD} \ldots \hat{w}_n^{DD})$
2: similarities $= [\hat{v}_A, \hat{v}_B, \ldots]^T \cdot W^{DD} \cdot [\hat{v}_A, \hat{v}_B, \ldots] - thresh^{DD}$
3: ▷ We use agglomerative clustering with a distance threshold to find groups (clusters) of semantically equivalent values.
4: $\bar{V} \leftarrow$ Agglom.cluster(spans=V, dist=-similarites, dist_thresh=0)
5:          ▷ Return longest span per cluster.
6: **return** $\{\text{argmax}_{v \in \text{cluster}} |v| \mid \text{cluster} \in \bar{V}\}$

---

the diagnosis of Bob from the text, even when other patients are mentioned in the text. To do that, we use the *row-detect (RD)* head that is pre-trained to extract only values for the particular entity mentioned in the input of the model (e.g., Bob) as given by the tuple from the table. For extracting the output of a union, we want to use example values and use *attribute-detect (AD)* as we discuss below.

Both the RD and the AD head extract values from the text to fill in values for latent attributes (e.g., diagnosis) and thus use the embeddings for extracting values $\hat{w}^{VE}$ (ⓑ). The RD head pairs the embedding of each text token $\hat{w}^{VE}$ (orange in Figure 3) with the embedding of a masked cell $\hat{c}$ of a latent attribute (purple) and classifies whether the token is part of a value for the cell. It consists of matrices W_I^{RD}, W_B^{RD} and a learned threshold $thresh^{RD}$ to perform the classification according to Algorithm 2. We employ so-called *I-O-B (Inside-Outside-Beginning) classification* to extract potentially multiple tokens for each attribute. With I-O-B tags, the first text token for a value is marked with a B-tag, and subsequent tokens belonging to the same value receive an I-tag. Otherwise, tokens are marked with an O-tag. The AD head works identically as the RD head but first pools all cell embeddings $\hat{c}_{i,j}$ of the same latent attribute into an attribute embedding $\hat{a}_j = \frac{1}{k} \sum_{i=1}^{k} \hat{c}_{i,j}$. Then, it classifies based on this embedding, learned weights W_I^{AD}, W_B^{AD} and learned threshold $thresh^{AD}$ which tokens are a value for that latent attribute, independent of the entity.

Finally, to avoid extracting the same value twice (e.g., to avoid duplicate rows in a *multi-row* latent table, as we explain later), it is necessary to check whether multiple extracted spans (i.e., all tokens extracted for a latent attribute) refer to the same concept. For this, we compute span embeddings $\hat{v}$ according to Lee et al. [29] from the text token embeddings coming from the final transformer layer for deduplication $\hat{w}^{DD}$ (ⓒ) and then classify which spans are duplicates using the *duplicate-detect (DD)* head. The DD head consists of a learned matrix W^{DD} and a learned threshold $thresh^{DD}$ and considers two spans v_A and v_B as duplicates if the learned similarity $\hat{v}_A^T \cdot W^{DD} \cdot \hat{v}_B$ is larger than $thresh_{DD}$. The procedure to find duplicates is shown in Algorithm 3.

After the values for all latent cells have been extracted and deduplicated, they can be inserted into the latent table to produce the materialized output. However, in the general case of *multi-row* latent tables, where each text can contribute multiple tuples to the latent table, only value extraction is insufficient. We explain the algorithm to support *multi-row* latent tables in Section 5.

4.2 Pre-training

The functionality of each of the three decoder heads represents a skill of the ELEET model that is necessary to perform MMOps as specified in the algebra in Section 3. These skills are independent of concrete data sets and thus should be taught to the model during pre-training. For pre-training the ELEET model, we pair the encoder with our decoder heads and train them end-to-end. However, we do not start pre-training from scratch but start with the pre-trained weights of TaBERT [71] for the transformer and vertical self-attention layers. These are the model components that also exist in TaBERT. For the decoder layers, which do not have a counterpart in TaBERT, we start with randomly initialized weights. During TaBERT's pre-training, the model sees pairs of texts and tables. Hence, the resulting weights are a good starting point for ELEET. We show this and the additional importance of our pre-training in our ablation study in Section 6.5.

Skill 1: Align latent attributes to text (AD head). To support that our model learns to detect all values for certain attributes, we introduce the *Attribute-Text-Alignment (ATA)* task, which aligns table attributes to text. In this task, we pair texts with tables and use the AD head to detect segments of text that are a potential value for each attribute. We use the labels of our pre-training corpus to compute a cross-entropy classification loss $\mathcal{L}_{ATA}$.

Skill 2: Extract values for table rows (RD head). For joins, as explained before, it is important to extract only values belonging to a given table row (e.g., only extract values for a given patient). To let our model learn which values correspond to which table row (e.g., in texts about multiple entities), we introduce the *Masked Cell Reconstruction (MCR)* pre-training task. In this task, we pair table rows and texts, mask out random cell values of table rows (that occur in both text and table row), and use the RD head to reconstruct the masked values from the text. To do this, the RD head leverages signals from neighboring cells, including those from the same row. MCR thus teaches the model to extract values for a certain row only (e.g., only the diagnoses of Bob), as required for the RD head. For this pre-training objective, we use the labels of our pre-training corpus to compute a cross-entropy classification loss $\mathcal{L}_{MCR}$ for classifying the I-O-B tags in Algorithm 2.

Skill 3: Detect duplicates (DD head). Finally, we introduce the *Duplicate-Detection (DD)* pre-training task to let the model learn to detect whether two spans refer to the same concept. For each attribute, *DD* takes pairs of text spans as input and is trained to predict whether they are the same. We train the DD head by classifying whether pairs of spans are duplicates using the labels of our pre-training corpus to compute a cross-entropy loss $\mathcal{L}_{DD}$.

Combined Pre-Training. We use a combined pre-training to add up all the losses of all objectives and train the entire model (including the encoder) using this multi-task loss. This has also shown benefits in other transformer-based models [11, 15] where multiple pre-train objectives are used. To balance the losses, we use a weighted sum. We choose $\alpha = 300$, $\beta = 80$, $\gamma = \delta = 1$ by examining the performance on a development set.

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{MCR} + \beta \cdot \mathcal{L}_{ATA} + \gamma \cdot \mathcal{L}_{DD} + \delta \cdot \mathcal{L}_{MLM}$$

Moreover, for the pre-training, we realized that our model benefits from adding the original Masked Language Model loss $\mathcal{L}_{MLM}$ of BERT [15]. We thus added $\mathcal{L}_{MLM}$ to the combined loss. Finally, to ensure that the model also utilizes signal from the table values during pre-training and not only the schema information (i.e., attribute names) of the table, we randomly mask out attribute names from the linearized input to our model in 15% of the cases (which is a fraction we empirically validated to provide the best performance). The complete procedure is shown in Algorithm 4.

A New Pre-Training Corpus To pre-train our model using the above procedure, we created a new open-domain pre-training corpus with Wikipedia abstracts as texts, tables constructed from Wikidata, and labels from T-REx [16]. We describe the corpus in Section 5.5.

5 MULTI-MODAL OPERATIONS

In the previous section, we introduced the ELEET-model by demonstrating how it can be used for multi-modal joins for *single-row* latent tables. In this section, we discuss how the details of all operators of the ELEET algebra can be realized using the ELEET-model.

5.1 Multi-Modal Scans

The scan is the most important operator that turns a latent table into a normal one by extracting the values for all latent attributes from each text. For implementing the scan of a *single-row* latent table, we feed in each document together with the latent attributes into our encoder: the input sequences have the form $d \| [SEP] \| \text{linearized}(LA)$, where we use MASK tokens for linearizing the latent attributes as before.

Hence, unlike the join and union explained in Section 4, we cannot access any tabular context. Therefore, we set $k = 1$ for scans, disabling vertical signal flow between input rows via vertical self-attention and mean pooling across rows to obtain attribute embeddings. Hence, $\hat{a}_j$ is simply set to the masked cell embedding $\hat{c}_{1,j}$. After feeding the sequences in the encoder, we use the attribute-detect (AD) head to extract a value for every latent attribute from each text using Algorithm 2 with matrices W_I^{AD}, W_B^{AD} and threshold $thresh^{AD}$ for I-O-B tagging.

Algorithm 4 Compute pre-training loss value for a single sample of k tuple-text pairs. In practice, multiple samples are fed into the model in a single batch, and the implementation avoids for-loops in favor of efficient matrix multiplications.

Require: texts $d_1 \ldots d_k$, table tuples $t_1 \ldots t_k$, labels Y^{RD}, Y^{AD}, Y^{DD}, loss weights $\alpha, \beta, \gamma, \delta$, MLM-loss $\mathcal{L}_{MLM}$

1: **Feed tuple-text pairs into Encoder to get all embeddings**

$$\begin{matrix} \hat{w}_{1,1}^{VE} & \hat{w}_{1,1}^{DD} & \hat{c}_{1,1} \\ \ddots & , \ddots & , \ddots \\ \hat{w}_{k,n}^{VE} & \hat{w}_{k,n}^{DD} & \hat{c}_{k,m} \end{matrix} \leftarrow \text{Encoder} \begin{pmatrix} d_1 \| [SEP] \| \text{linearized}(t_1), \\ \cdots \\ d_k \| [SEP] \| \text{linearized}(t_k) \end{pmatrix}$$

2: $\mathcal{L}_{MCR} \leftarrow \mathcal{L}_{CTA} \leftarrow \mathcal{L}_{DD} \leftarrow 0$
3: **for all** $i \in 1 \ldots k$, masked cell $c \in t_i$, token $w \in d_i$ **do**
4: $\quad l_I \leftarrow \hat{w}^T \cdot W_I^{RD} \cdot \hat{c}$ $\qquad\qquad\qquad$ ▷ Compute $\mathcal{L}_{MCR}$
5: $\quad l_B \leftarrow \hat{w}^T \cdot W_B^{RD} \cdot \hat{c}$
6: $\quad l_O \leftarrow thresh_{RD}$
7: $\quad \mathcal{L}_{MCR} \leftarrow \mathcal{L}_{MCR} - \sum\limits_{x \in \{I,O,B\}} \mathbf{1}_{Y_{i,w,c}^{RD}=x} \cdot \log \frac{\exp l_x}{\sum_{x' \in \{I,O,B\}} \exp l_{x'}}$
8: **end for**
9: **for all** $i \in 1 \ldots k$, $j \in 1 \ldots m$, token $w \in d_i$ **do**
10: $\quad \hat{a} \leftarrow \frac{1}{k} \sum_{i'=1}^{k} \hat{c}_{i',j}$ $\qquad\qquad$ ▷ Compute column embedding
11: $\quad l_I \leftarrow \hat{w}^T \cdot W_I^{AD} \cdot \hat{a}$ $\qquad\qquad\qquad$ ▷ Compute $\mathcal{L}_{CTA}$
12: $\quad l_B \leftarrow \hat{w}^T \cdot W_B^{AD} \cdot \hat{a}$
13: $\quad l_O \leftarrow thresh_{AD}$
14: $\quad \mathcal{L}_{CTA} \leftarrow \mathcal{L}_{CTA} - \sum\limits_{x \in \{I,O,B\}} \mathbf{1}_{Y_{i,j,w}^{AD}=x} \cdot \log \frac{\exp l_x}{\sum_{x' \in \{I,O,B\}} \exp l_{x'}}$
15: **end for**
16: **for all** $i \in 1 \ldots k$, values $v_A, v_B \in d_i$ (given by Y^{AD}) **do**
17: $\quad \hat{v}_A, \hat{v}_B, \cdots \leftarrow \text{span\_embed.}[29](v_A, v_B, \ldots; \hat{w}_{i,1}^{DD} \ldots \hat{w}_{i,n}^{DD})$
18: $\quad l_Y \leftarrow \hat{v}_A^T \cdot W^{DD} \cdot \hat{v}_B$ $\qquad\qquad\qquad$ ▷ Compute $\mathcal{L}_{DD}$
19: $\quad l_N \leftarrow thresh_{DD}$
20: $\quad \mathcal{L}_{DD} \leftarrow \mathcal{L}_{DD} - \sum_{x \in \{Y,N\}} \mathbf{1}_{Y_{v_A,v_B}^{DD}=x} \cdot \log \frac{\exp l_x}{\sum_{x' \in \{Y,N\}} \exp l_{x'}}$
21: **end for**
22: $\mathcal{L}_{MCR} \leftarrow \frac{1}{N_{MCR}} \cdot \mathcal{L}_{MCR}$ $\qquad$ ▷ Compute mean of loss values
23: $\mathcal{L}_{CTA} \leftarrow \frac{1}{N_{CTA}} \cdot \mathcal{L}_{CTA}$
24: $\mathcal{L}_{DD} \leftarrow \frac{1}{N_{DD}} \cdot \mathcal{L}_{DD}$
25: $\mathcal{L} \leftarrow \alpha \cdot \mathcal{L}_{MCR} + \beta \cdot \mathcal{L}_{CTA} + \gamma \cdot \mathcal{L}_{DD} + \delta \cdot \mathcal{L}_{MLM}$
26: **return** $\mathcal{L}$ $\qquad\qquad$ ▷ Train using AdamW [38] optimizer

---

Next, we explain the scan on a *multi-row* latent table. In a *multi-row* latent table, a single text $d \in D$ may correspond to multiple tuples $t_{d,1}, t_{d,2}, \ldots$. For instance, a patient report can document multiple diagnoses of a patient, resulting in multiple output tuples of a scan. As described before, the user needs to define a *document-level key* la_{key} for the *multi-row* latent table, which is required to distinguish between the different tuples coming from the same text. For the patient reports in the example before, we assume the attribute la^{key} =diagnosis serves as the *document-level key*, meaning that all rows coming from the same text should represent a different diagnosis. For realizing a multi-modal scan on such a table, we use an Algorithm 5 which is composed of two phases.

Algorithm 5 Multi-row multi-modal scan for a single document.

Require: $d = (w_1, \ldots, w_n)$, $LA = (la_{key}, la_1, \ldots, la_m)$

1: **1. Get values** for la^{key}
2: $\hat{w}_1 \ldots \hat{w}_n, \hat{c}_{la_{key}}, \hat{c}_1, \ldots, \hat{c}_m \leftarrow \text{Encoder}(d \,\|\, [SEP] \,\|\, \text{lin.}(LA))$
3: $V^{key} \leftarrow \text{attribute} - \text{detect}((\hat{w}_1, \ldots, \hat{w}_n), \hat{c}_{la_{key}})$
4: $\{v_1^{key}, \ldots, v_l^{key}\} \leftarrow \text{duplicate} - \text{detect}(V^{key})$
5: **2. Get values for** $la_1, \ldots, la_m$
6: $\hat{W}, \hat{C} \leftarrow \text{Encoder}(d \,\|\, [SEP] \,\|\, \text{lin.}(v_1^{key}) \,\|\, \text{lin.}(la_1, \ldots, la_m), \ldots$
7: $\qquad d \,\|\, [SEP] \,\|\, \text{lin.}(v_l^{key}) \,\|\, \text{lin.}(la_1, \ldots, la_m))$
8: **for** $i = 1$ to l **do**
9: $\quad V_{i,1}, \ldots, V_{i,m} \leftarrow \text{row} - \text{detect}((\hat{w}_{i,1}, \ldots, \hat{w}_{i,n}), \hat{c}_{i,1}), \ldots$
10: $\qquad \text{row} - \text{detect}((\hat{w}_{i,1}, \ldots, \hat{w}_{i,n}), \hat{c}_{i,m})$
11: $\quad \{v_{i,1}\}, \ldots, \{v_{i,m}\} \leftarrow \text{duplicate} - \text{detect}(V_{i,1}), \ldots$
12: $\qquad \text{duplicate} - \text{detect}(V_{i,m})$
13: **end for**
14: **return** $\begin{bmatrix} v_1^{key} & v_{1,1} & \ldots & v_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ v_l^{key} & v_{l,1} & \ldots & v_{l,m} \end{bmatrix}$
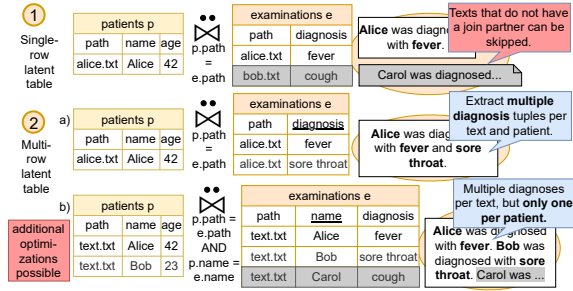

Figure 4: Besides the join with a *single-row* latent table ①, there are two cases for *multi-row* latent tables ②. In the first case, multiple tuples need to be extracted per table row of the tabular operand (a). An interesting special case is when for each table tuple, only a single tuple needs to be extracted (b). Joins allow for several automatic optimizations, depending on the case. For example, texts not having a join partner can be skipped during extraction, which is particularly beneficial when the table has been filtered before the join. Moreover, in the case of (b), there is no need to run Algorithm 5.

In the first phase of the procedure, the model extracts all values for the *document-level key* la_{key}, using our AD head in the decoder (to extract all diagnoses; see lines 2+3 of Algorithm 5). The number of values extracted in this step determines the number of output rows for a given text d. Moreover, we use the duplicate-detect head to avoid generating duplicate rows when the text mentions values for la_{key} twice (e.g., when the same text mentions both *fever* and *high body temperature*; see line 4 of Algorithm 5).

In the second phase, the extraction process is conducted on the remaining attributes that depend on the *document-level key*, denoted as $la_1, la_2, \ldots$. To accomplish this, the MASK token of the *document-level key* la_{key} is now replaced with the values extracted in the first phase (line 6). For the remaining attributes (which still have a MASK in the input), we extract the values using the ELEET model. Here, we use the row-detect (RD) head to extract only values corresponding to those extracted in the first phase (line 9).

To process texts longer than our model's context window of 512 tokens, we use a sliding window to process the whole text. The first phase is executed independently using the sliding windows to extract all keys. Afterwards, we collect and deduplicate the extracted values for the *document-level key* across windows. Finally, the second phase can process each window independently with the extracted keys. Afterwards, all values are collected across windows to generate the output tuples. In an experiment in Section 6.8, we show that this approach works well even for longer texts.

5.2 Multi-Modal Joins and Unions

As introduced in Section 3, multimodal joins ($T \bowtie D.LT$) and unions ($T \uplus D.LT$) can replace the multi-modal scan followed by a traditional join ($T \bowtie \ddot{S}can(D.LT)$) or union ($T \cup \ddot{S}can(D.LT)$). As such, they also extract values for latent attributes from text but can leverage additional context given by their tabular operand for better and faster extractions. For *single-row* latent tables, joins and unions are implemented as explained in Section 4, feeding sequences as shown in Figure 3 ① into the model and then using the RD head for joins, and the AD head for unions for value extraction.

Joins and unions for *multi-row* latent tables are implemented using Algorithm 5. The only difference is the additional context fed into the encoder (see lines 2 and 6 in Algorithm 5). For joins, we feed additional table values added to each input sequence, coming from a tuple linked to the text of the sequence (as shown in Figure 3 ①). For unions, we again randomly sample two rows from the tabular operand and feed these as example values the ELEET-model, which can be used by vertical self-attention to improve extractions.

The join comes with opportunities for optimization as shown in Figure 4. In the case ①, not all texts in the text collection may have a join partner in the table, especially if the table has been filtered beforehand (e.g., only patients with age < 18 are selected, and thus, only texts of such patients need to be processed). As such, the path in the table data acts as an index to the text collection to decide which subset of text documents need to be scanned (i.e., we can avoid scanning all text documents). For *multi-row* joins, there are two cases. In Figure 4 ② (a), we need to extract multiple diagnoses per text document for each tuple (i.e., patient) from the normal table. Here, we must run Algorithm 5 to first extract all diagnoses from the text. Afterwards, the values for potential dependent columns (e.g., treatment) are extracted. However, case ② (b) can be optimized thanks to the row-detect head, as shown below. In this case, each text is about multiple patients and we have only one diagnosis per patient. As such, multiple tuples from the table refer to the same text and we only need to extract a single row per patient tuple. For this, we feed each text multiple times into the encoder, each time paired with a different patient tuple from the table. The RD head is pre-trained to extract only values corresponding to the table tuple
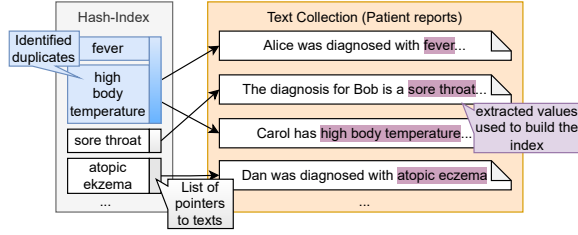
Figure 5: The multi-modal index used for selections. When extracting the values stored in the index from the texts (using attribute-detect), we use the duplicate-detect head to identify values that refer to the same concept. This allows the index to return texts of patients that have a fever when users query for patients with high body temperature.

(i.e. patient) it is paired with, which allows us to extract multiple rows from a single text without Algorithm 5. The difference to case (a) is that the tabular join partner already contains the values for the *document-level key* la_{key} = *name*. Hence, we can skip extracting these in the first phase of Algorithm 5.

5.3 Multi-Modal Selection

The multi-modal selection $\ddot{\sigma}_{cond}$ filters data based on attributes extracted from text. For example, users might be interested only in *treatments* for patients diagnosed with *fever*. As discussed in Section 3, the multi-modal selection comes in two variants.

Variant 1. The first variant $\sigma_{cond}(D.LT)$ takes a latent table as input and produces a (filtered) latent table as output (i.e., without materializing neither input nor output table). Important is that the selection thus reduces the number of text documents in a collection before using them as input (e.g., to a join). Moreover, the selection uses semantic similarity for filtering. In ELEET, we provide an implementation for this scan that uses a multi-modal index on the selection column. The core idea of a multi-modal index is that it maps a search key for the query attribute (e.g., diagnosis) to text documents that contain the search key. To construct a multi-modal index, our approach leverages the attribute-detect head to extract all values for the search key from all texts. For building the index, we put the extracted values and the pointers to the text documents into a traditional index to be able to retrieve text documents quickly. ELEET currently uses a hash index. However, unlike traditional hash indexes, we group similar extracted values into one index entry using the duplicate-detect head of our ELEET-model, as depicted in Figure 5. This allows the index to return text documents containing the diagnosis of *high temperature*, even if the user queries for *fever*.

Variant 2. As discussed in Section 3, ELEET comes with a second variant $\ddot{\sigma}_{cond}(T)$ that can be evaluated on a normal table. Here, we might still want to select rows based on the semantic similarity of values extracted from the text (e.g., if the selection is executed after a multi-modal join). Given a selection condition (e.g., diagnosis = fever), we embed it by feeding `<attribute> is <value>` (e.g., diagnosis is fever) into our encoder and compute a span-embedding of `<value>` as discussed before. Then, we use the DD head of our model to decide whether a value embedding from the table and the selection value refer to the same semantic concept. To support this

selection, an important optimization is to keep the span embeddings of extractions created during a scan, join or union and attach it to the values of the respective output table. This avoids recomputing these embeddings if a selection follows one of those operators.

5.4 Multi-Modal Aggregation

The last operation supported in ELEET is a multi-modal aggregation $\ddot{\chi}_{F,A'}(T)$ that can be used for group-by aggregation based on attributes extracted from text. Like the second selection variant, this operation is designed to work on tables T created from texts (e.g., to aggregate the result of the output of a multi-modal join). Unlike a normal aggregation, the operation uses the semantic similarity of group-by values to form groups. For example, assume we only use the patient reports as input and want to count how often a certain diagnosis was named across all texts. The solution is first to extract all diagnoses using a scan and then use a multi-model aggregation to elegantly deal with cases where the same diagnosis is expressed differently in different texts (e.g., as fever and high body temperature) but is still counted as the same diagnosis. More specifically, for the multi-modal aggregation, we compute a similarity matrix of the group-by values (as in line 2 in Algorithm 3). Afterwards, we use agglomerative clustering with a distance threshold to find all clusters (i.e., groups) for group-by (as in line 4 in Algorithm 3). Note that computing the distance matrix is quadratic in the number of input values, which becomes expensive for very high numbers of table rows. Hence, in the future, we aim to replace the clustering-based aggregation algorithm with latent-semantic-hashing-based DBSCAN [70], which is linear in the number of input embeddings.

5.5 A New Pre-Training Corpus

Unfortunately, currently no pre-training corpus exists that allows us to pre-train our ELEET-model as outlined in Section 4. In contrast to existing corpora such as [5, 24, 73], we need a different parallel corpus where the texts contain the same information (e.g., same entities) as the tables, and where we know which cell values also occur in a text, allowing us to mask them for pre-training. Hence, we constructed a new parallel open-domain pre-training corpus from Wikidata and Wikipedia for pre-training multi-modal database models. We open-source the corpus together with our code[1].

The main idea of our data set is that we make use of T-REx [16], a large-scale alignment of Wikidata triples and Wikipedia abstracts. The T-REx data set contains 11 million alignments of Wikidata triples to Wikipedia abstracts. All the 3.09 million abstracts occurring in T-REx are also part of our data set. T-REx itself is created automatically using the distant supervision assumption for computing the alignment and can thus be noisy sometimes, but it allows us to construct a large pre-training corpus with objectives aligned to the downstream task of multi-modal database operations. It has been used by other researchers for training their ML models [45].

Hence, we use the alignment of T-REx as a starting point to construct our parallel corpus of texts and relational tables. T-REx contains information about the location of mentioned entities in the texts, which we can use as labels for our pre-training objectives. As texts, we simply use the aligned Wikipedia abstracts and construct additional tables using Wikidata, by grouping similar entities

---

[1]https://github.com/DataManagementLab/eleet

Figure 6: Example pre-training sample consisting of three rows and texts from our pre-training data set.

together in a table and using Wikidata properties as columns. We use several techniques to obtain a rich and diverse data set, e.g. we randomize column names using Wikidata's aliases or concatenate multiple abstracts to simulate texts about multiple entities. See Figure 6 for an example training sample for pre-training. In total, our pre-training data set consists of 8.2m such samples in its training set and 7.8k in its development set.

6 EXPERIMENTAL EVALUATION

In this section, we present the results of our experimental evaluation, which justifies the design of ELEET. To do so, we constructed a challenging benchmark containing 70 multi-modal query plans over four data sets that we publish along with this paper[1].

6.1 Evaluation Setup

Data sets. Our benchmark consists of four data sets. Based on each data set, we build a database consisting of 1-6 relational tables and 1-3 text collections; see Table 3 (upper part) for detailed statistics. Note that all data sets are from different domains and include data that ELEET has not seen during pre-training.

(1) *Rotowire*: The *rotowire* data set [68] contains a text collection of 4850 basketball game reports with an emphasis on game statistics. Hence, the values to be extracted are primarily numeric. We complement the text collection with several tables of Basketball Players and Teams to enable multi-modal queries. In total, this data set has 6 structured tables and 1 text collection, while 2 latent tables are derived from the text collection.

(2) *T-REx (unseen):* The second multi-modal database is built using T-REx [16]. Importantly, this data set is composed of three unseen domains that were not used in our pre-training: nobel prize winners, skyscrapers, and countries. The data set uses Wikipedia abstracts as text collections and table rows are constructed from Wikidata. In total, this data set has 6 structured tables and 3 text collections, while 6 latent tables are derived from the text collections. Importantly, T-REx is constructed automatically and is thus too noisy to be used for evaluation. Hence, we fine-tune the models on the three mentioned domains but evaluate only on queries from the nobel domain, which is the least noisy.

(3) *Aviation*: Based on the *aviation* data set from [23], we construct a document collection, where each document is an aviation accident report published by the United States National Transportation Safety Board. Attributes that can be extracted from the texts are the location of the accident, the severity of the damage, and so on. In total, this data set has 3 structured tables and 1 text collection, while 1 latent table is derived from the text collection.

Table 3: Statistics of our benchmark. The lower part indicates how often each multi-modal operator is used in queries.

Data set	Rotowire	T-REx	Aviation	Corona
#text collections	1	3	1	1
#tables	6	6	3	1
#latent tables	2	6	1	1
#queries	28	12	15	15
#attributes latent tables	21, 14	8	7	7
#texts (test set)	728	221	30	30
#join	10	2	5	0
#union	4	6	5	15
#scan	14	4	5	0
#selection	6	2	3	0
#aggregation	6	2	3	0

(4) *Corona*: The final data set is again based on data used in [23]. It consists of one text collection containing the daily status reports by the German RKI. From these texts, information like the number of persons infected by or recovered from Covid-19 can be extracted. We pair these texts with a single table (used for multi-modal unions only). In total, this data set has 1 structured table and 1 text collection, while 1 latent table for the multi-modal union with the same attributes can be derived from the text.

Query Generation. Based on these data sets, we generate 70 query plans with 1-3 multi-modal operators and 0-1 traditional operators. See Table 3 (lower part) for statistics of the generated queries. A list of all queries is available in appendix A.

Baselines. We compare the ELEET to several strong LLM and SLM baselines. Since there are no other systems so far that do joins and unions on multi-modal table/text data, we build our baselines from recent state-of-the-art models from the NLP community.

(1) *Text-To-Table [69]*: Text-To-Table is a machine learning model based on BART (-base) [31] with a similar size as our model that can be trained to translate texts to tables. Different from the ELEET-model, it did not receive any special pre-training (i.e., it needs to be trained with the BART-weights as starting point for each new data set). Moreover, it uses an autoregressive decoder, which is less efficient than our lightweight decoder.

(2) *LLaMA-2 (7B) [59]*: LLMs such as LLaMA can also be used to translate texts to tables using few-shot prompting (i.e., in-context learning, ic) [4, 67] or fine-tuning (ft). We evaluated several prompts and found the following prompt to work well: "Transform the input text into a <name of latent table> table. ↩ Only output the table in
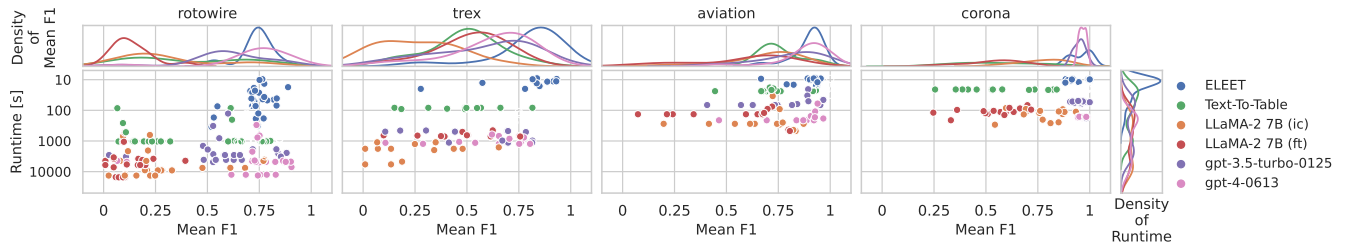
Figure 7: We plot the runtime and F1 scores of each approach on all 70 queries in our benchmark in a scatter plot. Due to the vastly different runtimes of individual queries, the runtime is shown in log scale. The GPT-models and LLaMA (ic) use few-shot prompting (in-context learning) and the other models are fine-tuned. We see that ELEET is always the fastest approach, while the baselines are sometimes several orders of magnitude slower. The density plot above the scatter plot makes it easy to see that ELEET is among the most accurate approaches for all data sets despite being a small model.

CSV-Format without explanations. If there is no information for a cell, leave it empty. The header row is: <columns of the result table>. ↪ Input: <text to translate to a table>" For in-context learning, we add as many task demonstrations as the context window of 4096 tokens allows. For fine-tuning, we use Q-LoRA [14].

(3) *GPT-3.5 (gpt 3.5 turbo) [4] and GPT-4 (gpt-4-0613) [41]*: GPT-3.5 and GPT-4 are even larger LLMs with 175 billion parameters and 1.76 trillion parameters respectively. We use few-shot prompting for both models, using the same prompts as for LLaMA. GPT-4 is used by Evaporate-Direct [3] to extract information from semi-structured documents (e.g., XML-Documents) similar to our baseline. Unfortunately, all OpenAI models (GPT-3.5 and GPT-4) are closed source and thus can only be accessed via the API and not deployed on our hardware for comparing runtimes. Nonetheless, we compare in Experiment 1 against these models to get a basic understanding of their accuracy. However, since these models use many orders of magnitude more parameters than ELEET and thus are naturally much slower, we skip these baselines in Experiments 2 and 3 and concentrate on the smaller language models such as LLaMA-2 and investigate how fine-tuning for those smaller models helps to achieve better accuracies compared to ELEET.

We do not compare against WannaDB [23], because it requires user interaction for information extraction. Moreover, we do not compare against Evaporate-Code [3], because it is designed to extract information from semi-structured documents (e.g., XML, JSON) and not continuous text. To run MMOps with the baselines, we use their capability to transform texts to tables to transform all documents in the document collection to tables. Afterwards, we perform the corresponding traditional database operation.

Training and Fine-tuning. All experiments were executed on a DGX A100 server. For pre-training, we used 4 GPUs, which took approximately 8 days to train our model for 6 epochs on our pre-training data set. For fine-tuning and inference — in particular, for the performance measurements — we used 2 GPUs only (for all models except GPT-3/4, which we cannot run on our hardware).

Metrics. In our experiments, we focus on two main dimensions: (1) First, we measure the quality of the query results computed by ELEET compared to the baselines. We use Exact Match (EM) to evaluate the quality of query results. All datasets come with a ground truth translation for all texts in the dataset. A value is

considered correct if it exactly matches its label. We compute an F1 score for each text and report their mean. While extractive models will generally output the values exactly how they are mentioned in the text (e.g., US president), generative models might output the values in arbitrary form (e.g., President of the United States). Hence, the datasets come with different alternatives for each value, each of which counts as correct. For aggregations, we group-by a certain attribute and collect for all other attributes the values for all groups. We then compute an F1 score per group and report the mean. (2) To compare the efficiency, we compare the runtime for each query.

6.2 Exp. 1: Runtime vs. Accuracy

Our main goal is to show that ELEET is orders of magnitude more runtime efficient than the SLM and LLM baselines while being more accurate. We therefore execute the full set of queries on all data sets using ELEET and all baselines. Since the collection of training data per text collection is expensive, we focus on a scenario where only limited data is available for fine-tuning. Hence, in this experiment, we limit the number to 64 labeled texts for fine-tuning per data set (for ELEET, Text-To-Table, LLaMA). For LLaMA, GPT-3, and GPT-4, we use few-shot prompting and include the annotated examples in the input prompt, as explained before. We think this is a reasonable number of texts that can be labeled manually. We later evaluate in more detail how a different amount of labeled training data for fine-tuning affects the accuracy of ELEET compared to the baselines. **Overall Results.** In Figure 7 (lower part) we plot runtime and F1 scores for each query in our benchmark in a scatter plot. Overall, we see that ELEET provides high accuracy (i.e., a high F1-score) while being fast in execution (i.e., often in the order of seconds). **Runtime.** As Figure 7 shows, ELEET is the fastest approach, while other approaches are up to 575× slower. For instance, the following query which is included in our testing set (i.e., $player\_info \bowtie player\_to\_reports \bowtie reports.player$) takes about three minutes with ELEET, 17 minutes using Text-To-Table, 2.5 hours using LLaMA, 1.5 hours using GPT-3, and almost 4 hours using GPT-4. This shows how expensive in terms of runtime LLMs are when applied to many texts. Overall, ELEET clearly outperforms the LLM-based baselines (LLaMA, GPT-3 and GPT-4), even though the GPT-models run on the hardware of OpenAI. But even the comparatively small Text-To-Table is significantly slower than ELEET, which we attribute to

its use of a transformer-based autoregressive decoder that requires many passes through the model compared to our model.

Accuracy. The density of F1 values (upper part of Figure 7) nicely shows that ELEET usually exceeds the performance of the baselines despite being a small model. For aviation and corona, ELEET achieves F1-scores of over 90% for most queries. For the other two data sets, *rotowire* and *T-REx*, which are more challenging, the F1 scores of ELEET cluster above and around 75% and 80%, outperforming the baselines. The most competitive model is GPT-4, which is very accurate on aviation and corona. However, due to its immense size, it is much computationally more expensive as discussed before. Moreover, we see the effect of our pre-training when comparing the performance to Text-To-Table and fine-tuned LLaMA, which have F1 scores of around 25% for most queries on *rotowire*. 64 samples are not enough for these models to allow for decent extractions.

6.3 Exp. 2: Varying Data Sizes for Fine-tuning

In the previous experiment, we have seen that ELEET can accurately compute multi-modal queries with only a few fine-tuning samples. However, users might often have different requirements on the quality of query results and access to different amounts of labeled texts for training. In this experiment, we show how ELEET behaves with training sets of various sizes compared to the baselines. Here, we use five queries on the *rotowire* data set for testing. The queries used in this experiment cover all different MMOps:

- **Scan:** $\ddot{S}can(reports.player)$
- **Join:** $(player\_info \bowtie player\_to\_reports) \ddot{\bowtie} reports.player$
- **Union:** $player\_stats \ddot{\cup} reports.player$
- **Selection:** $\ddot{S}can(\ddot{\sigma}_{Points=28}(reports.player))$
- **Aggregation:** $\ddot{\chi}_{name}(\ddot{S}can(reports.player))$

We fine-tune several models for ELEET, Text-To-Table, and LLaMA using a varying number of labeled texts. We vary between 4 labeled texts up to the full training set (3398 texts) and report the mean F1 score of all queries from above for each model. As discussed before, we limit ourselves to models that run on our hardware.

Accuracy with varying training data. Figure 8 shows the results. With limited training data (4-16), only LLaMA using in-context-learning and ELEET achieve accuracies of around 40%. The other fine-tuned methods LLaMA and Text-To-Table struggle in these few-shot scenarios due to the missing specialized pre-training. When we increase the amount of training data for fine-tuning, the accuracies of all fine-tuned methods increase. LLaMA using in-context learning, on the other hand, cannot use this additional training data due to the limited context size. Overall, we see that ELEET outperforms both fine-tuned baselines across all training set sizes. In particular, for scans, joins, and unions, ELEET achieves a better Mean F1 score than Text-To-Table and LLaMA even when trained on the entire data set of *rotowire*. We achieve an unmatched F1 score of 87% for joins, unions, and scans when trained on the entire data set.

6.4 Exp. 3: Runtime of Multi-modal Operators

In the next experiment, we zoom into query runtime and show what typical query runtimes for ELEET are, and how efficient each individual operator is. Figure 9 shows the results.

Overall results. As we have seen before, ELEET vastly outperforms all baselines in terms of query runtime. However, comparing

Figure 8: Mean F1 scores for different queries and training set sizes on rotowire. The F1 scores for all models that use fine-tuning increase as the number of labeled texts for training increases, but ELEET consistently outperforms all baselines.

Figure 9: Runtime comparisons on the two latent tables of rotowire (`reports.player` and `reports.team`, containing Player and Team statistics). Comparing the query runtime of ELEET on the different queries, we see that joins are faster than scans or unions as they do not run Algorithm 5.

the query runtime of different queries using ELEET, we see big differences. In particular, joins are more efficient than unions or scans, because they do not necessitate execution of Algorithm 5 as discussed in Section 5.2. The multi-modal union and scan will extract multiple tuples per game report by first extracting the name of the Team/Player (name is the *document-level key* of the latent `reports.player` and `reports.team` tables). Only in a second iteration are the statistics of each Player or Team extracted. The join on the other hand uses the signal from the table, the document collection is joined with (`player_info` and `team_info`). These already contain names and other information of players and teams, and hence the first iteration can be skipped. This effect can be best seen in Figure 9 (right), where a join with the latent `teams` table takes 26 seconds, while scans and unions take 70 seconds. Moreover, selection operations can reduce query runtime to the order of seconds. This is due to the use of indexes, allowing efficient execution if only a few texts need to be processed based on the filter predicate. A similar effect also holds for selective join, as we show next.

6.4.1 Selective Multi-modal Join Queries. To investigate the scenarios of selective joins where the data in the tabular input helps

to reduce the number of texts we need to analyze, we look at such join queries on the latent player table of `rotowire`:

$$(\sigma_{cond}(player\_info) \bowtie player\_to\_reports) \ddot{\bowtie} reports.player$$

In these queries, *cond* is an arbitrary condition that selects a certain amount of players. The join queries in this experiment have different selectivities; i.e., the query only selects a few `players` before executing the multi-modal join with the text collection. Since tuples in the table are linked to the game reports (e.g., a tuple about a player is linked to all the game reports the player participated in), this usually results in only a few selected texts as well (i.e., the filter on the table acts as an index into the text collection).

Runtime for different selectivities. Figure 10 (left) shows how the different selectivities affect the overall query runtime of the queries containing the selective multi-modal join operator. Here we encounter another benefit of ELEET: Since ELEET only needs to materialize those rows that have a join partner in the tabular operand (by feeding table tuples in the model), the runtime is reduced to the order of a few seconds. All other approaches translate the entire text to a table, which results in runtime overhead.

6.4.2 Multi-Modal Selection. In the second scenario, we analyze the runtime of simple queries that only need to scan a subset of texts in a text collection using a multi-modal selection operation (variant 1) on extracted attributes (e.g., $\ddot{S}can(\ddot{\sigma}_{Points=28}(reports.player))$). If implemented naively, this query results in a costly operation since, independent of the selectivity, all texts need to be first transformed to tuples to judge which texts qualify for the filter condition. Instead, ELEET uses an index, as explained in Section 5.3.

Runtime for different selectivities. Figure 10 (right) shows the runtime with different filter conditions resulting in different amounts of selected texts. For queries with low selectivity, ELEET can again compute the query results in a few seconds. The naive solution to translate all texts into tables first and then doing the selection afterwards would always take a few minutes, independent of selectivity.

6.5 Exp. 4: Ablation Study for Pre-Training

In the next experiment, we show the effect of our pre-training objectives by comparing it to other existing pre-training procedures.

Alternative pre-training procedures. To show the importance of multi-modal pre-training that teaches the model the necessary skills to perform MMOps, we examine whether our pre-training procedure results in better extractions compared to other pre-training procedures. We compare against two alternative pre-training procedures: the pre-training procedure used for BERT [15] and the pre-training procedure used for TaBERT [71]. The BERT model aims at natural language understanding and is thus pre-trained on plain text only, meaning it has never seen any tabular data before. TaBERT, on the other hand, is also pre-trained on a parallel corpus of texts and tables scraped from the web. However, in the parallel corpus of TaBERT, texts and tables are often only vaguely related. Moreover, the pre-training objectives are designed for tasks like text-to-SQL and are not ideal for enabling MMOps.

Effect of different pre-training procedures. Figure 11 shows how the different pre-training procedures affect the quality of query
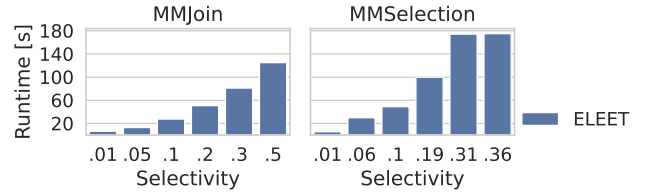

Figure 10: Runtime behavior of ELEET for selective queries on rotowire. (Left) The query selects a subset of rows (i.e. players) from the table before a multi-modal join, which results in fewer texts being processed. (Right) A multi-modal selection operation (with subsequent scan) using our secondary index. The index reduces the number of processed texts, allowing much faster runtimes for lower selectivities.
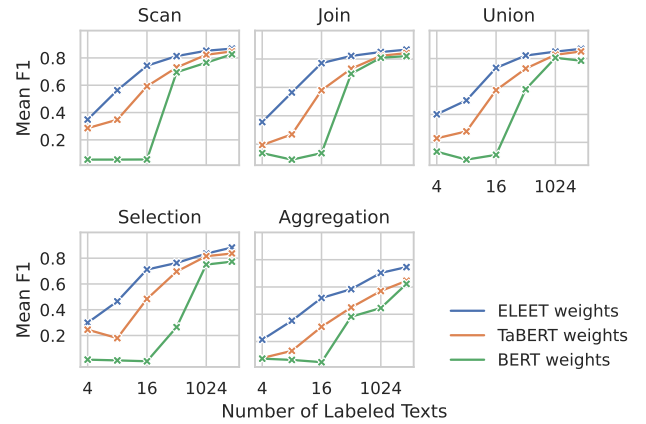

Figure 11: Comparison of the query result quality when using ELEET with different pre-trained weights. The pre-trained weights resulting from our pre-training procedure result in better extractions across training set sizes.

results when varying the number of samples for fine-tuning. Especially with only limited fine-tuning data, our model can consistently extract tabular data from text more accurately. Comparing TaBERT to BERT, we see that TaBERT can consistently outperform BERT since it is already using a form of tabular pre-training.

6.6 Exp. 5: Architectural Design Decisions

In this section we justify the design decision of our model, in particular the use of an extractive model as well as the use of vertical-self attention in our model.

Extractive vs. Generative Models. As explained in Section 2.2, we chose an extractive decoder. The alternative, a generative auto-regressive decoder generates the output tokens one-by-on and thus requires many passes through the model. This can also be seen in our previous experiments. In Figure 7, we show that ELEET is much faster than Text-to-Table [69] across all queries. Text-To-Table is an SLM with a similar number of parameters as ELEET
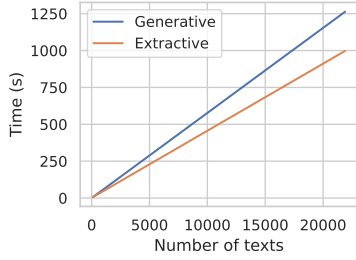
Figure 12: Comparison of extractive and generative decoding on the SQuAD dataset. The generative variant needs 25 percent more time than the extractive variant.

that uses autoregressive decoding. This shows that for the task of transforming texts to tables, extractive decoding is much faster than autoregressive decoding.

Moreover, to isolate the effect of extractive decoding versus generative decoding using an autoregressive model, we did a micro-benchmark, where we stripped down our model to a simplified version, essentially consisting only of a transformer-based encoder and either an extractive decoder or a transformer-based autoregressive decoder. Hence, this stripped-down model can be used to either extract values from text (as we do) or generate them using its autoregressive decoder (as the GPT models do). For the autoregressive model, we use 12 transformer layers for the Encoder (as discussed in Section 4) and another 12 for the decoder. To make the comparison fair, we increased the number of encoder layers of the extractive model to 24. Hence, both variants have the same number of transformer layers and are equivalent in size. For the evaluation, we used the SQuAD [47] dataset, which requires extracting a single value from each text (can be seen as a table with a single column) to avoid the effects of different output serializations for the generative model.

Figure 12 shows that the extractive variant is 25 percent faster than the generative variant, which means the absolute runtime difference scales linearly with the number of documents in the document collection. Moreover, this effect is more pronounced when the output sequences are longer, i.e., when the output is not just a single value consisting of a few tokens.

Vertical Self-Attention. As explained in Section 4, one crucial component of our model is the use of vertical self-attention [71] to let signal flow between input sequences. Especially for unions, letting signal flow between the different input sequences is important. In a union, the other rows contain example values from the tabular operand that can be used as additional context that can help during extraction. These example values can, for example, resolve potential ambiguities in the names of the query attributes.

To evaluate this effect in isolation, we created a new text collection of ambiguous patient reports that contain two kinds of diagnoses: First, the patient reports contain diagnoses regarding the patients' health problems. Additionally, technical equipment (e.g., the equipment in the room) is also broken, and the text contains diagnoses regarding their malfunctioning. We constructed the

dataset using a set of patient report templates and letting GPT-4 fill in the blank health-related and technical diagnoses.

Hence, in a latent table with columns name and diagnosis, it is unclear which diagnosis should be extracted, the health-related one or the technical equipment-related one. However, this ambiguity can be resolved when performing a union with either a health-related diagnosis table or a technical-equipment-related diagnosis table, if the model is able to let the signal flow from the example rows to the row containing the mask tokens for extraction. Table 4 shows that ELEET is able to resolve this ambiguity when using the example rows as additional context. With vertical self-attention, ELEET is able to extract the correct diagnosis almost perfectly. Without vertical self-attention, the aforementioned ambiguity cannot be resolved, causing the F1-score to drop to around 50%.

Table 4: Comparison of ELEET with and without vertical attention on a synthetic dataset with ambiguous column names. Vertical attention allows signal flow between the input rows, which helps resolve ambiguous column names.

Model	user's interest	mean F1
ELEET	technical diagnosis	0.99
ELEET	health diagnosis	0.99
ELEET (w/o vertical attention)	technical diagnosis	0.41
ELEET (w/o vertical attention)	health diagnosis	0.55

6.7 Exp. 6: Automatic Latent Table Registration

As explained in Section 3.1, registering a latent table comes with some manual overhead. More specifically, in order to register a latent table, users have to define a schema, define whether a latent table is *single-row* or *multi-row* and in the latter case, specify the *document-level key*. To reduce the manual overhead, we tried to automate the selection of the *document-level key* and the decision of whether a latent table is *multi-row* such that the user does not need to make this decision by screening texts manually. We used Chat-GPT-3.5 and prompted it to decide what the *document-level key* of a latent table should be and whether a latent table is *multi-row* (see prompt below). In the prompt, we put a small sample of three example texts and the schema of the latent table. Interestingly, we found that Chat-GPT-3.5 (gpt-3.5-turbo-0125) was able to correctly identify the *document-level key* and whether a latent table is *multi-row* for all latent tables of our four datasets as shown in Table 5. As this process needs to be done only once per latent table (and not per document) and latent table registration happens before query execution time, we believe it is suitable to use a large language model like Chat-GPT-3.5 for this task.

Details on prompt. We used the following prompt for the identifying attribute:

```
   I want to transform the information stored in texts
into a table. The table should have the following
columns: <columns>. ↵ In a first step, please specify
```
one of the columns that act as a document-level key, meaning that all extracted values should be unique per document. `Only output the name of that column without`

Table 5: Chat-GPT-3.5 is able to correctly identify the *document-level key* and whether a a latent table is *multi-row* on all of our four datasets.

Task	Accuracy
Identify *document-level key* per table	100%
Identify *multi-row* per table	100%

any explanations. ↩ ↩ Sample of texts: ↩ <sample of three texts>

We used the following prompt for classifying latent tables on multi-row texts:

    I want to transform the information stored in texts
into a table. The table should have the following
columns: <columns>. ↩ In a first step, please **determine
if one or many rows are extracted per text document.
Please only output "many" or "one"** without any explanations.
↩ ↩ Sample of texts: ↩ <sample of three texts>

6.8 Exp. 7: Varying Text Lengths

Our work focuses on small to medium-sized texts (such as patient reports) but can also support longer texts using a windowed approach as explained in Section 5.1. The main idea is to extract the values from longer text by processing them using a sliding window. We can feed each text window independently into the ELEET-model and collect the extracted values across windows afterwards to compose the output table. For *multi-row* texts, the procedure is slightly more involved due to the two phases of Algorithm 5. Here, we again process the different text windows independently for each phase. However, after extracting the values for the *document-level key* in the first phase, we need to first collect and deduplicate all extracted values *across windows*, before we can continue with the second phase. After extracting all values for all text windows in the second phase, we can collect them to compose the output table.

We found that this windowed approach works well and does not come with a degradation in extraction quality, as shown below. This indicates that large context windows are not required to extract values from text. To show that our model works well independent of text length, we break down the extraction quality by text length on the rotowire and T-REx datasets, which are the two datasets that contain long texts. For this experiment, we removed all aggregations and selections from our benchmarks to show the effects of pure extractions. Then, we executed all queries on the datasets and recorded the results. For each tuple in an output table, we trace back from which text the tuple is coming (i.e., from which text have the tuple's values been extracted). Then, we compute an F1 score for each text separately by only considering the tuples that were produced from each text. Figure 7 shows the result, where we plot the text length on the X-axis and the F1 score for each text on the Y-axis. The lines in the background show the F1 scores of each text, while the lines in the foreground are a smoothed version of the results by averaging the over the last 10 texts. The dashed black line denotes the context length of the ELEET-model (512 tokens). We can clearly see that the performance of ELEET is stable under

increasing text length. Important is that ELEET has the best F1 score despite having a smaller context length than the baselines, and the accuracy does not drop for texts that are longer than the context length of the ELEET-model (right to the dashed line).

7 RELATED WORK

Multi-Modal Data Systems. Integrating textual data into data systems is a long-standing problem. Early work implemented a join that retrieves the most relevant documents for each tuple without extracting any structured data from the text [7]. Later systems allowed users to write small extraction functions or UDFs to extract structured data from text and then allowed them to easily combine these extraction functions by writing SQL or datalog queries [10, 21, 50, 52]. However, these systems still require the user to write extraction functions, which is not necessary in ELEET. Other works focus only on filtering the multi-modal data using natural language filters [26, 34]. Some early systems extract data from texts by using techniques such as information extraction, named entity recognition, part-of-speech tagging, and/or hand-designed grammars or rules [6, 20, 28, 55]. However, they typically do not consider the multi-modal case where tabular data is available in addition to texts. More recently, NeuralDB [57] and WannaDB [23] use pre-trained language models to run queries directly on text documents, similar to ELEET. Both systems, again, do not consider the multi-modal case and do not support *multi-row* operations, where multiple tuples are extracted from a single text document. Symphony [8] can query multi-modal data using natural language. The setting in data lakes differs from databases since the main concern is retrieving data from multi-modal data sets. For retrieval, they propose an information compression pre-training objective to embed many modalities in the same latent space. Caesura [61] uses LLMs to generate multi-modal query plans similar to ours. Plugging ELEET into Caesura is an interesting avenue for future work.

Extraction of Tabular Data. GIO [17] and Evaporate [3] tackle the related problem of translating custom data formats (e.g., machine logs) or semi-structured documents (e.g., XML), respectively, into tables using code generation. While GIO uses template-based code generation, Evaporate uses LLMs such as GPT-4. However, we found that code generation is hard with free-form text, as in our setting. Text-to-Table [69] and STable [46] are sequence-to-sequence models trained to transform tables into text. Both introduce several model adjustments to ensure that the model outputs a correctly structured table. Different from Text-To-Table, STable can output table cells in arbitrary order. Unlike our work, both are trained in a supervised manner from scratch for every new data set.

LLMs for Data Management. By now, many research groups have integrated language models into data systems to tackle various data management tasks. Language models have been used to tune databases [60], solve data engineering tasks like entity matching, entity resolution, or missing value imputation [1, 18, 35, 40, 65], or augment databases with knowledge stored inside of LLMs [49, 63]. Moreover, Foundation Models for data management promise to be a solution for many different data management problems [33, 64].

Pre-training Models. Large pre-trained language models [4, 15, 37, 44] are by now dominating NLP and are quickly adapted for multi-modal [32, 39, 53, 54] and tabular data [13, 25, 66]. To reduce the
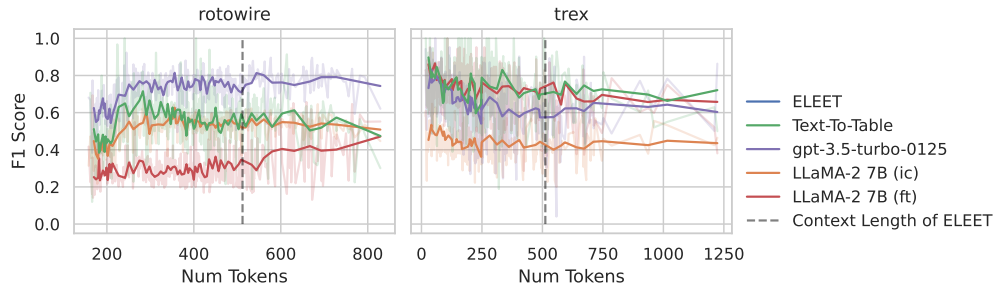
Figure 13: Extraction quality of ELEET broken down by length of text. The dashed black line denotes the context length of the ELEET-model (512 tokens). The performance of ELEET is stable under increasing text length. Important is that ELEET has the best F1 score despite having a smaller context window than the baselines and the accuracy does not drop for texts that are longer than the context length of the ELEET-model (right to the dashed line).

overhead of adaption to downstream tasks, pre-training objectives began to be more aligned with the downstream task for many core-NLP [19, 27, 30, 48] and also structure-aware tasks [36, 42, 51, 72]. Most similar to our pre-training objectives are those pre-training procedures that rely on weak or distant supervision to align pre-training more to the downstream task. ReasonBERT [12] uses a pre-training objective inspired by distant supervision for the downstream task of multi-hop hybrid question answering. StruG's [11] pre-training data set designed for the text-to-SQL task is based on the table-to-text data set ToTTo [43], which was extracted from Wikipedia using heuristics and is much smaller than our data set.

8 CONCLUSIONS AND FUTURE WORK

We presented ELEET, a new execution engine that allows users to seamlessly query textual and tabular data. The cornerstone of ELEET is the concept of multi-modal database operators, which are realizable using a small pre-trained language model. As a result, multi-modal queries containing multi-modal database operators can be executed on new data sets with only minimal fine-tuning overhead and high performance. Clearly, there are still many challenges when integrating ELEET into a real database system. The query parser must be able to instantiate the multi-modal query plans containing multi-modal and traditional operators. The query optimizer must reason about the effects of replacing traditional operators with multi-modal ones. Moreover, an extension to other modalities like images is also an interesting avenue for future work.

REFERENCES

[1] Naser Ahmadi, Hansjorg Sand, and Paolo Papotti. 2022. Unsupervised Matching of Data and Text. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 1058–1070. https://doi.org/10.1109/ICDE53745.2022.00084

[2] Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. 2023. PaLM 2 Technical Report. arXiv:2305.10403 [cs]

[3] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proc. VLDB Endow.* 17, 2 (2023), 92–105. https://www.vldb.org/pvldb/vol17/p92-arora.pdf

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[5] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: exploring the power of tables on the web. *Proc. VLDB Endow.* 1, 1 (2008), 538–549. https://doi.org/10.14778/1453856.1453916

[6] Michael J. Cafarella, Christopher Ré, Dan Suciu, and Oren Etzioni. 2007. Structured Querying of Web Text Data: A Technical Challenge. In *Third Biennial Conference on Innovative Data Systems Research, CIDR 2007, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*. www.cidrdb.org, 225–234. http://cidrdb.org/cidr2007/papers/cidr07p25.pdf

[7] Surajit Chaudhuri, Umeshwar Dayal, and Tak W. Yan. 1995. Join Queries with External Text Sources: Execution and Optimization Techniques. *SIGMOD Rec.* 24, 2 (May 1995), 410–422. https://doi.org/10.1145/568271.223856

[8] Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Samuel Madden, and Nan Tang. 2023. Symphony: Towards Natural Language Query Answering over Multi-modal

Data Lakes. In *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023*. www.cidrdb.org. https://www.cidrdb.org/cidr2023/papers/p51-chen.pdf

[9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. PaLM: Scaling Language Modeling with Pathways. *J. Mach. Learn. Res.* 24 (2023), 240:1–240:113. http://jmlr.org/papers/v24/22-1144.html

[10] Eric Chu, Akanksha Baid, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. 2007. A Relational Approach to Incrementally Extracting and Querying Structure in Unstructured Data. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold (Eds.). ACM, 1045–1056. http://www.vldb.org/conf/2007/papers/research/p1045-chu.pdf

[11] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-Grounded Pretraining for Text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 1337–1350. https://doi.org/10.18653/v1/2021.naacl-main.105

[12] Xiang Deng, Yu Su, Alyssa Lees, You Wu, Cong Yu, and Huan Sun. 2021. ReasonBERT: Pre-trained to Reason with Distant Supervision. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 6112–6127. https://doi.org/10.18653/v1/2021.emnlp-main.494

[13] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. TURL: Table Understanding through Representation Learning. *SIGMOD Rec.* 51, 1 (2022), 33–40. https://doi.org/10.1145/3542700.3542709

[14] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. *CoRR* abs/2305.14314 (2023). https://doi.org/10.48550/ARXIV.2305.14314 arXiv:2305.14314

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[16] Hady ElSahar, Pavlos Vougiouklis, Arslen Remaci, Christophe Gravier, Jonathon S. Hare, Frédérique Laforest, and Elena Simperl. 2018. T-REx: A Large Scale Alignment of Natural Language with Knowledge Base Triples. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*, Nicoletta Calzolari, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Kôiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asunción Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga (Eds.). European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2018/summaries/632.html

[17] Saeed Fathollahzadeh and Matthias Boehm. 2023. GIO: Generating Efficient Matrix and Frame Readers for Custom Data Formats by Example. *Proc. ACM Manag. Data* 1, 2 (2023), 120:1–120:26. https://doi.org/10.1145/3589265

[18] Bar Genossar, Roee Shraga, and Avigdor Gal. 2023. FlexER: Flexible Entity Resolution for Multiple Intents. *Proc. ACM Manag. Data* 1, 1 (2023), 42:1–42:27. https://doi.org/10.1145/3588722

[19] Michael R. Glass, Alfio Gliozzo, Rishav Chakravarti, Anthony Ferritto, Lin Pan, G. P. Shrivatsa Bhargav, Dinesh Garg, and Avirup Sil. 2020. Span Selection Pretraining for Question Answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 2773–2782. https://doi.org/10.18653/v1/2020.acl-main.247

[20] Michael Gubanov and Philip Bernstein. 2006. Structural text search and comparison using automatically extracted schema.

[21] Michael Gubanov, Michael Stonebraker, and Daniel Bruckner. 2014. Text and structured data fusion in data tamer at scale. In *2014 IEEE 30th International Conference on Data Engineering*. 1258–1261. https://doi.org/10.1109/ICDE.2014.6816755

[22] James R. Hamilton and Tapas K. Nayak. 2001. Microsoft SQL Server Full-Text Search. *IEEE Data Eng. Bull.* 24, 4 (2001), 7–10. http://sites.computer.org/debull/A01DEC-CD.pdf

[23] Benjamin Hättasch, Jan-Micha Bodensohn, Liane Vogel, Matthias Urban, and Carsten Binnig. 2023. WannaDB: Ad-hoc SQL Queries over Text Collections. In *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10. März 2023, Dresden, Germany, Proceedings (LNI)*, Birgitta König-Ries, Stefanie Scherzinger, Wolfgang Lehner, and Gottfried Vossen (Eds.), Vol. P-331. Gesellschaft für Informatik e.V., 157–181. https://doi.org/10.18420/BTW2023-08

[24] Madelon Hulsebos, Çağatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. *Proc. ACM Manag. Data* 1, 1 (2023), 30:1–30:17. https://doi.org/10.1145/3588710

[25] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In *Proceedings of NAACL-HLT 2021*. Association for Computational Linguistics, 3446–3456.

[26] Saehan Jo and Immanuel Trummer. 2023. Demonstration of ThalamusDB: Answering Complex SQL Queries with Natural Language Predicates on Multi-Modal Data. In *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*, Sudipto Das, Ippokratis Pandis, K. Selçuk Candan, and Sihem Amer-Yahia (Eds.). ACM, 179–182. https://doi.org/10.1145/3555041.3589730

[27] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Trans. Assoc. Comput. Linguistics* 8 (2020), 64–77. https://doi.org/10.1162/tacl_a_00300

[28] Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. 2009. SystemT: a system for declarative information extraction. *SIGMOD Rec.* 37 (2009), 7–13. https://api.semanticscholar.org/CorpusID:8749741

[29] Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end Neural Coreference Resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, 188–197. https://doi.org/10.18653/v1/D17-1018

[30] Mike Lewis, Marjan Ghazvininejad, Gargi Ghosh, Armen Aghajanyan, Sida Wang, and Luke Zettlemoyer. 2020. Pre-training via Paraphrasing. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/d6f1dd034aabde7657e6680444ceff62-Abstract.html

[31] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 7871–7880. https://doi.org/10.18653/V1/2020.ACL-MAIN.703

[32] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. 2019. VisualBERT: A Simple and Performant Baseline for Vision and Language. *CoRR* abs/1908.03557 (2019). arXiv:1908.03557 http://arxiv.org/abs/1908.03557

[33] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2023. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263* (2023).

[34] Yuliang Li, Aaron Feng, Jinfeng Li, Saran Mumick, Alon Y. Halevy, Vivian Li, and Wang-Chiew Tan. 2019. Subjective Databases. *Proc. VLDB Endow.* 12, 11 (2019), 1330–1343. https://doi.org/10.14778/3342263.3342271

[35] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (2020), 50–60. https://doi.org/10.14778/3421424.3421431

[36] Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: Table Pre-training via Learning a Neural SQL Executor. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=O50443AsCP

[37] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

19

[38] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=Bkg6RiCqY7

[39] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 13–23. https://proceedings.neurips.cc/paper/2019/hash/c74d97b01eae257e44aa9d5bade97baf-Abstract.html

[40] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proc. VLDB Endow.* 16, 4 (2022), 738–746. https://doi.org/10.14778/3574245.3574258

[41] OpenAI. 2023. GPT-4 Technical Report. https://doi.org/10.48550/arXiv.2303.08774 arXiv:2303.08774 [cs]

[42] Liangming Pan, Wenhu Chen, Wenhan Xiong, Min-Yen Kan, and William Yang Wang. 2021. Unsupervised Multi-hop Question Answering by Question Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 5866–5880. https://doi.org/10.18653/v1/2021.naacl-main.469

[43] Ankur P. Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A Controlled Table-To-Text Generation Dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 1173–1186. https://doi.org/10.18653/v1/2020.emnlp-main.89

[44] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, Marilyn A. Walker, Heng Ji, and Amanda Stent (Eds.). Association for Computational Linguistics, 2227–2237. https://doi.org/10.18653/v1/n18-1202

[45] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick S. H. Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. 2021. KILT: a Benchmark for Knowledge Intensive Language Tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 2523–2544. https://doi.org/10.18653/v1/2021.naacl-main.200

[46] Michal Pietruszka, Michal Turski, Lukasz Borchmann, Tomasz Dwojak, Gabriela Palka, Karolina Szyndler, Dawid Jurkiewicz, and Lukasz Garncarek. 2022. STable: Table Generation Framework for Encoder-Decoder Models. *CoRR* abs/2206.04045 (2022). https://doi.org/10.48550/arXiv.2206.04045 arXiv:2206.04045

[47] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, Jian Su, Xavier Carreras, and Kevin Duh (Eds.). The Association for Computational Linguistics, 2383–2392. https://doi.org/10.18653/V1/D16-1264

[48] Ori Ram, Yuval Kirstain, Jonathan Berant, Amir Globerson, and Omer Levy. 2021. Few-Shot Question Answering by Pretraining Span Selection. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 3066–3079. https://doi.org/10.18653/v1/2021.acl-long.239

[49] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. 2023. Querying Large Language Models with SQL. *arXiv preprint arXiv:2304.00472* (2023).

[50] Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. 2007. Declarative Information Extraction Using Datalog with Embedded Extraction Predicates. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold (Eds.). ACM, 1033–1044. http://www.vldb.org/conf/2007/papers/research/p1033-shen.pdf

[51] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cícero Nogueira dos Santos, and Bing Xiang. 2021. Learning Contextual

[52] Representations for Semantic Parsing with Generation-Augmented Pre-Training. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 13806–13814. https://ojs.aaai.org/index.php/AAAI/article/view/17627

[52] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. 2015. Incremental Knowledge Base Construction Using DeepDive. *Proc. VLDB Endow.* 8, 11 (2015), 1310–1321. https://doi.org/10.14778/2809974.2809991

[53] Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. 2020. VL-BERT: Pre-training of Generic Visual-Linguistic Representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=SygXPaEYvH

[54] Hao Tan and Mohit Bansal. 2019. LXMERT: Learning Cross-Modality Encoder Representations from Transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 5099–5110. https://doi.org/10.18653/v1/D19-1514

[55] Luis Tari, Phan Huy Tu, Jorg Hakenberg, Yi Chen, Tran Cao Son, Graciela Gonzalez, and Chitta Baral. 2010. Incremental information extraction using relational databases. *IEEE Transactions on Knowledge and Data Engineering* 24, 1 (2010), 86–99.

[56] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[57] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Y. Halevy. 2021. Database reasoning over text. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 3091–3104. https://doi.org/10.18653/v1/2021.acl-long.241

[58] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs]

[59] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR* abs/2307.09288 (2023). https://doi.org/10.48550/arXiv.2307.09288 arXiv:2307.09288

[60] Immanuel Trummer. 2022. DB-BERT: A Database Tuning Tool That "Reads the Manual". In *Proceedings of the 2022 International Conference on Management of Data*. ACM, Philadelphia PA USA, 190–203. https://doi.org/10.1145/3514221.3517843

[61] Matthias Urban and Carsten Binnig. 2024. CAESURA: Language Models as Multi-Modal Query Planners. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, CA, USA, January 14-17, 2024*. www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p14-urban.pdf

[62] Matthias Urban and Carsten Binnig. 2024. ELEET: Efficient Learned Query Execution over Text and Tables. *Proc. VLDB Endow.* 17, 13 (2024), XXXX–XXXX. https://www.vldb.org/pvldb/vol17/xxxx.pdf

[63] Matthias Urban, Duc Dat Nguyen, and Carsten Binnig. 2023. OmniscientDB: A Large Language Model-Augmented DBMS That Knows What Other DBMSs Do Not Know. In *Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management* (Seattle, WA, USA) *(aiDM '23)*. Association for Computing Machinery, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/3593078.3593933

[64] Liane Vogel, Benjamin Hilprecht, and Carsten Binnig. 2023. Towards Foundation Models for Relational Databases [Vision Paper]. arXiv:2305.15321 [cs]

[65] Jin Wang, Yuliang Li, Wataru Hirota, and Eser Kandogan. 2022. Machop: an end-to-end generalized entity matching framework. In *aiDM '22: Proceedings of*

the Fifth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, Philadelphia, Pennsylvania, USA, 17 June 2022, Rajesh Bordawekar, Oded Shmueli, Yael Amsterdamer, Donatella Firmani, and Ryan Marcus (Eds.). ACM, 2:1–2:10. https://doi.org/10.1145/3533702.3534910

[66] Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. TUTA: Tree-based Transformers for Generally Structured Table Pre-training. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 1780–1790. https://doi.org/10.1145/3447548.3467434

[67] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. https://doi.org/10.48550/arXiv.2206.07682 arXiv:2206.07682 [cs]

[68] Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. 2017. Challenges in Data-to-Document Generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, Martha Palmer, Rebecca Hwa, and Sebastian Riedel (Eds.). Association for Computational Linguistics, 2253–2263. https://doi.org/10.18653/v1/d17-1239

[69] Xueqing Wu, Jiacheng Zhang, and Hang Li. 2022. Text-to-Table: A New Way of Information Extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022,* *Dublin, Ireland, May 22-27, 2022*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 2518–2533. https://doi.org/10.18653/v1/2022.acl-long.180

[70] Yi-Pu Wu, Jin-Jiang Guo, and Xue-Jie Zhang. 2007. A Linear DBSCAN Algorithm Based on LSH. In *2007 International Conference on Machine Learning and Cybernetics*, Vol. 5. 2608–2614. https://doi.org/10.1109/ICMLC.2007.4370588

[71] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 8413–8426. https://doi.org/10.18653/v1/2020.acl-main.745

[72] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. 2021. GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=kyaIeYj4zZ

[73] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 19–27. https://doi.org/10.1109/ICCV.2015.11

A BENCHMARK QUERIES

In the following, we list all queries used as benchmarks for the evaluations in Section 6 in the paper.

A.1 Rotowire

(1) $(player\_info \bowtie_{name} player\_to\_reports) \ddot{\bowtie}_{path,name} reports.Player$

(2) $player\_stats \ \ddot{\cup} \ reports.Player$

(3) $\ddot{S}can(reports.Player)$

(4) $(player\_info \bowtie_{name} player\_to\_reports) \ddot{\bowtie}_{path,name} \ddot{\pi}_{name,Points,Assists,Steals}(reports.Player)$

(5) $\pi_{name,Points,Assists,Steals}(player\_stats) \ \ddot{\cup} \ \ddot{\pi}_{name,Points,Assists,Steals}(reports.Player)$

(6) $\ddot{S}can(\ddot{\pi}_{name,Points,Assists,Steals}(reports.Player))$

(7) $(\sigma_{0.01}(player\_info) \bowtie_{name} player\_to\_reports) \ddot{\bowtie}_{path,name} reports.Player$

(8) $(\sigma_{0.05}(player\_info) \bowtie_{name} player\_to\_reports) \ddot{\bowtie}_{path,name} reports.Player$

(9) $(\sigma_{0.1}(player\_info) \bowtie_{name} player\_to\_reports) \ddot{\bowtie}_{path,name} reports.Player$

(10) $\ddot{S}can(\ddot{\sigma}_{Points=32[sel=0.056]}(reports.Player))$

(11) $\ddot{S}can(\ddot{\sigma}_{Points=28[sel=0.096]}(reports.Player))$

(12) $\ddot{\chi}_{list,name}(\ddot{S}can(reports.Player))$

(13) $\ddot{\chi}_{list,name}(\ddot{S}can(\ddot{\pi}_{name,Points,Assists,Steals}(reports.Player)))$

(14) $\ddot{\chi}_{list,name}(\ddot{S}can(\ddot{\sigma}_{Points=28[sel=0.096]}(reports.Player)))$

(15) $(team\_info \bowtie_{name} team\_to\_reports) \ddot{\bowtie}_{path,name} reports.Team$

(16) $team\_stats \ \ddot{\cup} \ reports.Team$

(17) $\ddot{S}can(reports.Team)$

(18) $(team\_info \bowtie_{name} team\_to\_reports) \ddot{\bowtie}_{path,name} \ddot{\pi}_{name,Wins,Losses,Totalpoints}(reports.Team)$

(19) $\pi_{name,Wins,Losses,Totalpoints}(team\_stats) \ \ddot{\cup} \ \ddot{\pi}_{name,Wins,Losses,Totalpoints}(reports.Team)$

(20) $\ddot{S}can(\ddot{\pi}_{name,Wins,Losses,Totalpoints}(reports.Team))$

(21) $(\sigma_{0.01}(team\_info) \bowtie_{name} team\_to\_reports) \ddot{\bowtie}_{path,name} reports.Team$

(22) $(\sigma_{0.05}(team\_info) \bowtie_{name} team\_to\_reports) \ddot{\bowtie}_{path,name} reports.Team$

(23) $(\sigma_{0.1}(team\_info) \bowtie_{name} team\_to\_reports) \ddot{\bowtie}_{path,name} reports.Team$

(24) $\ddot{S}can(\ddot{\sigma}_{Totalpoints=99[sel=0.049]}(reports.Team))$

(25) $\ddot{S}can(\ddot{\sigma}_{Totalpoints=102[sel=0.063]}(reports.Team))$

(26) $\ddot{\chi}_{list,name}(\ddot{S}can(reports.Team))$

(27) $\ddot{\chi}_{list,name}(\ddot{S}can(\ddot{\pi}_{name,Wins,Losses,Totalpoints}(reports.Team)))$

(28) $\ddot{\chi}_{list,name}(\ddot{S}can(\ddot{\sigma}_{Totalpoints=102[sel=0.063]}(reports.Team)))$

A.2 T-REx

(1) $nobelPersonaltbl \ \ddot{\cup} \ nobel\_reports.Personal$

(2) $nobelCareertbl \ \ddot{\cup} \ nobel\_reports.Career$

(3) $nobelCareerinfo \ \ddot{\bowtie}_{path} \ nobel\_reports.Personal$

(4) $nobelPersonalinfo \ \ddot{\bowtie}_{path} \ nobel\_reports.Career$

(5) $\ddot{S}can(nobel\_reports.Personal)$

(6) $\ddot{S}can(nobel\_reports.Career)$

(7) $nobelPersonaltbl \ \ddot{\cup} \ \ddot{\sigma}_{countryofcitizenship=unitedstatesofamerica[sel=0.443]}(nobel\_reports.Personal)$

(8) $nobelCareertbl \ \ddot{\cup} \ \ddot{\sigma}_{occupation=physicist[sel=0.127]}(nobel\_reports.Career)$

(9) $\pi_{name,placeofbirth,countryofcitizenship}(nobelPersonaltbl) \ \ddot{\cup} \ \ddot{\pi}_{name,placeofbirth,countryofcitizenship}(nobel\_reports.Personal)$

(10) $\pi_{name,awardreceived,educatedat}(nobelCareertbl) \ \ddot{\cup} \ \ddot{\pi}_{name,awardreceived,educatedat}(nobel\_reports.Career)$

(11) $\ddot{\chi}_{list,countryofcitizenship}(\ddot{S}can(nobel\_reports.Personal))$

(12) $\ddot{\chi}_{list,fieldofwork}(\ddot{S}can(nobel\_reports.Career))$

A.3 Aviation

(1) $(aircraft \bowtie_{aircraft\_registration\_number} aircraft\_to\_reports) \ddot{\bowtie}_{path,aircraft\_registration\_number} reports.incident$

(2) $incidents \ \ddot{\cup} \ reports.incident$

(3) $\ddot{S}can(reports.incident)$

(4) $(aircraft \bowtie_{aircraft\_registration\_number} aircraft\_to\_reports) \ddot{\bowtie}_{path,aircraft\_registration\_number} \ddot{\pi}_{location\_city,location\_state}(reports.incident)$

(5) $\pi_{location\_city,location\_state}(incidents) \ \ddot{\cup} \ \ddot{\pi}_{location\_city,location\_state}(reports.incident)$

(6) $\ddot{S}can(\ddot{\pi}_{location\_city,location\_state}(reports.incident))$

(7) $(\sigma_{0.3}(aircraft) \bowtie_{aircraft\_registration\_number} aircraft\_to\_reports) \ddot{\bowtie}_{path,aircraft\_registration\_number} reports.incident$

(8) $(\sigma_{0.5}(aircraft) \bowtie_{aircraft\_registration\_number} aircraft\_to\_reports) \ddot{\bowtie}_{path,aircraft\_registration\_number} reports.incident$

(9) $(\sigma_{0.8}(aircraft) \bowtie_{aircraft\_registration\_number} aircraft\_to\_reports) \ddot{\bowtie}_{path,aircraft\_registration\_number} reports.incident$

(10) $incidents \mathbin{\ddot{\cup}} \ddot{\sigma}_{location_state=colorado[sel=0.100]}(reports.incident)$

(11) $incidents \mathbin{\ddot{\cup}} \ddot{\sigma}_{weather_condition=visualmeteorologicalconditions[sel=0.433]}(reports.incident)$

(12) $incidents \mathbin{\ddot{\cup}} \ddot{\sigma}_{aircraft_damage=destroyed[sel=0.667]}(reports.incident)$

(13) $\ddot{\chi}_{list,aircraft\_damage}(\ddot{S}can(reports.incident))$

(14) $\ddot{\chi}_{list,location\_state}(\ddot{S}can(reports.incident))$

(15) $\ddot{\chi}_{list,weather\_condition}(\ddot{S}can(reports.incident))$

A.4 Corona

(1) $corona\_stats \mathbin{\ddot{\cup}} reports.summary$

(2) $\pi_{new\_cases,new\_deaths,vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{new\_cases,new\_deaths,vaccinated}(reports.summary)$

(3) $\pi_{date,patients\_intensive\_care,twice\_vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{date,patients\_intensive\_care,twice\_vaccinated}(reports.summary)$

(4) $\pi_{date,incidence,vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{date,incidence,vaccinated}(reports.summary)$

(5) $\pi_{date,new\_cases,new\_deaths}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{date,new\_cases,new\_deaths}(reports.summary)$

(6) $\pi_{new\_deaths,vaccinated,twice\_vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{new\_deaths,vaccinated,twice\_vaccinated}(reports.summary)$

(7) $\pi_{date,vaccinated,twice\_vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{date,vaccinated,twice\_vaccinated}(reports.summary)$

(8) $\pi_{new\_deaths,incidence,vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{new\_deaths,incidence,vaccinated}(reports.summary)$

(9) $\pi_{new\_cases,new\_deaths,incidence}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{new\_cases,new\_deaths,incidence}(reports.summary)$

(10) $\pi_{date,new\_cases,patients\_intensive\_care,vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{date,new\_cases,patients\_intensive\_care,vaccinated}(reports.summary)$

(11) $\pi_{new\_cases,new\_deaths,incidence,twice\_vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{new\_cases,new\_deaths,incidence,twice\_vaccinated}(reports.summary)$

(12) $\pi_{date,new\_cases,new\_deaths,patients\_intensive\_care}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{date,new\_cases,new\_deaths,patients\_intensive\_care}(reports.summary)$

(13) $\pi_{date,new\_cases,vaccinated,twice\_vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{date,new\_cases,vaccinated,twice\_vaccinated}(reports.summary)$

(14) $\pi_{date,new\_deaths,patients\_intensive\_care,twice\_vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{date,new\_deaths,patients\_intensive\_care,twice\_vaccinated}(reports.summary)$

(15) $\pi_{date,new\_deaths,incidence,vaccinated}(corona\_stats) \mathbin{\ddot{\cup}} \ddot{\pi}_{date,new\_deaths,incidence,vaccinated}(reports.summary)$