STAR: Efficient, Scalable, and Modular Retrieval of Statistical Tables

Antoine Gauquier[1][0009−0005−9573−6364], Simon Ebel[2][0009−0003−0045−0175], Helena Galhardas[3][0000−0002−9330−3910], Théo Galizzi[2][0009−0000−6759−6783], Ioana Manolescu[2][0000−0002−0425−2462], Aurélien Peden[2][0009−0009−1239−7889], and Pierre Senellart[1][0000−0002−7909−5369]

[1] DI ENS, ENS, CNRS, PSL University & Inria, Paris, France
{antoine.gauquier,pierre.senellart}@ens.psl.eu
[2] Inria & Institut Polytechnique de Paris, Palaiseau, France
ioana.manolescu@inria.fr
[3] INESC-ID & IST, Universidade Lisboa, Lisbon, Portugal

Abstract. Informed public debate needs high-quality data. In this context, high-quality statistical data sources are a valuable category of reference information based on which a claim can be checked. To facilitate the work of journalists or other fact-checkers, users' questions about a specific claim should be automatically answered based on statistical tables. This task is complicated by the large number, size, and variety of statistical datasets. We introduce the *statistical table discovery* problem (STD, in short), which aims, given a natural language question and a set of statistical datasets (multidimensional tables), to find the tables most relevant for the question. We then describe STAR, an algorithm for solving the STD problem. Unlike existing table discovery (TD) solutions aimed at relational tables, STAR is devised specifically for multidimensional ones. Further, STAR treats the space and time dimensions of statistical datasets separately and provides a modular architecture that supports flexible table retrieval and question answering under spatio-temporal constraints. We experimentally show that these features, together, make STAR outperform state-of-the-art TD systems adapted to the STD problem, in terms of scalability, search quality, pre-processing and question answering time.

Keywords: Statistical Tables · Table Discovery · Table Retrieval

1 Introduction

Public debates often involve *metrics* about a society, a country, or a region. Sample metrics attracting interest and controversy include inflation, growth, (un)employment, education, life expectancy, etc. Such metrics are typically measured by national organizations, such as France's INSEE (Institute for Economic Studies and Statistics), the Labor and Census Bureaus in the US, or *ministries*,

such as those of Justice, Education, etc. International bodies, such as Eurostat in Europe, the International Monetary Fund, and the UN, also gather statistics over many countries. Smaller-scale entities, such as trade unions or NGOs, may also gather statistics on specialized topics, e.g., work-related health issues for people in a given job or the presence of wildlife species or pollutants in an area.

The *value of a given metric, on a given spatial scale, and a given point in time*, such as "deaths in Belgium in 2021", is a useful ingredient for public debate. However, even though statistical datasets are published as Open Data, finding such values is not easy. First, statistical data are published in a *variety of formats* such as CSV, Excel variants, HTML tables, RDF graphs, or specific standard formats such as SDMX [43], an XML vocabulary for statistical (multidimensional) data and metadata exchange. Further, *a statistics file may be very large*, with two consequences: on the one hand, it may be published *compressed*, perhaps together with other types of files, thus search engines may not index its content; on the other hand, it is cumbersome for users to look up a value in a file of millions of rows (such as published by Eurostat, as we discuss later on).

To make statistical data more accessible to use, *question answering methods over statistical datasets* are needed, where a sample question is: "How many deaths were recorded in Belgium in 2021?" The scientific literature comprises a set of methods that, given a set of tables (usually CSV files) and a user query (a phrase or set of keywords), find the most relevant table(s); this is known as the **table discovery** problem (**TD**, in short). State-of-the-art methods include, e.g., Solo [50], Pneuma [6], and Birdie [22] (see Section 7 for a more detailed discussion of prior work). Separately, methods have been developed to extract question answers from a given table, the so-called **table question answering** (**TQA**, in short) problem.

In recent years, we have been collaborating with journalists from Radio France, the French national public radio broadcaster, developing the **StatCheck** system, which solves the TD problem *for statistical datasets*. In practice, StatCheck is already used by journalists at Radio France, helping them quickly find relevant statistics in large datasets and saving valuable time. Beyond statistics use in journalism, TD for statistical datasets also arises, e.g., in a business intelligence setting where numerous spreadsheets hold aggregated numerical data, in science such as epidemiology, public health, or education sciences, where outcomes (e.g., people with a certain diagnosis, or students at a certain level of study and performance, etc.) are counted and presented in aggregate form. Improving over earlier work [10], StatCheck has been demonstrated in [9] and recently with a new algorithm, called **STAR** (see below) in [7]. StatCheck is based on the observation that *a statistical dataset (table) is an (unordered) set of statistical facts*, where each fact consists of a set of *dimensions*, together with their values, and a *measure*. For instance, in the example above, the dimensions with their respective values are ("location"="Belgium"), ("year"="2021"), while the measure is "number of deaths". Further dimensions may be present in the data and/or in users' search, e.g., ("age"="toddlers"), ("prior pathology"="asthma"), etc.

Understanding a statistical dataset as a set of facts enables investigating *order-independent* methods for TD and TQA; this has also been noted in [6, 9, 10, 50]. In contrast, several existing TQA methods start by *serializing* table content (typically rows) and leveraging Language Models (LMs, in short), possibly fine-tuned on these serializations [17]. Such methods, by design, reflect (or learn) *the serialization order* of cells in a row, yet this order is meaningless in statistical tables. As a consequence, result quality may be negatively impacted. A second core insight in StatCheck [7, 9], is that *the fact values, which make up a majority of cells in statistical tables, answer questions but are not part of the questions themselves*, for TD nor for TQA. Only the table cells *other than fact values* need to be pre-processed (indexed), so that at query time *the index can lead to the fact values* that answer the question. As we will show, this keeps StatCheck indexes of reasonable size even on large statistical corpora, on which indexing is unfeasible (taking excessive time or space) or much slower for competitor systems [6, 22, 50]. Finally, the novel insight of [7] is: statistical data dimensions, *time* and *space* are omnipresent in statistical data, and frequent also in user queries. Therefore, the *S*pace- and *T*ime-*A*ware Statistic *R*etrieval (**STAR**, in short) method solves the TD problem by *extracting space and time dimension values from the statistics*, and from user queries, and *separately processing the time, space, and remaining part of the search query*. As shown in [7], this has significantly improved the quality of StatCheck results. This paper extends our previous work, including the ICDE publication [18] and the demonstration [7], a prior iteration of the same system.

In [18], we made the following contributions.

- We formalized the new **Statistical Table Discovery** (STD, in short) problem as a variant of TD, adapted to the particularities of *multidimensional statistical tables*.
- We **detailed the STAR method** for the STD problem, which had only been briefly outlined in the demonstration [7]. While the demonstration provided limited evidence for some core choices in STAR, the ICDE paper detailed it, and thoroughly compared it to state-of-the-art TD systems on the STD task, on a new benchmark (see below).
- We **improved over [7]** by replacing the popular SBert [42] phrase embedding model used in [7] with the recent PEARL [12] model better adapted to short phrases; this improved result quality by up to 14%. We introduced a relaxation mechanism that allows STAR to return tables semantically relevant even when no exact match exists for the specified time or location. Additionally, we extended STAR to handle natural language questions, which are more suitable than keywords for complex queries and pave the way for combining STAR with TQA systems for full Statistical Question Answering.
- We **generalized STAR's score** to reflect the distance between a user question and a candidate statistical table, along three dimensions: space, time, and the measure itself, enabling it to identify more answers that are relevant for a given question.

- We presented a **novel, comprehensive performance comparison** on the STD task, between STAR, the recent systems [6,22,50], and the classic BM25 retrieval metric. Our experiments demonstrated that: (i) when user questions use the exact terminology of the datasets, the BM25 method outperforms all the others in terms of result *quality*, while also providing fast indexing and query answering; (ii) when questions are reformulated to use synonyms instead, *STAR with PEARL embeddings yield the highest-quality results*, while BM25 result quality degrades drastically; (iii) the STAR approach *scales up to large statistical corpora*, on which Solo [50] and Birdie [22] are unfeasible (out-of-memory errors) while Pneuma [6] is 12× slower.
- To support further research on the STD problem, we shared a **novel STD benchmark consisting of 7 605 statistical tables from Eurostat, a gold-standard set of 200 questions with carefully curated answers, and nearly 5 000 manually annotated (question, table) pairs.** This first STD benchmark, on high-quality, real, and varied Open Data, does not suffer from errors in benchmarks used in prior TD work (see Section 6.3).

In this journal extension, we make the following additional contributions.

- We introduce **two new algorithms** that describe the retrieval procedure.
- We explain **how STAR can be used to perform TQA**, by returning the most relevant *facts* within tables that satisfy the question's spatio-temporal constraints.
- We provide a **systematic analysis of STAR's architectural flexibility**, including language model modularity, multilingual support, and practical indexing considerations, highlighting how STAR differs from end-to-end neural and LLM-based approaches.
- We provide a **consolidated and quantitative justification of STAR's indexing and retrieval configuration**, by gathering and harmonizing experimental results obtained across previous iterations of the system. In particular, we systematically compare alternative design choices, quantify their respective impact on performance, and make explicit the rationale behind the final configuration adopted in this journal version.
- We detail **the construction of the BM25+Syn. baseline**, an augmented BM25 variant leveraging synonym expansion, **analyze and explain its performance** on both questions that match dataset terminology exactly, and reformulated ones.
- We provide a **deeper analysis of the impact of replacing SBERT with PEARL** on the Eurostat dataset. In particular, we analyze the subset of questions where SBERT performs better, showing that although such cases often involve long or complex queries, counterexamples exist. By plotting mean relevance scores as a function of question length, we demonstrate that STAR with PEARL follows trends observed in other models, rejecting the hypothesis that it struggles with longer or more complex queries.
- We expand the discussion of **related work beyond TD systems**, covering recent NLP approaches to TQA as well as emerging end-to-end question-answering solutions.

It is also worth noting that StatCheck with STAR is much more *frugal* than recent competitors [6, 22, 50]: STAR does not rely on large language models (LLMs), reducing the energy consumption and computational costs incurred by statistical question answering; also, running it does not require a GPU. While LLMs have unparalleled abilities for text generation, summarization, etc., training and using them has significant environmental costs [11], which argues for using them only for problems that lower-footprint tools cannot address. On the STD task, StatCheck with STAR significantly outperforms baselines *simultaneously* in terms of answer quality, indexing and query speed, scalability, low resource use.

This paper is organized as follows. Section 2 defines important background concepts and presents our problem statement. Section 3 and Section 4 describe our approach for solving the STD problem, then Section 5 discusses STAR's architectural flexibility, multilingual capabilities, and key configuration choices. Section 6 presents our experimental validation. We discuss related work in Section 7 before concluding. Supplementary material including datasets, labeled evaluation questions, experiment results and scripts to compute metrics are available at [20]. The code of STAR is available at [19].

2 Preliminaries

We describe the datasets we consider, before formally stating the problem we study.

2.1 Statistical tables

A statistical dataset, encountered on the Web under the form of a file in Excel, CSV, SDMX [43] format, etc., comprises information which we describe in classic data warehouse [26] and temporal databases [27] terminology below.

A metric is something measured multiple times and reported in a dataset, e.g., "unemployment", "food bank beneficiaries", "tennis amateur licenses", etc. A dimension characterizes a certain property (feature) of a metric. Sample dimensions are the time and location attached to the facts counted by the metric; age, gender, or other properties of a population, etc. A dimension may have multiple granularities, that may be hierarchically organized in a directed acyclic graph (DAG, in short). For instance, location can be reported at granularities: continent, country, region, department, city, etc. Similarly, time can be reported at granularities: year (Y), quarter (Q), month (M), week (W), day (D), e.g., child births are counted per day of the year. A dimension can take different dimension values. The granularity of a value is the lowest level in the dimension hierarchy where the value is given. For instance, "2025", "March 2025", and "March 13, 2025" are three temporal values. Their granularities are, respectively, Y, M, D. A fact consists of: a metric; a set of (dimension = value) pairs; a "datum"[4] which is a numeric value;

[4] While this is the *value of the metric*, we call it data to avoid a confusion with the *dimension values* above.

optionally, one or several comments, e.g., "under consolidation", or "partial data as of 04/2024", etc. The datum can be an absolute value, or a relative one, e.g., a share or percentage. In practice, the dimension name may be missing, e.g., when deaths are separately counted among Men and Women (values of the dimension "gender"), the word "gender" (dimension name) may be absent.

A table contains a set of facts together with an optional title and some comments. For example, in Figure 1, the title appears at the top underlined; below on the left, we depict some cubes (with the dimensions: country, branch of the economy, and year), each of which is characterized by one value for the (implicit) dimension "age group". These cubes hold a set of four-dimensional facts. At the right of this conceptual representation, we show a statistical table as it appears in an Excel or CSV file, or HTML table, within a statistical portal.

A statistical source is a provider of tables, such as INSEE, Eurostat, the US Census Bureau, etc.

2.2 Problem statement

Let $\mathcal{D} = \{T_1, T_2, \ldots, T_n\}$ be a dataset containing n tables T_i, $1 \leqslant i \leqslant n$. Without loss of generality, we assume all tables from $\mathcal{D}$ come from the same provider (source) P; our approach directly generalizes to multiple providers. We consider **natural language questions** requesting one or several fact values. The question is free text, but we assume it mentions at least a metric and typically one or several dimensions values, that may (or may not) be accompanied by dimension names. In the sample question q_1: "How many US people own a smartphone?" the metric is "people owning a smartphone" and the dimension is "location"="US". In the sample question q_2: "What part of EU agricultural subsidies has been spent on soybean-growing farms in Poland in 2020?", the metric is "agricultural subsidies" while the dimensions are "location"="Poland", "category"="soybean-growing farm", "year"="2020". Note that: (i) dimension names are often implicit, in particular location and time; (ii) some crucial dimensions may be absent, e.g., the time dimension in q_1, even though it plays an important role in determining the fact (the percentage of US population owning a smartphone grew from 35% in 2011 to 91% in 2024) (iii) metric names are specified in varied, flexible ways, characteristic of natural language questions; in particular, they may differ quite significantly from the metric names used in $\mathcal{D}$.

We say a **fact** f, present in table T_i, **answers a question** q when the fact's metric and dimension-value pairs are identical or close to those stated in the question. Several facts may answer a given question, e.g., if the question specifies fewer dimensions than the facts have. For instance, high-school students studying Advanced Maths may be recorded by gender in a table, while the question does not mention gender. Also, some facts may answer a question better than others, e.g., a fact f_1 about "job seekers for less than 3 months" answers a question about "short-term unemployment" better than a fact f_2 about the "long-term unemployed". We say a table T **answers a question** q when T contains at least one fact answering the question.

The **Statistical Table Discovery (STD)** problem we consider is: given a question q, find a ranked list of $\mathcal{D}$ tables $T_1^q, T_2^q, \ldots, T_N^q$, for some integer N, such that each T_j^q, $1 \leqslant j \leqslant N$, answers q to some extent, and the tables are sorted in the decreasing order of their relevance (or score).

Solving the problem requires a scoring method that quantifies how relevant a fact is for a question; given the flexibility and ambiguity of natural language, the gold standard comes from human evaluation; prior systems made different proposals in this area. We will present our ranking method below.

Why TD does not solve (well) STD. One may wonder why not convert each statistical table T into a relational one, R_T, then apply existing TD methods on the latter. Assume T is characterized by the dimensions $d_1, \ldots, d_m$, where each d_i ranges among n_i distinct values for some integers $1 \leqslant i \leqslant m$, $1 \leqslant n_i$. T holds $n_1 \cdot \ldots \cdot n_m$ facts which is also its number of cells (ignoring one or a few header row(s) and/or line(s)). R_T should have one attribute for each dimension, and one row for each fact f, of $m+1$ cells (one with each d_i value characterizing f, and one with the measure value of f). Thus, R_T has $(m+1)$ times more cells than T. The actual table size increase is even higher, due to repeated appearance of dimension values in all the rows; even though dimension values are short strings, they still occupy more space than measure values (numbers). This is problematic even for very recent TD systems that *run out of memory* on large tables (see Section 6). But even more problematic is the fact that *the relational table will lack many attribute names*: these would be the *dimension names*, that are often absent in statistical datasets. Equally problematic is the fact that TD methods on relational tables *crucially depend on the attribute names* (table schema), see Section 7. This is why below we develop specific STD methods.

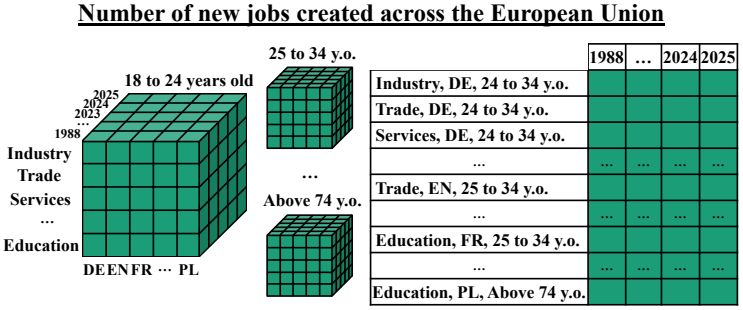


Fig. 1: Sample statistical dataset: conceptual multidimensional view (left); two-dimension serialization (right).

3 Indexing statistical datasets

We now describe our method for *analyzing and indexing the tables* prior to question answering: linguistic content (Section 3.1), respectively, time and location (Section 3.2).

3.1 Indexing the datasets' linguistic content

The content of a table is described (in human-understandable terms) in its title, comments, and any text found in headers. A proper representation and indexing of these components is crucial for both the retrieval and re-ranking phases. Accordingly, most TD methods, e.g., [6, 22, 50], and modern TQA techniques, e.g., [24], compute from each table one or several texts, which are then projected in a high-dimensional space via a Language Model embedding.

Given a table T, several methods can be used to reflect the linguistic content of T (again, recall that data are numbers and not text). We call these **table representation functions**, each of which returns a set of one or more strings:

- $f_{\mathrm{t}}(T)$ returns the table title.
- $f_{\mathrm{ch}}(T)$ returns the concatenation of all column header cells, which always occupy the first (topmost) row in a table, and may or may not span over a few more subsequent rows. These have been used in many prior works, e.g., [14,44], because they intuitively correspond to a relational table's schema (attribute names). However, in statistical tables (Section 2.1), interesting linguistic content is also present in row headers, i.e., leftmost column(s).
- $f_{\mathrm{rh}}(T)$ returns the concatenation of all row header cells, i.e., those in the leftmost column(s), symmetrically to f_{ch}.
- $f_{\{\mathrm{h}\}}(T)$ returns a set of all distinct strings appearing in a (row or column) header cell of T.
- $f_{\mathrm{h}}(T)$ returns the concatenation of all row and column headers, i.e., the concatenation of strings in $f_{\{\mathrm{h}\}}(T)$.

For instance, in the statistical table in Figure 1, $f_{\mathrm{t}}(T)$ is "Number of new jobs created in the European Union", header cells include "Industry, DE, 24 to 34 y.o." and all the cells below, as well as "1988" up to "2025". $f_{\mathrm{h}}(T)$ is a single string starting with "1988", and ending with "Education, PL, Above 74 y.o." with some separator. $f_{\mathrm{ch}}(T)$ is simply the concatenation of column headers "1988" to "2025", again with separator. Whatever the choice of a function f, we will apply an *embedding*, denoted ϕ, to transform each $f(T)$ into a multidimensional vector, which we insert into a **multidimensional index** providing efficient **approximate nearest neighbor (ANN)** look-up. At query time, an embedding $\phi(q)$ computed from the question is used as a lookup key, and we search for the closest vectors, thus, the closest table definitions.

The choice of the string representations, and, to a lesser extent, the choice of the embedding method, are crucial for the quality of our solutions. Regarding the string representations, we make the following remarks.

First, f_{t} may miss a large part of the linguistic content of a table. For instance, in our example, it does not include the words "Industry" and "EN" which may lead to the sample table being (wrongly) considered not very relevant for a question about "industry in England".

Second, as mentioned in Section 1, functions such as f_{ch} and f_{h}, which concatenate several header cells from a table T, impose a certain *order* over their content, however, this order is meaningless from the perspective of understanding the data. For instance, "Belgium" being just before "Bulgaria" in a sequence of header cells storing countries does not entail any connection between them. Language models, however, are strongly sensitive to proximity and order between words, given that these are crucial in natural language. Thus, the semantic signal of the concatenated string suffers from some noise.

Third, concatenating dimension values leads to long strings which may make a table appear not similar to the question. Considering again the "industry in England" question, if all economy branches appear in the row headers, f_{h} will include "Agriculture", "Tourism & Hospitality", "Commerce", etc., as well as "Industry". *The linguistic signal of the concatenation may appear to differ significantly from each of the concatenated strings, again potentially leading to a false negative.* This has been experimentally demonstrated in STAR [7] and it has been known under the name "Lost in the middle" in Natural Language Processing (NLP) research [35].

Fourth, note an important difference with respect to prior TD or TQA methods for relational (not multi-dimensional) data. In relational tables, *questions can carry over any attribute in any record.* For instance, "what is the address of the city hall of Boston?" needs the "city name" and "city hall address" attributes from the row describing Boston. In such settings, indexing $f(T)$, where f is a table representation function that does not reflect all the table content (all attributes from all the rows), risks false negatives: an index look-up may fail to find an entry matching the question, even when one exists. This is why recent TQA methods rely on a first step (typically a TD system) to *pre-select a small set of tables*, which are *integrally given (at query time) to an LM* that has been trained to extract from each table the most relevant facts for a question. When the answers are in tables of hundreds of thousands, or millions of lines, depending on the LM, it may be impossible to feed them to the model, or some chunking may be needed, and TQA may be slow. Also, (i) depending on the LM used, a GPU may be needed, or there may be significant computational costs; and (ii) LM answers do not always come with a clear explanation or provenance. In contrast, in the STD problem, *questions only refer to the linguistic content* of tables. This enables us to index this content only, and not the many other table cells (fact values).

Based on these four observations, the representation $f_{\{\mathrm{h}\}}$, returning all header cells' content, is more likely than f_{h} and f_{ch} to avoid false negatives. Further, table titles contain information that may be absent from the headers, e.g., typically, the measure name. For this reason, we decide to also use f_{t}. Thus, the total

set of strings that we need to embed through ϕ and index for a table T is $f_{\mathrm{t}}(T) \cup f_{\{\mathrm{h}\}}(T)$.

3.2 Separate, exact indexing for locations and time

Directly embedding $f_{\mathrm{t}}(T) \cup f_{\{\mathrm{h}\}}(T)$ still has limitations.

First, in short titles, location names may skew the similarity between titles and questions. For instance, "Jobs in Saint-Pierre-et-Miquelon" (a French overseas territory) may seem similar to "Health in Saint-Pierre-et-Miquelon", even though the topics are quite different.

Second, when embedding locations, similarity between $\phi(l_1), \phi(l_2)$ may not correlate well with users' natural understanding of location hierarchies (Section 2.1). For instance, let l_1, l_2 be the French regions "Île-de-France" and "Provence-Alpes-Côte d'Azur", and l_3 be the French department "Yvelines", a subdivision of l_1. Then, $\phi(l_1)$ may be closer to $\phi(l_2)$ than to $\phi(l_3)$ because both l_1 and l_2 are regions thus they appear in similar contexts in the language's training data, e.g., "the region council has decided", "the region's subsidies to high schools", etc. However, for a question about some facts from "Île-de-France" (l_1), a table with the right facts from "Yvelines" (l_3) is clearly relevant, because l_3 is in l_1, whereas the same facts measured in l_2 are not (wrong location). Similar remarks hold concerning time values. For human users, "Q3, 2023" and "November, 2023" are clearly part of "2023", while "November 2024" is not, even though linguistically it may look more similar. Thus, among locations, or among time values, **the meaningful relationship that should be exploited by question answering is inclusion** (parent-child or ancestor-descendant in the respective DAGs, discussed in Section 2.1).

For these reasons, we decide to **extract location and time information** from f_{t} and $f_{\{\mathrm{h}\}}$ results, as follows. We call *spatial value* (or **sval**, in short) any individual spatial unit, e.g., "New York City", "NY State", etc. The **spatial hierarchy** of all sval in the dataset $\mathcal{D}$ is a directed acyclic graph (**DAG**). We call *temporal value* (or **tval**) any mention of time found in the data; tvals are also organized in a natural **time hierarchy**.

Locations on Earth are described by high-quality, Open Data reference sources, e.g., GeoNames, or dictionaries standardized by statistical institutes, e.g., Cog [1] or NUTS [3]. Building a complete hierarchy of time values over a given interval and at a given granularity is even easier. Thus, without loss of generality, we will denote by $\boxed{H_{\mathrm{S}}}$, respectively, $\boxed{H_{\mathrm{T}}}$ the hierarchies of space and time dimension values that may occur in $\mathcal{D}$. In particular, the geographic reference H_{S} (also) provides *alternate names* for a given sval reflecting renaming along the time, shortnames and acronyms, or different languages. H_{S} may also contain *same-name svals*, e.g., "Hamburg" as the name of a German city or of the corresponding state.

Based on H_{S}, we assign to each statistical source (Section 2.1) a **default location**, e.g., for Eurostat, this is "the 27 EU states", for the US Census Bureau, it is "USA", etc. This will allow interpreting user questions that do not specify a location, as we explain below.

We **index each table T by its sval and tval values**, as follows. We call **spatial scope**, respectively, **temporal scope** of a table T, denoted σ_T, respectively, τ_T, its sval, respectively, tval sets.

To compute σ_T and τ_T, first, we extract location and time occurrences from the table title $f_{\mathrm{t}}(T)$, using a small trained language model. We also need to extract them from the many table header cells. However, in contrast with the title that is a well-formed phrase, each cell usually contains a very short string, on which extraction performed poorly due to the lack of context. This happened especially with locations, which may be confused with dimension values, e.g., Apple is the name of a location, but can also appear in a header cell to specify the fruit, e.g., when reporting fruit production.

Thus, we apply the following heuristic: (i) We tentatively identify svals in header cells by looking for an exact match with the name (or an alternate name) from the geographic reference H_{S}. If we find a sval label shared by two different spatial reference nodes, such as "Hamburg" previously mentioned, we include in the table's spatial scope all such sval nodes. (ii) We tentatively identify tvals based on pattern matching. More advanced time value extraction tools exist, e.g., HeidelTime [46], yet in our experiments HeidelTime did not improve accuracy, possibly because time values are well-written in statistical datasets, thus pattern matching is quite successful. (iii) If at least $t\%$ of the values in a given header column (or row) are time (resp., location) values, we consider all of them to be of this type.

For instance, in the table in Figure 1, the values in the top row are all tentatively labeled years and then all validated as such. In the leftmost column, the country codes DE to PL are tentatively mapped to H_{S} and then, based on the fact that there is one in every row, they are all identified as svals. Thus, τ_T is $\{1988, \ldots, 2025\}$, while $\sigma_T = \{\mathrm{DE}, \mathrm{EN}, \ldots, \mathrm{PL}\}$.

For each table T, all the pairs $(s, T.id)$ for $s \in \sigma_T$ are inserted in a **key-value index** $\boxed{I_{\mathrm{S}}}$. Similarly, all $(t, T.id)$ pairs where $t \in \tau_T$ are inserted in a **key-value index** $\boxed{I_{\mathrm{T}}}$.

Finally, to avoid the undesired impact of location and time on string embeddings mentioned at the beginning of this section, from each string $s \in f_{\mathrm{t}}(T) \cup f_{\{\mathrm{h}\}}(T)$, we remove any sval and any tval identified as above. We call the remaining string the **stripped version** of s, denoted $\overline{s}$. We do so carefully based on a syntactic analysis of s, so that $\overline{s}$ remains as natural as possible from a linguistic perspective. For instance, stripping "premature deaths in Poland in 2022 per maternal age" yields "premature deaths per maternal age": we remove "Poland", "2022", and also their leading "in". A stripped string may be empty, if it only consisted of location or time values; this is the case for the header cells on the top row in Figure 1.

For each non-empty stripped string $\overline{s}$ obtained from $f_{\mathrm{t}}(T)$ and $f_{\{\mathrm{h}\}}(T)$, the embedding $\phi(\overline{s})$ is inserted in an **ANN index** $\boxed{L_{\mathrm{t}}}$ (linguistic content from titles), respectively, $\boxed{L_{\mathrm{h}}}$ (linguistic content from headers), paired, in both indexes, with the table ID $T.id$.

Algorithm 1: Compute Scores for Candidate Tables

Input: Stripped question $\overline{q}$, candidate table IDs C^1, penalty pen, and threshold θ
Output: Occurrences and scores of tables

$\phi(\overline{q}) \leftarrow$ Embedding of $\overline{q}$;
$C^2 \leftarrow \left(L_{\mathrm{t}}(\phi(\overline{q}))_{1:N_1} \cup L_{\mathrm{h}}(\phi(\overline{q}))_{1:N_1}\right) \cap C^1$;
Remove entries in C^2 with similarity $< \theta$;
foreach $(\phi(\overline{x_T}), T.\mathrm{id}) \in C^2$ **do**
  Record occurrence of $T.\mathrm{id}$ with x_T;
foreach *table T with at least one occurrence in C^2* **do**
  $\mathrm{score}(T) \leftarrow \sum_{x_T} \mathrm{sim}(\phi(\overline{q}), \phi(\overline{x_T})) - pen$;
return Occurrences and $\mathrm{score}(T)$;

Algorithm 2: Statistical Table Retrieval

Input: The NL question q
Output: The top-N most relevant tables to q

τ_q, $\sigma_q \leftarrow$ Time and location values from q;
s_q, $t_q \leftarrow$ Unique location and time of q (conjunctive semantics otherwise);
$C_q^1 \leftarrow I_{\mathrm{S}}(s_q) \cap I_{\mathrm{T}}(t_q)$;
if $C_q^1 \neq \emptyset$ **then**
  Occurrences, scores $\leftarrow$ Alg. 1 with $C^1 = C_q^1$, $pen = 0$;
else
  Initialize relaxation distance $d \leftarrow 1$;
  while *fewer than N results found* **do**
    $(s_q)^d \leftarrow$ neighbors of s_q at distance d in H_{S};
    $(t_q)^d \leftarrow$ neighbors of t_q at distance d in H_{T};
    foreach $t^d \in (t_q)^d$ **do**
      $C^1 \leftarrow I_{\mathrm{S}}(s_q) \cap I_{\mathrm{T}}(t^d)$;
      if $C^1 \neq \emptyset$ **then**
        Occurrences, scores $\leftarrow$ Alg. 1 with C^1, $pen = d \cdot p$;
    foreach $s^d \in (s_q)^d$ **do**
      $C^1 \leftarrow I_{\mathrm{S}}(s^d) \cap I_{\mathrm{T}}(t_q)$;
      if $C^1 \neq \emptyset$ **then**
        Occurrences, scores $\leftarrow$ Alg. 1 with C^1, $pen = d \cdot p$;
    foreach $s^d \in (s_q)^d$ **do**
      foreach $t^d \in (t_q)^d$ **do**
        $C^1 \leftarrow I_{\mathrm{S}}(s^d) \cap I_{\mathrm{T}}(t^d)$;
        if $C^1 \neq \emptyset$ **then**
          Occurrences, scores $\leftarrow$ Alg. 1 with C^1, $pen = 2 \cdot d \cdot p$;
    $d \leftarrow d + 1$;
return top-N tables by decreasing $\mathrm{score}(T)$;

4 Statistical table retrieval

Algorithm 2 presents the procedure for retrieving the tables most relevant to a natural language question q. The following provides a detailed explanation of its operation.

We begin by extracting possible **question time and location values** from q just like we did from table titles during the pre-processing stage. Because q is a phrase, a language model yields good results for this task. This leads to zero or more H_{S} locations, and zero or more time values from H_{T}. The typical question we expect has one of each; we denote them s_q and t_q respectively, and we describe the next steps considering them present (otherwise, the steps based on the absent s_q and/or t_q are skipped). If there are more than one location (or time) values, we assign them a conjunctive semantics, i.e., we search for tables containing *as many (ideally all)* of the question's s_q and t_q values as possible.

1. We look up I_{S} with s_q, respectively, I_{T} with t_q. Each of these look-ups yields a set of table IDs; let C_q^1, the set of candidate table IDs for q, be their intersection. If $C_q^1 = \emptyset$, no table has in its scope both s_q and t_q; we handle such cases via *question relaxation* (see below). For now, we handle the case when $C_q^1 \neq \emptyset$ (steps (2)–(5) are described by Algorithm 1).
2. We compute the embedding $\phi(\overline{q})$, where $\overline{q}$ is the user question, stripped as described in Section 3.
3. We then perform a look-up in the ANN index L_{t} with $\phi(\overline{q})$ as search key *and* C_q^1 *as constraint*, i.e., we look for the N_1 multidimensional vectors closest to $\phi(\overline{q})$ and whose associated table IDs are in C_q^1. N_1 is an integer larger than N, the number of desired results (recall Section 2.2). This finds the C_q^1 tables whose titles are closest to the stripped question. Similarly, we look up in L_{h} with $\phi(\overline{q})$, to find the header cell(s) most similar to $\overline{q}$.
4. Call C_q^2 the union of the results of the above two look-ups (in L_{t} and L_{h}). Each entry in C_q^2 is of the form $(\phi(\overline{x_T}), T.id)$ where $x_T \in f_{\mathrm{t}}(T) \cup f_{\{\mathrm{h}\}}(T)$. The ID of a given table may appear several times in C_q^2, with different strings x_T, e.g., if in a table titled "Student population", two header cells contain "college students" and "high school students", and the question is about "students", the table is returned by the L_{t} lookup together with its title, and twice by the L_{h} lookup, with each of the relevant header cells.
5. For each table T whose ID appears in some C_q^2 pairs, we **score** T by **summing up** similarities between $\overline{q}$ and $\overline{x_T}$, for each string x_T that lead to an occurrence of T in C_q^2.

Question relaxation. We now show how to handle the case $C_q^1 = \emptyset$, i.e., if no table matches the stated question's location and time. First, this may happen if the look-ups in I_{S} and/or I_{T} have no result. For instance, schoolchildren are not counted at the granularity of a village. Second, some tables may match s_q, while other tables match t_q, but none matches both.

What can be done in such cases? $\mathcal{D}$ may hold facts that answer the question at *a different space or time granularity*. For instance, instead of the student count for a village, a table T' may have the count for the department containing the village. In other cases, the desired facts are *absent* from $\mathcal{D}$ even at different granularities,

but *close* information exists. For instance, $\mathcal{D}$ may have no facts about students in (FR, 2025), because the year is not over yet, but it may have table T'' about students in (FR, 2024). Such tables T', T'' can be seen as best-effort results that come close to the user question even if they do not exactly match it. To find tables such as T' and T'', we *relax* the question as follows.

6. Using the spatial hierarchy H_{S}, we gather all *neighbors of s_q (the query location) at distance 1* in a set denoted $(s_q)^1$, and similarly the neighbors of t_q at distance 1 in H_{T}, the time hierarchy, in the set $(t_q)^1$.
7. We repeat the above process (steps (1)-(2)) with (s_q, t^1) for each $t^1 \in (t_q)^1$: this corresponds to keeping s_q unchanged and relaxing t_q. In parallel, we try it also with (s^1, t_q) for each $s^1 \in (s_q)^1$ (relaxing s_q and keeping t_q fixed). One or both of these partial relaxations may lead to relevant tables, which we score as explained at step (5) above, subtracting from the score a fixed **penalty** p (a constant empirically determined).
8. If first-level neighbors of s_q, t_q yield no results, we try all combinations (s, t) where $s \in (s_q)^1$ and $t \in (t_q)^1$. This corresponds to simultaneously relaxing s_q and t_q at a distance of 1; results obtained in this way are scored with a penalty $2 \cdot p$.
9. We repeat the process by relaxing further, e.g., $(s_q)^2$ contains the neighbors of s_q in S at distance 2, etc., increasing the penalty accordingly, until N results have been found, or all neighbors have been explored.

Note that we do not relax $\overline{q}$, the question stripped of the location and time, *explicitly*, but *implicitly*: this is exactly what the approximate nearest neighbor index over $\phi(\overline{q})$ does (find the closest existing matches). Further, the actual algorithm has two more significant aspects:

- If q specifies no s_q and no t_q, at step (1), we take the *default location of the statistics provider P* (recall Section 3.2) as s_q, and the current year as t_q. If $\mathcal{D}$ holds tables from several providers $P_1, \ldots, P_m$, we answer q separately over each $\mathcal{D}$ subset produced by one provider, with the respective default location as s_q.
- At step (3), it is important to *reject ANN lookup results that are too far from $\phi(\overline{q})$*. If we accept "too distant" matches, we might return a table with the exact s_q, t_q but semantically far from $\overline{q}$, which users would consider irrelevant. To avoid this, at step (3), when the similarity between an ANN look-up result and $\phi(\overline{q})$ is below a **threshold** θ, we discard the result. If, as a consequence, C_q^2 has less than N entries, we relax the location and time dimensions (run steps (6) to (9) above). This ensures the algorithm returns tables pertinent for the meaningful, semantic-rich part of the question $\overline{q}$, while remaining as close as possible in the time and space dimensions.

Best-effort TQA. While TQA is not the main focus of our work, we briefly explain how STAR may also solve it in order to reduce the users' effort, by showing them the most relevant facts. For each table T in an STD answer computed as above, based on StatCheck's knowledge of the table structure, and knowing the strings $\overline{x_T}$ that brought table T in the result, STAR shows to users, next to a link to T, also *the most specific fact(s) of T relevant for q*. Thus, if T has both a

row and a column header pertinent for q, we show the cell at the intersection of the respective row and column; if only a row or column was pertinent, we show (a bounded portion) of that row or column, together with the relevant header. We need to bound this because it may be very large. If only T's title matched q, STAR simply returns a link to T.

5 Architecture and Configuration

We now present the architectural principles and configuration choices underlying STAR. We first describe its modular and multilingual design (Section 5.1), and then detail the indexing and retrieval configuration (Section 5.2), linking these choices to their empirical validation.

5.1 Model Modularity and Multilingual Capabilities

A key strength of STAR's architecture is its *modular separation of language understanding from retrieval logic*. Unlike end-to-end neural systems that embed the representation model into learned parameters, STAR cleanly separates the core retrieval logic (spatial/temporal indexing, relaxation mechanism, scoring) from the embedding function used for text representation. The embedding function is used only during indexing and retrieval, and is never integrated into trainable weights, allowing it to be replaced with any fixed-dimensional model without modifying the core algorithm or retraining. This design provides flexibility across languages, domains, and changing corpora.

STAR's multilingual capacity stems from three architectural choices. First, *spatial and temporal processing is language-agnostic*: location names are matched against standardized hierarchies, and date formats are recognized using general patterns. Second, *STAR can leverage any mono- or multilingual embedding model* to represent text, automatically supporting queries and tables in different (and potentially multiple) languages. Third, *separate indexes can be built for different languages to optimize retrieval*. We evaluated this capability in two scenarios: (i) at the national scale, we indexed INSEE tables in French using finer-grained spatial hierarchies, with queries in French (experiments presented in [7]); and (ii) at a regional and international scale, we indexed Eurostat tables covering European countries and occasionally other regions, using coarser spatial hierarchies; most tables are in English, with some fragments in other languages, and queries in English (see Section 6.1). In both cases, multilingual embeddings were used (SBERT, PEARL; further described in Section 6.2), highlighting STAR's ability to combine language-agnostic reasoning with cross-lingual representations.

The advantages of this approach are clear when comparing STAR to alternative methods, which are described in detail in Section 6.2. Systems such as Solo [50] require training from scratch on question-table-query triples in the target language, a process that is *language-specific, time-consuming, and dependent on the generation of synthetic queries*. Birdie [22] similarly relies on generating synthetic queries for each language and retraining an encoder-decoder

model end-to-end, with *no opportunity for cross-lingual transfer*. LLM-based systems such as Pneuma [6] can leverage multilingual capabilities, but incur high API costs, show variable quality across languages, and require careful prompt engineering. In particular, creating prompts that are both semantically accurate and efficient for multilingual data is inherently difficult, as generic LLMs cannot be directly controlled or adapted without fine-tuning or augmenting them via additional training, which brings the drawbacks of end-to-end neural systems. In contrast, STAR's design allows new languages to be incorporated simply by providing an appropriate embedding model and, if necessary, a regional spatial hierarchy, with no algorithmic changes or retraining required.

These architectural decisions ensure that STAR scales across languages and domains while maintaining high retrieval quality. For STD, where corpora can be multilingual and continuously evolving, this flexibility is essential for practical and scalable deployment. Importantly, it also enables the inclusion of statistics in less-resourced or rare languages, which may require niche monolingual language models, ensuring comprehensive coverage of all countries, populations, and regions.

5.2 Indexing and Retrieval Configuration

We have explained in Sections 3 and 4, our configuration choices (for both the indexed content and the retrieval strategy). These choices are supported by experimental results reported in the prior demonstration paper [7], an earlier version of this work [8], and the conference paper [18]. We outline these earlier experimental findings here; they are the basis for the choices made in [18].

The space of choices comprised four main parameters:

A. Linguistic representation model. As described in Section 5.1, STAR allows the user to choose its linguistic representation model in a "plug-and-play" manner. We show in Section 6.5 that PEARL embeddings lead to better performance on our Eurostat benchmark than the more generic SBERT embeddings. The advantages of these neural embedders compared to more traditional approaches have been demonstrated in [7], which evaluated the pioneering Word2Vec [38] embedder against SBERT, on a dataset of INSEE tables. When tested on natural-language queries, SBERT consistently outperforms Word2Vec independently of the other three parameters, for both original questions (using the same terminology as the data) and reformulated ones. For original questions, we observe gains of (at least) 25.7% in Mean Reciprocal Rank@10 (MRR) and (at least) 16.0% in Precision@10; for reformulated questions, the gains reach (at least) 14.5% in MRR@10 and 36.4% in Precision@10. These results confirm that recent transformer-based models better capture the semantics of the facts. Purely syntactic approaches such as BM25 fail to represent the linguistic information of the facts accurately, especially when the vocabulary used in queries diverges from that of the indexed tables.

B. Text representation. As depicted in Section 3.2, the linguistic model does not embed the raw textual content directly, but a *stripped* version, in which spatio-temporal information is removed from $f_{\mathrm{t}}(T) \cup f_{\{\mathrm{h}\}}(T)$. The goal is for embeddings to capture what the facts are about, rather than where and when they hold, since location (and, to a lesser extent, temporal) information can bias semantic similarity. Ideally, a query should match facts on the same subject even if they differ in time or space. This design choice is empirically validated in [7], where indexing the original text s is compared to indexing its stripped version $\bar{s}$, both using Word2Vec. On original questions, indexing $\bar{s}$ instead of s improves MRR@10 by 27.2% and Precision@10 by 93.3%; on reformulated questions, the gains reach 83.3% in MRR@10 and 80.7% in Precision@10. These results highlight the strong impact of separating spatio-temporal information from the textual content used for embedding.

C. Semantic Indexes. As detailed in Section 3.2, STAR relies on two complementary families of indexes: *spatio-temporal indexes*, dedicated to location and time information, and *table representation indexes*, dedicated to the semantic content of tables. The *spatio-temporal indexes* were first explored in [7]. Their construction leverages existing geographic reference systems at different granularities, depending on the characteristics of the indexed datasets. While the geographic references may vary across experimental settings, the underlying principles are the same as in [18]. Regarding the *semantic indexing* of tables, Section 3.1 formalizes several possible table representation functions and motivates the choice of $f_{\mathrm{t}}(T)$ (table titles) and $f_{\{\mathrm{h}\}}(T)$ (set of distinct header strings) as the most semantically informative and robust representations of a table T. Accordingly, two indexes are defined: L_{t} for titles and L_{h} for header strings. The impact of alternative indexing strategies for table representations is experimentally investigated in [7], where three configurations were compared:

1. Three indexes: $f_{\mathrm{t}}(T)$ over the table titles, $f_{\{\mathrm{h}\}}(T)$ over all distinct header strings, and one for table-level comments when available. Note that the comment index was not kept in [18], since comments are often absent from datasets.
2. Three indexes: $f_{\mathrm{t}}(T)$ over the table titles, $f_{\mathrm{ch}}(T)$ over the concatenated column headers, and $f_{\mathrm{rh}}(T)$ over the concatenated row headers.
3. Two indexes: $f_{\mathrm{t}}(T)$ and $f_{\{\mathrm{h}\}}(T)$. This configuration corresponds to the final design adopted (Section 3.1).

Configuration 1 matches the setting of the prior [8] and is used as a baseline for assessing STAR's improvements. It was evaluated using Word2Vec embeddings, augmented with spatio-temporal information extraction. In contrast, configurations 2 and 3 are evaluated with SBERT embeddings; comparisons between 1 and 2–3 therefore reflect both the change in embedding model and the change in index structure. Quantitatively, configurations 2 and 3 consistently outperform the baseline configuration 1. On original questions, switching from configuration 1 to 2 yields a 32.5% increase in MRR@10 and 14.3% in Precision@10, while configuration 3 yields gains of 37.7% and 43.5%, respectively. On reformulated

questions, configuration 2 improves over 1 by 14.9% in MRR@10 and 36.5% in Precision@10, whereas configuration 3 reaches 19.4% and 77.3%. These results, observed consistently across both metrics and both query sets, validate the indexing strategy adopted in Section 3.1. Moreover, the systematic improvements of configuration 3 over 2 highlight the benefit of using the finer-grained header representation $f_{\{\mathrm{h}\}}(T)$ rather than separating concatenated column and row headers ($f_{\mathrm{ch}}(T)$ and $f_{\mathrm{rh}}(T)$). This empirically confirms the design choice discussed in the second and third remarks of Section 3.1.

D. Score aggregator. Since STAR indexes semantic content at the header granularity (in addition to table titles), a question q using L_{t} and L_{h} typically retrieves several textual elements $x_T \in f_{\mathrm{t}}(T) \cup f_{\{\mathrm{h}\}}(T)$ associated with the same table T. As described in Section 4, the final score of a table is obtained by *aggregating* the similarities between the stripped query $\bar{q}$ and each stripped element $\bar{x_T}$. In [7], two aggregation strategies are compared: *sum* and *maximum*. These two strategies are evaluated while keeping all other parameters fixed (SBERT embedder, stripped textual representations, and same indexes). Although the impact of the aggregation parameter is less pronounced than other ones, the sum consistently outperforms the maximum. On original questions, using summation increases MRR@10 by 10.7% and Precision@10 by 14.8%; on reformulated questions, the gains amount to 3.5% in MRR@10 and 12.2% in Precision@10. Overall, these results support the use of sum as the default score aggregator. They empirically confirm the intuition that a table is more likely to be relevant when more than one of its header cell and/or its title match the query, rather than a single (even strong) signal.

6 Experimental Results

We now present our experimental validation of STAR on the STD task. We describe the statistical tables we use (Section 6.1), and our baselines in Section 6.2. We present the evaluation questions and the methodology used to generate them in Section 6.3, then our evaluation metrics in Section 6.4. We discuss experiment results in Section 6.5, before concluding in Section 6.6. Figure 2 presents the end-to-end pipeline of the whole experimental process; we discuss it through our presentation.

6.1 Dataset of Statistical Tables

We use Open Data statistical tables provided by *Eurostat*, the official statistical office of the European Union, for their varied and rich content. We detail the Eurostat table structure to make our discussion self-contained; STAR is not tied to this structure, e.g., [7, 8] use HTML and Excel tables from INSEE, which are organized differently.

Each table has $d \geqslant 1$ dimensions (recall the sample table in Figure 1). The *time* dimension is always present. The *location* dimension is also frequent, unless

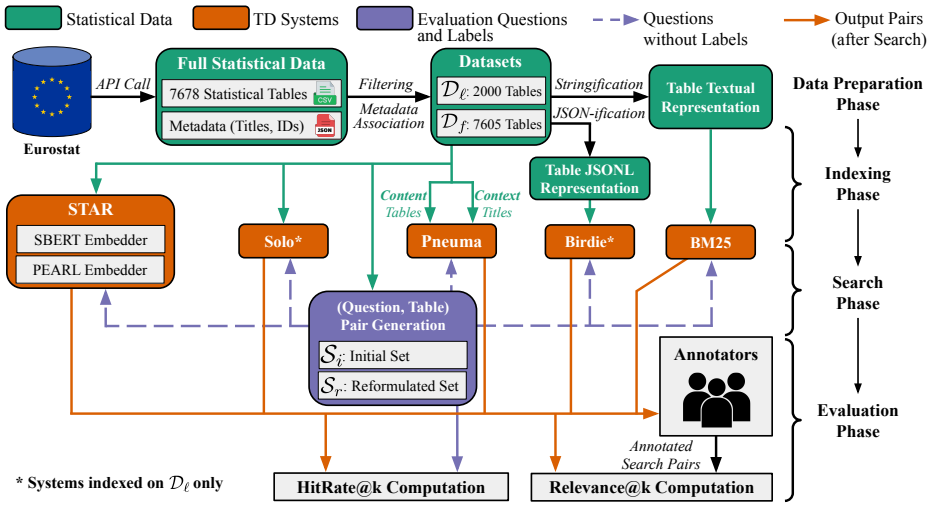


Fig. 2: End-to-end pipeline of the experimental process.

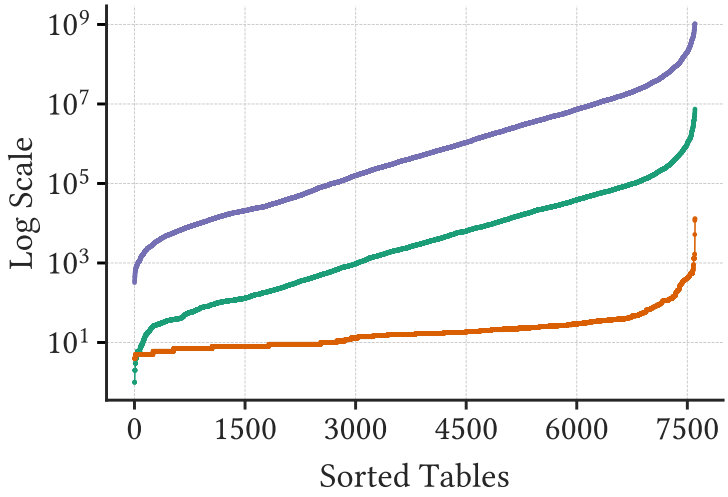


Fig. 3: Rows, columns, and file sizes for $\mathcal{D}_f$.

the table is only about a specific region (e.g., all facts are at the EU scale): in such a case, it is in the title. The remaining $d-1$ dimensions vary depending on the table topic, e.g., demographic attributes (such as age or sex), production types or economic indicators.

To give our TD baselines (see next) the best chance at retrieval, we **transform each statistical table into a relational one** (see Section 2.2). We chose Eurostat tables also because they are *ideally suited* for this transformation: they

come with *all dimension names explicit in the first row*. This gives **optimal chances to each TD baseline**, that has access to a complete, high-quality relational schema, which they require: in other datasets, such as INSEE [7, 8], this is never the case.

As of Jan. 2025, Eurostat's API gave access to 7 678 tables which we retrieved, together with their *metadata* (title and unique identifier). The raw tables encode dimension values using Eurostat-internal *codes*. We reconstructed human-readable tables by mapping the codes into their NL labels using the Eurostat's English code dictionary. We then apply two filtering steps: (i) **Duplicate removal:** Tables with different identifiers but identical content are discarded. (ii) **Size filtering:** Tables exceeding 1GB in size are excluded for practical processing reasons (see Section 6.2). Only 24 such tables were removed.

We thus obtained a cleaned collection of **7 605** distinct statistical tables, denoted $\boxed{\mathcal{D}_f}$. Since some competitors (as we show below in Section 6.5) suffered from Out-Of-Memory (OOM) failure on $\mathcal{D}_f$, we extracted from $\mathcal{D}_f$ its subset $\boxed{\mathcal{D}_\ell}$, consisting of the 2 000 smallest tables (in terms of file size). Table 1 presents the main characteristics of $\mathcal{D}_\ell$ and $\mathcal{D}_f$: the number of tables (#**Tables**), their size in MB (**Size (MB)**), and the mean number of rows and columns per table, with their standard deviation (respectively, #**rows** and #**columns**). Figure 3 shows the distribution of the number of rows, columns, and file sizes (in bytes) in $\mathcal{D}_f$, with the tables sorted in the increasing order of their size. We can see that about 3 000 tables are larger than 1 MB.

Table 1: Main characteristics of the datasets.

Dataset	#Tables	Size (MB)	#rows	#columns
$\mathcal{D}_\ell$	2 000	30.3	111 (± 101)	19 (± 23)
$\mathcal{D}_f$	7 605	98 521.7	69 077 (± 318 706)	41 (± 270)

6.2 Systems

On the STD task, we evaluate STAR against several baseline retrieval models from the literature. Since we are the first to study STD as a particular flavor of table discovery (TD), we consider three categories of retrieval systems: the classical **BM25** which is *lexical-only*, and a variant we call **BM25+Syn** (see below) augmented with synonyms; two very recent TD systems, **Solo** and **Birdie**, based on *language semantics*, both demonstrating performance superior to prior systems (STAR also mostly belongs to this category); and **Pneuma**, another very recent *hybrid* system, leveraging both lexical and semantic table aspects. The various steps involved in the processing are shown in Figure 2. Unless otherwise specified, we used all settings from the original works.

BM25 is a purely lexical retrieval model based on the *Term Frequency–Inverse Document Frequency (TF–IDF)* framework [45]. As it was originally designed for

textual documents rather than tabular data, we convert each table into a textual representation by concatenating:

1. The title of the table;
2. The first row of the table, which contains the $d-1$ non-temporal dimension names and the associated time points;
3. The first $d-1$ columns of the table (excluding the first row already included above), representing all combinations of values for the $d-1$ non-temporal dimensions.

We do not serialize the fact values, since as explained in Section 3, this is not needed for STD. The text obtained as above from each table is indexed in an ElasticSearch [2] engine, which implements the BM25 algorithm. Given the question q, ElasticSearch returns the IDs of the N most pertinent tables.

BM25+Syn is an extension of **BM25** where each query is extended with synonyms of the verbs and nouns it contains (1 synonym per verb or noun). Observe that enriching *the index* by synonyms would bloat it significantly thus adding synonyms to the query is preferable. To give this baseline its best chance, we performed hyper-parameter tuning over three aspects: (i) *Which words to expand via synonyms?* We tested three options: all words; only nouns and verbs; and nouns, verbs, plus adjectives. (ii) *Term weighting.* Since not all words have synonyms (and not the same number), we either *appended synonyms* directly to the query, or *reweighted* all terms so that the original word and its synonyms shared equal weight (e.g., with 3 synonyms, each of these terms and the original word has weight $\frac{1}{4}$). (iii) *Number of synonyms per word.* We tested values from 1 to 10. If a word had less than the target number of synonyms in the thesaurus, we kept all the synonyms. We tested all combinations of the parameters above; when synonyms were restricted to specific word classes, we applied POS tagging with `nltk`'s `averaged_perceptron_tagger_eng` model. We chose the best parameter combination based on the Hit-Rate@10 obtained on our datasets and questions. The best results were obtained with the configuration: *synonyms only for verbs and nouns, appended to the query, with one synonym per word.* We denote this new baseline **BM25+Syn**. Indeed, adding more than one synonym degraded the results. This is expected: the more synonyms added, the more irrelevant noise outweighs potential gains. Interestingly, performance already drops with two synonyms, showing that any benefit from a good synonym is quickly overwhelmed by noise (results are further discussed in Section 6.5).

Solo [50] is a semantic data discovery system that employs a self-supervised method to learn meaningful representations of tabular data. Given a set of CSV tables paired with their titles, Solo embeds and indexes them in a FAISS ANN index [29]. Once the system is trained and the index is built, users can submit natural language questions. Solo retrieves the top-N most relevant tables from FAISS, then it re-ranks them using a pre-trained teacher retrieval model [25]. We use the authors' implementation from [49].

Pneuma [6] leverages LLMs for both table representation and retrieval. It indexes tables based on a combination of their *content* (tabular data) and *context* (title).

During indexing, an LLM generates concise *schema summaries* and *row-level summaries* for a sample of rows from each table. Pneuma uses a *hybrid retrieval strategy*, combining *lexical matching* (via BM25) and *semantic similarity*. These two signals are weighted and combined to get an initial ranking of relevant tables. This ranking is refined by the LLM acting as a *ranking judge*, to give a top-k ranked tables. We use the authors' implementation from [5].

Birdie [22] integrates indexing and search into a single encoder–decoder LM. Unlike systems that separately encode tables into vectors, build an index, and then perform retrieval, Birdie assigns a unique identifier to tables, and uses an LLM to generate synthetic queries for each table. It then trains the encoder–decoder to map (query, table) pairs directly to their table IDs through supervised learning, using structured JSON representations of tables as input. The parameters of the trained model directly embed the data, such that at search time, it encodes NL queries to generate likely table IDs, from which the top N results are selected. We used the authors' implementation [21]. Since the fine-tuned LLaMA3 [37] used for query generation is not provided, we ported the system to OpenAI's GPT-3.5 Turbo [39] instead. Converting the Eurostat CSV tables into the required JSONL format increased their size by approx. 2.75×, resulting in a total size of 267GB.

STAR with SBERT is the system used in [7], to which, *for the experiments in this work* we add: the relaxation (steps (6) to (9)) and the rejection of low-relevance ANN search results, described in Section 4. Experiments in [7] on a smaller, French-language corpus have shown that the representation functions f_{t} and $f_{\{\mathrm{h}\}}$, together, lead to highest-quality results (as discussed in Section 3) thus we also use them here. This system uses the pre-trained SBERT model [42] to represent (embed) linguistic table content (stripped titles and each stripped header cell, as discussed in Section 3). Retrieval is then performed as described in Section 4. We use SpaCy to perform Named-Entity Recognition (NER) for extracting location and time information from both table titles and user questions, QDrant for the ANN indexes L_{t}, L_{h} and Redis for the key-value indexes I_{S}, I_{T}. STAR's parameters had the values we determined empirically to work well: t=80 (Section 3.2), θ=0.6 (limit on the scipy cosine distance), p=0.1 (Section 4).

STAR with PEARL is the same as above, but replacing SBERT with PEARL [12], a more recent embedding model trained with contrastive learning techniques. Unlike SBERT, which is designed for sentence-level semantics, PEARL is optimized for encoding short phrases with little or no context. This makes it particularly well-suited for statistical tables, whose titles and especially headers texts are short.

6.3 Generation of Evaluation Questions

Our evaluation aims to answer the following questions:

Q1. How does each system perform on questions that are syntactically very similar to the tables they refer to?

Q2. How does each system perform on paraphrased questions, with the same meaning but different wording?
Q3. How does each system scale faced with a large number or volume of tables?

Of course, **Q1.** is of academic interest only, since it allows to differentiate syntactic from semantic solutions. In practice, users may formulate things differently, thus **Q2.** is much more important. **Q3.** is of practical interest for real-life statistical tables such as those we use. While Eurostat tables are quite large, other providers also publish large tables, e.g., in tables published by the US Census Bureau, there are on average more than 600 000 cells per table; in the US National Center for Education Statistics, more than 85 000, etc. Thus, scalability is important.

To answer **Q1.** and **Q2.**, we respectively build two sets: an initial question set $\mathcal{S}_i$ (on $\mathcal{D}_\ell$ tables) and a paraphrased one $\mathcal{S}_r$. Since $\mathcal{D}_\ell \subset \mathcal{D}_f$, questions generated on $\mathcal{D}_\ell$ remain relevant for $\mathcal{D}_f$, allowing to study performance as the corpus grows.

$\mathcal{S}_i$ is built semi-automatically, as shown in Figure 4 (a zoom on the purple box on Figure 2). We randomly sample 100 tables from $\mathcal{D}_\ell$, and prompt GPT-4o [40] to generate five English question templates per table using only the table's title, dimension names, and time range. Templates must include placeholders (`[]`) for dimension values, and we provide the model with an example of five illustrative templates. For each template, we instantiate five concrete questions by sampling a row, and replacing the placeholders with its values. A second LLM call then selects, for each table, the single clearest question (and may rewrite it for clarity, without changing the meaning), yielding one final question per sampled table, i.e., 100 questions.

However, automatic generation alone proves insufficient. Many questions are grammatically incorrect or semantically incoherent, e.g., *How many health graduates in Nurses (EU recognised qualification) were there in Iceland during 2017?* (a correct version could be about "nursing graduates"), or *What was the production volume of Products obtained (1 000 t) milk powder in Germany for 2021?* (a quantity wrongly appears in the question). Also, most questions simply replicate terms from the table titles. The core issues we encountered can be summarized as: (i) **LLMs often fail to produce sound, interesting, and complete questions, especially when the data is complex or multi-dimensional.** As noted in the Pneuma paper [6], directly prompting LLMs to ask questions over tables often leads to poor results. Instead, they first generate SQL queries over tables, and then translate them into NL using an LLM. However, this still produces many low-quality questions: on their *Chicago Open* [16] dataset, we find examples such as: *Which address corresponds to the condition that September is 7626?*, *What is the average zip code for the state of IL?*, or *Which gender category is associated with the total of 97 exits?* (from [4]). Many of their generated questions are of this kind: while grammatically correct, they are semantically incoherent or irrelevant, limiting their usefulness for training or evaluation. (ii) **LLM outputs are highly sensitive to prompt wording, which can introduce biases.** This is especially problematic when the generated questions are used to train retrieval models. For example, the prompt used in Birdie [22] for

query generation specifies that the question should include table context. This leads the LLM to insert the full table title into many questions, leading, on our dataset, to outputs like: *What is the average propensity to consume by household type in 2020 based on the table 'Aggregate propensity to consume by household type - experimental statistics'?* or *What is the trend of passenger cars by unloaded weight from 1996 to 2023 in the table 'Passenger cars by unloaded weight'?*. Such questions offer little value for training, as the table needed to answer them is already given, creating a mismatch between training and real-world usage (users typically do not mention table names in their queries).

To mitigate these problems, we manually curate and **sanitize** each question in $\mathcal{S}_i$, ensuring that the questions are interesting, correct, and likely to be asked by a human being.

$\mathcal{S}_r$ is built by manually paraphrasing each question (**reformulation** in Figure 4) in $\mathcal{S}_i$. The goal is to extensively reword each question, so that they largely differ lexically, but still have the exact same semantic meaning. This tests model robustness to lexical variation and shows how over-reliance on cues like table titles can limit generalization and performance.

Together, $\mathcal{S}_i$ and $\mathcal{S}_r$ form two small but high-quality evaluation sets that, to our knowledge, are the first to systematically assess the models' ability to perform over complex and multi-dimensional tables, beyond surface-level matching.

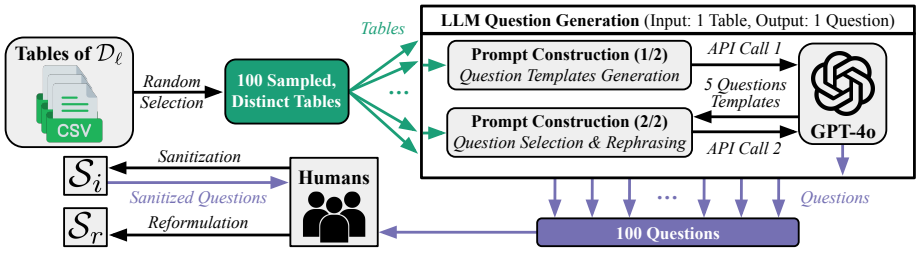


Fig. 4: (Question, Table) pair generation procedure for $\mathcal{S}_i$ and $\mathcal{S}_r$ (zooming in on the purple box of Figure 2).

6.4 Evaluation Metrics

We use two evaluation metrics that capture different aspects of retrieval quality.

HitRate@k measures the proportion of questions for which a system successfully retrieves the table from which the question was originally generated, within its top-k ranked results, where $1 \leqslant k \leqslant 10$. This metric assesses the model's ability to accurately and efficiently locate the relevant table. Varying k gives insight into how quickly the correct table is retrieved: the smaller the value of k at which the hit occurs, the better the model's performance for that metric. HitRate@k

(sometimes called Precision@k) is a standard and widely used metric in table retrieval evaluation [6, 22, 50].

Relevance@k measures a system's ability to return *semantically relevant* tables among its top-k results, where $1 \leqslant k \leqslant 5$. This provides a finer-grained evaluation than HitRate@k, as it captures the relevance of the retrieved tables beyond the "originally intended" table. To compute this score, **human annotators** scored each (question, table) pair returned by any system listed in Section 6.2 with one of three labels: (i) *highly relevant* (score of 2), when the table contains one or more facts that answer the question; (ii) *relevant* (score of 1), when the table does not contain the answer but is about the same or a closely related topic; (iii) *not relevant* (score of 0), when the table is unrelated or only distantly related to the question. For a $k \in \{1, \ldots, 5\}$, we compute the relevance score of the top-k tables retrieved for each question, aggregated over an evaluation set $\mathcal{S}$ and for a system $\mathcal{M}$:

$$\text{Relevance}_k(\mathcal{D}, \mathcal{S}, \mathcal{M}) = \frac{1}{|\mathcal{S}|} \sum_{q \in \mathcal{S}} \sum_{i=1}^{k} \text{Score}\left(\mathcal{M}\left(q, \mathcal{D}\right)^i, q\right)$$

where $\mathcal{M}\left(q, \mathcal{D}\right)^i$ is the i-th table returned by system $\mathcal{M}$ on dataset $\mathcal{D}$ for question q, and $\text{Score}(\cdot, q)$ is the human relevance score for the corresponding (table, question) pair.

6.5 Results and Analysis

All experiments were conducted on a server running AlmaLinux 8.10 with 187 GB of RAM and dual Intel Xeon Gold 5218 CPUs, totaling 32 cores (64 threads). It is equipped with an NVIDIA Tesla V100 PCIe 32GB GPU. Note that STAR does not require a GPU, but the competitors Solo, Pneuma and Birdie do, thus we run all systems on the same machine.

Table 2: Indexing times (in seconds).

Model	$\mathcal{D}_\ell$	$\mathcal{D}_f$
BM25 & BM25+Syn.	13	7 701
Solo	81 847	**OOM**
Pneuma	45 300	532 020
Birdie	24 135	**OOM**
STAR - SBERT	371	45 103
STAR - PEARL	208	44 279

Computational Efficiency Table 2 presents the **data pre-processing and/or indexing times** for all systems, covering all steps until the system is ready to

Table 3: Search times (in seconds), for all 100 questions.

Model	$(\mathcal{D}_\ell, \mathcal{S}_i)$	$(\mathcal{D}_\ell, \mathcal{S}_r)$	$(\mathcal{D}_f, \mathcal{S}_i)$	$(\mathcal{D}_f, \mathcal{S}_r)$
BM25	0.95	0.91	41.83	45.23
BM25+Syn.	3.39	3.34	47.16	51.47
Solo	1 267.05	1 143.06	**OOM**	**OOM**
Pneuma	3 793.23	3 680.90	1 770.44	1 814.24
Birdie	20.83	20.32	**OOM**	**OOM**
STAR - SBERT	36.77	37.66	186.62	190.95
STAR - PEARL	29.68	34.83	106.19	116.68

answer questions, including training where applicable (Solo and Birdie). BM25 is the fastest on both datasets, as a purely lexical retriever. STAR is the second fastest and leads among hybrid and semantic-only retrievers: it is 116–393× faster than others on $\mathcal{D}_\ell$, and over 12× faster than Pneuma on $\mathcal{D}_f$. Solo and Birdie fail to scale to large datasets due to out-of-memory (OOM) errors: Solo, for $\mathcal{D}_\ell$ alone, used more than 43GB of RAM (for only 30.3 MB of raw data), and Birdie ran OOM during the embedding phase for files greater than 200MB. Pneuma, like STAR, avoids RAM issues but requires significantly more time due to its use of an LLM for schema and row summarization.

Table 3 presents **question answering time** for each set of 100 questions and all systems, excluding Solo and Birdie on $\mathcal{D}_f$ due to the noted OOM errors. BM25 remains the fastest, for the same reasons as above. On $\mathcal{D}_\ell$, Birdie is the second fastest, with search times in the same order of magnitude as STAR. Both outperform Pneuma and Solo by 33–180× on $\mathcal{D}_\ell$. As Birdie fails on $\mathcal{D}_f$, STAR becomes the second fastest on it, 15.5× quicker than Pneuma. Using PEARL instead of SBERT in STAR reduces both indexing and QA time.

Retrieval Performance We evaluate all models (excluding those with OOM errors during indexing) on both datasets and question sets. Note that Solo, Pneuma, and Birdie do not always return N (10) results. In Solo, the number of results is impacted by `retr_top_n`, the upper bound on the number of entries retrieved from the index. Birdie, which trains a model to generate (predict) table IDs, sometimes outputs an ID that does not correspond to any table. In our experiments, Solo returned on average 9.91 tables per question, Pneuma 9.58, and Birdie 9.27. STAR returns 10 tables for each question.

HitRates@k. Figures 5 presents the HitRate@k scores ($k \in \{1, \ldots, 10\}$) on $\mathcal{D}_\ell$ and $\mathcal{D}_f$, excluding Solo and Birdie on $\mathcal{D}_f$. We first observe that BM25 and BM25+Syn. perform best on $(\mathcal{D}_\ell, \mathcal{S}_i)$, the most lexical-friendly setting, with near-perfect scores for k=10. However, they perform poorly in all other evaluations, when relevant tables are surrounded by numerous others or when questions are lexically different from table content. This highlights the need for semantic-based retrieval to ensure generalization. Excluding $(\mathcal{D}_\ell, \mathcal{S}_i)$, STAR with PEARL

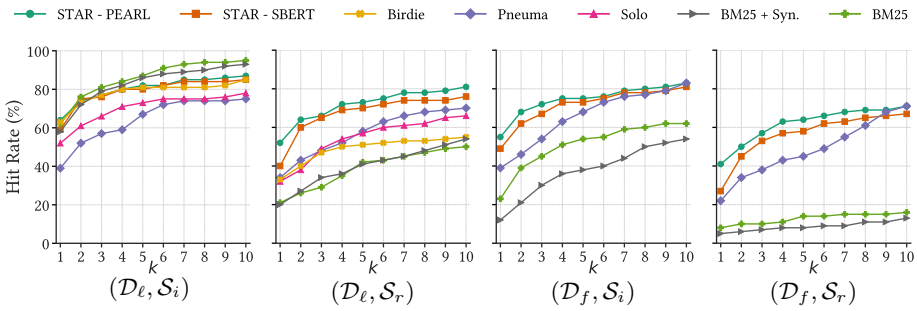


Fig. 5: HitRate@k for $k \in \{1, \ldots, 10\}$ on both datasets for all systems (except Solo and Birdie for $\mathcal{D}_f$).

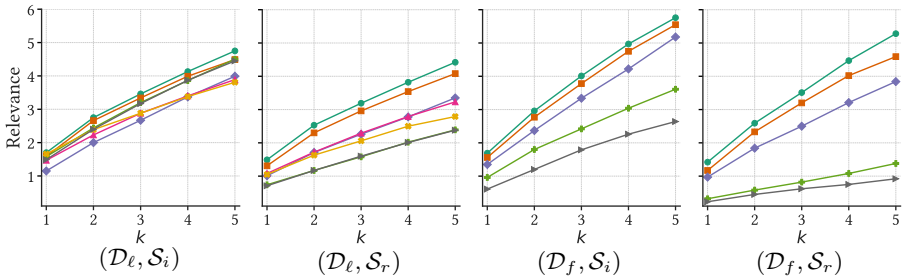


Fig. 6: Relevance@k for $k \in \{1, \ldots, 5\}$ on both datasets for all systems (except Solo and Birdie for $\mathcal{D}_f$).

consistently achieves the best performance across all evaluation sets, slightly outperforming its SBERT variant (ranging from similar results to up to +14%).

To answer **Q1.**, all systems perform well when the questions are lexically close to table content: BM25 and its variant lead, followed by STAR and Birdie. Solo and Pneuma, while being 10% behind for most k values, still deliver reasonable performance. **Q2.** reveals the limitations of lexical models: BM25 suffers a large performance drop (-45% for k=10 from $(\mathcal{D}_\ell, \mathcal{S}_i)$ to $(\mathcal{D}_\ell, \mathcal{S}_r)$), as expected. BM25+Syn. is slightly better than BM25 and competes with Birdie at k=10, showing relevance of synonym expansion in some cases. Birdie performance degrades significantly (-30%), reflecting its training bias toward LLM-generated questions that often include full table titles (recall Section 6.2). But $\mathcal{S}_r$ questions do not repeat table titles. On $(\mathcal{D}_\ell, \mathcal{S}_r)$, STAR with PEARL is best, outperforming Pneuma by 11% and Solo by 15% at k=10. STAR achieves strong performance from the top of the ranking: both PEARL and SBERT variants exceed 60% by k=2, whereas others reach it only from k=6.

Regarding **Q3**, STAR (with both PEARL and SBERT) and Pneuma adapt well to larger datasets, Pneuma even improving from $(\mathcal{D}_\ell, \mathcal{S}_i)$ to $(\mathcal{D}_f, \mathcal{S}_i)$. While they reach comparable performance at k=10, STAR obtains strong results quicker: for instance, on $(\mathcal{D}_f, \mathcal{S}_i)$, STAR with PEARL exceeds 70% at k=3, while Pneuma requires k=6. In contrast, BM25 fails to scale effectively, dropping by 33% at k=10 from $(\mathcal{D}_\ell, \mathcal{S}_i)$ to $(\mathcal{D}_f, \mathcal{S}_i)$. Except on $(\mathcal{D}_\ell, \mathcal{S}_i)$, and on Hit-Rate only, BM25+Syn. performs significantly worse than BM25:

- For questions in $\mathcal{S}_i$, at least one relevant table contains the exact terminology of the query (the golden one). Synonyms therefore add only noise, with no improvement in Hit-Rate and Relevance.
- For questions in $\mathcal{S}_r$, synonyms could in theory help. However, this only works if the expanded terms match the terminology in the relevant tables as opposed to introducing more noise; success here is essentially a matter of luck (depending on whether the reformulated term corresponds exactly to one of the synonyms). Identifying truly useful synonyms would require semantic understanding, which lexical methods lack by definition.

Thus, while adding synonyms to questions could seem a good idea to enhance BM25, it turns out not to help in practice, due to the "noise" added by the synonyms that did not exactly match the tables.

Relevance@k. Figure 6 reports Relevance@k ($k \in \{1, \ldots, 5\}$) on $\mathcal{D}_\ell$ and $\mathcal{D}_f$, excluding Solo and Birdie on $\mathcal{D}_f$. To compute Relevance@k scores, we manually annotated the top-5 retrieved table produced by each model, for both datasets and questions sets. In total, **4 963 (question, table) pairs were labeled by 9 human annotators**. To assess annotation consistency, 220 pairs (approx. 5%) were randomly selected for double annotation: 24 received differing labels, which shows that annotators agreed **89.1%** of the time, which is considered very high. In cases of disagreement, in the gold standard, we kept the more favorable score.

The trends observed for HitRate@k largely hold for Relevance@k, with a few exceptions. On $(\mathcal{D}_\ell, \mathcal{S}_i)$, BM25 (+ Syn.) no longer lead: STAR with PEARL

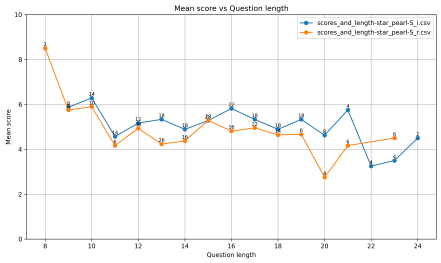
achieves the highest relevance scores, although BM25 (+ Syn.) remain close. This further shows BM25's inability to handle lexical variations. On the other evaluation settings ($(\mathcal{D}_\ell, \mathcal{S}_r)$, $(\mathcal{D}_f, \mathcal{S}_i)$, and $(\mathcal{D}_f, \mathcal{S}_r)$), STAR with both PEARL and SBERT outperforms all other models, with Pneuma usually following at a distance of less than half a relevance point. Interestingly, while STAR models underwent a performance drop from $\mathcal{D}_\ell$ to $\mathcal{D}_f$ on HitRate@k, they improve on Relevance@k. This suggests that, although they sometimes fail to retrieve the table the question was generated from, they still succeed in retrieving semantically relevant alternatives.

Replacing SBERT with PEARL improves Relevance@k across all four configurations. However, in some cases, we note that STAR with PEARL underperforms. The worst-case occurs for the question *"What was the percentage of GDP allocated to private sector credit flow in non-financial corporations for Slovakia in 2021?"* and its reformulation *"In 2021, what portion of the GDP was allocated to credit flow for private organizations that are not involved in finance in Slovakia?"*, where STAR with SBERT exceeds STAR with PEARL by 4 points on Relevance@5. Similarly, for the question *"What fraction of the population in Latvia lacked sanitary amenities in 2011?"* on $(\mathcal{D}_\ell, \mathcal{S}_r)$, the difference is 3 points. These examples might suggest that STAR with PEARL underperforms on long or complex questions, since PEARL is trained on short, context-free phrases. Yet, such a conclusion is not supported. Indeed, we encountered complex questions where STAR with PEARL is the only method to perform well, e.g. the first question analyzed in Section 6.5. Further supporting this, Figure 7 plots, for each model and dataset, the mean Relevance@5 as a function of the number of words in the query, with the numbers above points indicating the number of questions with that score. The trends for STAR with PEARL align with those of other models, rejecting the hypothesis that it struggles with longer or more complex queries.

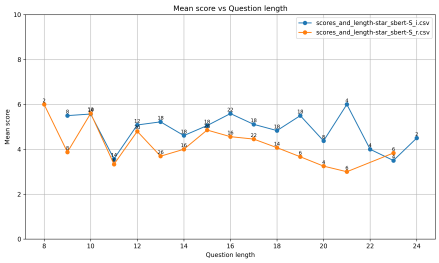
On both quality metrics considered, STAR outperforms all competitors, except BM25 (+ Syn.) which perform better only when questions are phrased almost identically to the data. Even when its performance is close to that of competitors (notably Pneuma), STAR remains faster in both indexing and search, making it the more practical option overall.

Qualitative Analysis We demonstrate STAR's relevance by analyzing outputs for two queries over $\mathcal{D}_\ell$ and $\mathcal{D}_f$.

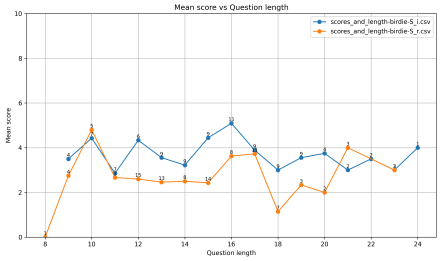
On $\mathcal{D}_\ell$, we consider: "In 2020, what was the median saving rate of employed persons in Bulgaria as a percentage of disposable income?" ($\mathcal{S}_i$) and "In 2020, what was the median rate of savings among people in employment in Bulgaria, as a percentage of available revenue?" ($\mathcal{S}_r$). The question is complex, using economic and demographic information. Lexical baselines perform well on $(\mathcal{D}_\ell, \mathcal{S}_i)$, e.g., BM25 achieves Relevance@5 (R@5) of 6 via exact overlaps: titles contain "Median saving rate", and tables "Percentage of disposable income" or "Employed persons". On $\mathcal{S}_r$, R@5 drops to 0, as only generic words like "median" and "percentage" remain (other new terms do not match relevant tables), retrieving off-topic tables (violence statistics, minimum wage, higher-education researchers). TD systems



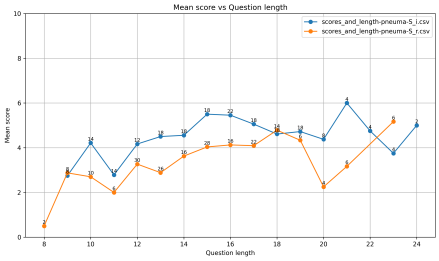
(a) STAR - PEARL



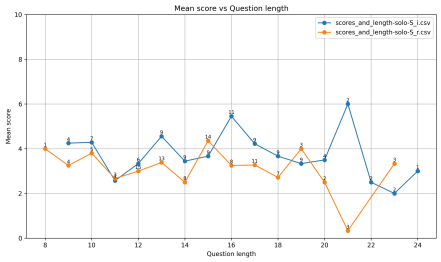
(b) STAR - SBERT



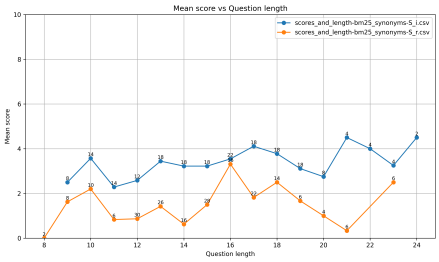
(c) Birdie



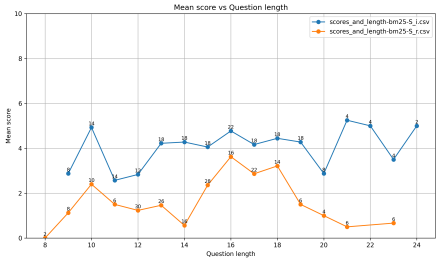
(d) Pneuma



(e) Solo



(f) BM25 + Syn.



(g) BM25

Fig. 7: Mean Relevance@5 score per question length (number of words), for all systems, on both $\mathcal{D}_\ell$ and $\mathcal{D}_f$. Numbers above each point denote the number of questions corresponding to that question length.

capture partial semantics but fail to cover the full meaning of the query (R@5 of 0 or 1 on $\mathcal{S}_i$ and $\mathcal{S}_r$). Solo returns tables with overlapping notions (saving rate, employment, population), showing partial understanding. Pneuma's titles are barely related (except "rate" and "people"), though content includes income, percentage, and people. Outputs are nearly identical between $\mathcal{S}_i$ and $\mathcal{S}_r$, showing good semantic proximity detection but poor weighting. Birdie outputs tables with titles about saving rate, employment, or compensation (with content partially matching the questions), but no tables connecting the notions, thus results irrelevant for the query. In contrast, STAR–PEARL performs well: for $\mathcal{S}_i$ it overlaps with BM25 on 4 tables, replacing the irrelevant one. For $\mathcal{S}_r$, predictions remain identical (showing strong semantic proximity), yielding R@5 of 7 for both. STAR–SBERT is identical on $\mathcal{S}_i$, but drops on $\mathcal{S}_r$ to a R@5 of 2, keeping only 2 tables: new ones are about minimum wage or unemployment rates, lacking core notions.

On $\mathcal{D}_f$, we study the (much simpler) query "How many deaths were recorded in Belgium in 2021?" ($\mathcal{S}_i$) and "How many people died in Belgium in 2021?" ($\mathcal{S}_r$). The quality of results returned by lexical baselines collapses: BM25 retrieves irrelevant tables (innovation surveys, ICT impact, financing difficulties), leading to a R@5 of 0 on $\mathcal{S}_i$ and $\mathcal{S}_r$. In details: 1. On $\mathcal{S}_i$, the *titles* of the top-5 tables returned by BM25 contain *no word from the query*. The *contents* of the top-5 returned tables have thousands of occurrences of the words "were", "recorded", and "Belgium"; the tables themselves are irrelevant. These frequent words probably lead BM25 to miss relevant tables even those containing "death" exactly like the simpler query above. Note that removing duplicates in table headers is not an option because word frequencies are needed by methods such as BM25. 2. On $\mathcal{S}_r$, the *titles* of BM25's results do not match the questions at all, whereas their *contents* have many occurrences of "people" and "many".

On the same question, Pneuma also performs poorly (R@5 of 1 on $\mathcal{S}_i$ and $\mathcal{S}_r$); it returns tables about Belgian railway traffic. In contrast, STAR separately looks up the meaningful metric (here, death), thanks to the separate indexing of location and time. One Pneuma result partially fits (transport-accident deaths), but lacks the geographic restriction.

In contrast, STAR–PEARL excels (R@5 of 9 on $\mathcal{S}_i$ and $\mathcal{S}_r$), retrieving only highly relevant mortality tables (overall deaths, deaths by sex or cause), that matches location and time constraints. STAR–SBERT follows closely (R@5 of 8 on $\mathcal{S}_i$, 7 on $\mathcal{S}_r$); on $\mathcal{S}_i$ its result set is almost identical to that of STAR–PEARL, while on $\mathcal{S}_r$ performance slightly drops, retrieving road accident fatalities or prison capacity data.

6.6 Conclusion

Our experiments demonstrate that STAR outperforms both semantic and hybrid models. It is outperformed by BM25+Syn. on $(\mathcal{D}_\ell, \mathcal{S}_i)$ and Hit-Rate only, when questions match the terminology used in the tables; however, this method collapses on a larger dataset and reformulated questions. In contrast, STAR returns the most semantically pertinent results, with significantly lower indexing and search

times than those of semantic competitors, no OOM errors, and robust scalability to large datasets. STAR does not require a GPU, nor does it rely on costly and potentially slow calls to remote LLMs, making it a robust and resource-efficient solution.

7 Related Work

Several problems previously studied for *relational (not multidimensional) tables* are related to the STD problem we introduced in this work.

The closest is **table discovery (TD)**, which seeks to find the relational table(s) most relevant to answer user questions. It is a subtopic of dataset discovery [41], where datasets are tables. [54] builds a set of representations, syntactic (or statistic) and semantic (based on word embeddings) for queries and tables, and combines with other features into a *learning-to-rank* framework. Semantic representations improve retrieval quality, compared to only using statistical methods (such as BM25), which fails to detect *synonyms* or semantically close words. In [48], the authors transform the query, and each table (title and all headers) into a single string, then use BM25 to retrieve the most relevant tables. To rank them, they introduce a set of non-neural and neural features, representing the query and the table based on RNNs; weight features are learned via a forest of decision trees. In [44], several *modalities* of each table (e.g., description, schema, each row, column) are independently embedded, then a joint representation thereof is learned using Gated Multimodal Units. LLMs use for TD problem is pioneered in [14], which uses BERT. Because only *limited-length strings* can be embedded, their *content selectors* extract, from each table, bounded-length fragments to represent it. On a very large table, this still fails to represent *most* of the content, leading to potentially many missed results (recall the "industry in England" example using f_{t} in Section 3.1). Also, this method needs to *embed each table jointly with the query at query time*, which can be prohibitively slow for datasets of many and/or large tables. The retrieval approach [23] builds a table representation as in [24], a similar one for the query, and retrieves the tables closest to the query.

More recently, such works have been outperformed by the recent systems Solo [50], Pneuma [6], and Birdie [22] which we presented in Section 6.2, and that STAR outperforms on statistical tables. **STAR's advantage** is due to: (i) its built-in understanding of statistical tables, which allows it to index only the tables' linguistic content; (ii) its strategy of representing a table by its title together with each non-empty stripped header cell; (iii) its separate indexing and treatment of location and time values, universal dimensions of any measured human or natural activity (thus, of any statistical dataset).

NLP question answering over a table (TQA) has been implemented by translating the question into a SQL query, answered by a database holding the tables (see the survey [32]). A text-to-SQL quality benchmark was introduced in [33] and updated since; as of June 2025, it shows an F1 score of 73.84% for the best-performing model, while humans score 92.96%. Others apply neural methods

directly on the table, e.g., TaPaS [24]. More recent works, e.g., Binder [53], Dater [15], or [51, 55] represent tables as strings, given as inputs to an LLM prompted to return the results from the table. For large tables, this is not feasible. Thus, authors in [34, 47] extract, from large tables, subtables that fit in an LLM's context, but they still face performance challenges for large tables, and search precision may suffer. TableRAG [13] starts with a *schema retrieval* step matching the question against the database schema (table and attribute names), complete with synthetic information about the values in each column, e.g., the minimum, maximum, and a few sample values. Then, *cell retrieval* narrows down on the cells (in various columns) that are needed in order to compute the question answers. Finally, TableRAG leverages the ReAct [52] framework, allowing an LLM to interact in a chain-of-thought fashion with tools such as Python, to compute the query answers. ReAcTable [56] also builds upon ReAct. It answers complex question over a given table, combining LLMs, Python, and SQL tools. Both TableRAG and ReAcTable surpass Binder, Dater and other prior works; TableRAG can handle very large tables, using a fixed budget of tokens of interaction with the LLM.

End-to-end QA systems aim to answer a question, given a large set of tables. This is typically implemented by a first TD (or *retrieval*) step, which restricts the scope of the search, followed by *(re)-ranking*, with a higher-quality but more expensive model. As an example, [23] uses a reader model to extract the answer for a given question using previously retrieved candidate tables. The model scores each candidate, and extracts a suitable answer span from the table. Tables and questions are jointly encoded using TaPaS [24]. Scrutinizer [30, 31] helps human users fact-check *statistic claims*, e.g., "Covid killed more people in France than in Italy", over relational tables, by interactively composing SQL queries. More generally, many end-to-end QA solutions can be obtained by combining a TD method and one for QA.

All the works mentioned above target relational tables, not multidimensional ones. As described in Section 2.2 and experimentally shown, even after converting statistical tables into relational ones, existing TD systems are either unable to handle large tables, or outperformed by STAR in terms of efficiency and/or result quality.

For the **STD** task, we are only aware of prior versions of StatCheck. In [10], individual facts were indexed by the values of their dimensions, and stored in an RDF database; this did not scale to datasets considered in this work. [8, 9] improve over [10] by indexing at the granularity of tables and a more efficient storage of statistics; the demonstration [7] improves result quality over [9] by adopting SBert, and indexing time and locations separately. The present work improves over [7] by: (i) pruning low-relevance results and relaxing queries; (ii) replacing SBert with PEARL embeddings. Also, we present a *novel and extensive experimental comparison* with recent, high-performing TD systems, which demonstrates STAR's superiority in both efficiency and result quality, and provide a large, novel benchmark.

8 Conclusion

We have formalized the *Statistical Table Discovery* (STD) problem and introduced STAR, a *Space and Time-Aware Statistic Retrieval* method tailored to statistical data. STAR combines structured indexing over spatial and temporal dimensions with semantic embeddings of tabular text, leveraging the PEARL model for improved phrase matching. Relevance is scored along space, time, and query semantics.

Experiments on a large Eurostat dataset show that STAR consistently outperforms prior *Table Discovery* systems as well as simpler methods based on BM25, especially when questions do not use the exact words present in the data. We show this on a novel benchmark of curated and reformulated queries, with corresponding relevance scores of tables. Competing systems either fail (OOM) on large datasets or degrade in quality on lexically diverse queries. Even when close in accuracy, they are at least 12× slower. Unlike LLM-based systems, STAR requires no GPU and has a lesser computational cost.

Several directions remain open for future work. First, *user feedback*: relevance judgments collected during use could be fed back into the retrieval model to refine its ranking over time, providing a natural mechanism to adapt STAR to a specific corpus or user community. Second, *full statistical question answering.* STAR already supports a best-effort form of TQA: when the answer exists as a directly addressable fact in the retrieved table, it can be obtained by intersecting the relevant row and column headers (Section 4). What remains open is the harder case where the answer does not exist as a single cell: for instance because it requires aggregating facts across dimensions, computing derived quantities such as rates, ratios, or differences, or combining information from several tables. These operations are common in practice: a user asking for the share of youth unemployment in total unemployment, or the evolution of a metric across two datasets from different providers, expects a computed answer rather than a pointer to a table. A promising direction here is to use STAR's retrieval output (a small, structured set of relevant facts with their spatio-temporal context) as the input to a small, locally run LLM tasked with performing the remaining computation. Recent work shows that accurate TQA is achievable with open-weight models running on a standard laptop [28], and that inference cost can be controlled by routing questions adaptively between cheaper and more expensive computation [36]. For statistical data, where privacy and offline operation matter, this pipeline would preserve STAR's scalability while extending its reach to genuinely complex queries. Third, *multilingual and cross-provider evaluation.* Since STAR's core retrieval logic contains no trainable parameters, replacing the embedding model requires no algorithmic changes, only re-indexing the corpus with the new encoder. Preliminary experiments on French INSEE datasets confirm this portability [7], and a systematic multilingual benchmark covering providers with varying metadata quality would consolidate this architectural claim.

Acknowledgments

This work was funded in part by the French government under management of Agence Nationale de la Recherche (ANR) as part of the "France 2030" program, references ANR-23-IACL-0008 (PR[AI]RIE-PSAI) and ANR-23-IACL-0005 (Hi! PARIS Cluster 2030), by the ANR and DGA under reference ANR-20-CHIA-0015, and by the Inria Action Exploratoire JoDaIA. It was also supported by the Portuguese government through Fundação para a Ciência e a Tecnologia, I.P. (FCT) under projects UID/50021/2025 and UID/PRR/50021/2025.

Disclosure of interests

The authors have no competing interests to declare that are relevant to the content of this article.

References

1. COG. https://www.insee.fr/fr/information/7766585 (2024)
2. Elasticsearch. https://www.elastic.co/ (2024)
3. NUTS. https://ec.europa.eu/eurostat/documents/345175/629341/NUTS2021-NUTS2024.xlsx/2b35915f-9c14-6841-8197-353408c4522d?t=1702990824080 (2024)
4. Balaka, M.I.L., Alexander, D., Wang, Q., Gong, Y., Krisnadhi, A., Fernandez, R.C.: Pneuma's LLM-generated questions on the Chicago dataset. https://storage.googleapis.com/pneuma_open/pneuma_chicago_10K_questions_annotated.jsonl
5. Balaka, M.I.L., Alexander, D., Wang, Q., Gong, Y., Krisnadhi, A., Fernandez, R.C.: Pneuma code repository. https://github.com/TheDataStation/pneuma (2025)
6. Balaka, M.I.L., Alexander, D., Wang, Q., Gong, Y., Krisnadhi, A., Fernandez, R.C.: Pneuma: Leveraging LLMs for tabular data representation and retrieval. Proc. ACM Manag. Data **3**(3), 200:1–200:28 (2025)
7. Balalau, O., Ebel, S., Galhardas, H., Galizzi, T., Manolescu, I.: STaR: Space and Time-aware Statistic Query Answering. In: ACM Conference on Information and Knowledge Management (CIKM). Boise, Idaho, United States (2024). https://doi.org/10.1145/3627673.3679209, https://hal.science/hal-04689206
8. Balalau, O., Ebel, S., Galizzi, T., Manolescu, I., Massonnat, Q., Deiana, A., Gautreau, E., Krempf, A., Pontillon, T., Roux, G., Yakin, J.: Fact-checking multidimensional statistic claims in french. In: Truth and Trust Online Conference (2022), https://truthandtrustonline.com/wp-content/uploads/2022/10/TTO_2022_paper_4.pdf
9. Balalau, O., Ebel, S., Galizzi, T., Manolescu, I., Massonnat, Q., Deiana, A., Gautreau, E., Krempf, A., Pontillon, T., Roux, G., Yakin, J.: Statistical claim checking: Statcheck in action. In: Hasan, M.A., Xiong, L. (eds.) Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022. pp. 4798–4802. ACM (2022). https://doi.org/10.1145/3511808.3557198, https://doi.org/10.1145/3511808.3557198

10. Cao, T.D., Manolescu, I., Tannier, X.: Searching for truth in a database of statistics. In: Proceedings of the 21st International Workshop on the Web and Databases, Houston, TX, USA, June 10, 2018. pp. 4:1–4:6. ACM (2018). https://doi.org/10.1145/3201463.3201467, https://doi.org/10.1145/3201463.3201467
11. Ceurstemont, S.: Controlling AI's growing energy needs. Commun. ACM **68**(3), 15–17 (2025). https://doi.org/10.1145/3703788, https://doi.org/10.1145/3703788
12. Chen, L., Varoquaux, G., Suchanek, F.: Learning high-quality and general-purpose phrase representations. In: Graham, Y., Purver, M. (eds.) Findings of the Association for Computational Linguistics: EACL 2024. pp. 983–994. Association for Computational Linguistics, St. Julian's, Malta (2024), https://aclanthology.org/2024.findings-eacl.66
13. Chen, S., Miculicich, L., Eisenschlos, J., Wang, Z., Wang, Z., Chen, Y., Fujii, Y., Lin, H., Lee, C., Pfister, T.: Tablerag: Million-token table understanding with language models. In: Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J.M., Zhang, C. (eds.) Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024 (2024), http://papers.nips.cc/paper_files/paper/2024/hash/88dd7aa6979e352fda7c4952ca8eac59-Abstract-Conference.html
14. Chen, Z., Trabelsi, M., Heflin, J., Xu, Y., Davison, B.D.: Table search using a deep contextualized language model. In: Huang, J., Chang, Y., Cheng, X., Kamps, J., Murdock, V., Wen, J., Liu, Y. (eds.) Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020. pp. 589–598. ACM (2020). https://doi.org/10.1145/3397271.3401044, https://doi.org/10.1145/3397271.3401044
15. Cheng, Z., Xie, T., Shi, P., Li, C., Nadkarni, R., Hu, Y., Xiong, C., Radev, D., Ostendorf, M., Zettlemoyer, L., Smith, N.A., Yu, T.: Binding language models in symbolic languages. In: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net (2023), https://openreview.net/pdf?id=lH1PV42cbF
16. City of Chicago: Chicago data portal. https://data.cityofchicago.org/
17. Freire, J., Fan, G., Feuer, B., Koutras, C., Liu, Y., Peña, E., Santos, A.S.R., Silva, C.T., Wu, E.: Large language models for data discovery and integration: Challenges and opportunities. IEEE Data Eng. Bull. **49**(1), 3–31 (2025), http://sites.computer.org/debull/A25mar/p3.pdf
18. Gauquier, A., Ebel, S., Galhardas, H., Galizzi, T., Manolescu, I., Peden, A., Senellart, P.: Efficient and Scalable Search for Statistics. In: Proc. of the 42nd IEEE International Conference on Data Engineering, ICDE 2026. Montréal, Canada (May 2026)
19. Gauquier, A., Ebel, S., Galhardas, H., Galizzi, T., Manolescu, I., Peden, A., Senellart, P.: Code of the STAR–StatCheck system. https://gitlab.inria.fr/cedar/star-statcheck (2025)
20. Gauquier, A., Ebel, S., Galhardas, H., Galizzi, T., Manolescu, I., Peden, A., Senellart, P.: Supplementary material for the paper Efficient And Scalable Search For Statistics. https://github.com/AntoineGauquier/efficient_and_scalable_search_for_statistics (2025)
21. Guo, Y., Hu, Z., Mao, Y., Zheng, B., Gao, Y., Zhou, M.: Birdie code repository. https://github.com/ZJU-DAILY/BIRDIE (2025)

22. Guo, Y., Hu, Z., Mao, Y., Zheng, B., Gao, Y., Zhou, M.: Birdie: Natural language-driven table discovery using differentiable search index. ArXiv preprint **abs/2504.21282** (2025), https://arxiv.org/abs/2504.21282
23. Herzig, J., Müller, T., Krichene, S., Eisenschlos, J.: Open domain question answering over tables via dense retrieval. In: Toutanova, K., Rumshisky, A., Zettlemoyer, L., Hakkani-Tur, D., Beltagy, I., Bethard, S., Cotterell, R., Chakraborty, T., Zhou, Y. (eds.) Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 512–519. Association for Computational Linguistics, Online (2021). https://doi.org/10.18653/v1/2021.naacl-main.43, https://aclanthology.org/2021.naacl-main.43
24. Herzig, J., Nowak, P.K., Müller, T., Piccinno, F., Eisenschlos, J.: TaPas: Weakly supervised table parsing via pre-training. In: Jurafsky, D., Chai, J., Schluter, N., Tetreault, J. (eds.) Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 4320–4333. Association for Computational Linguistics, Online (2020). https://doi.org/10.18653/v1/2020.acl-main.398, https://aclanthology.org/2020.acl-main.398
25. Izacard, G., Grave, E.: Distilling knowledge from reader to retriever for question answering. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net (2021), https://openreview.net/forum?id=NTEz-6wysdb
26. Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: Fundamentals of Data Warehouses. Springer (2001)
27. Jensen, C., Snodgrass, R.: Temporal data management. IEEE TKDE **11**(1) (1999). https://doi.org/10.1109/69.755613
28. Jiang, Y., Wei, F., Bao, E., Li, Y., Ding, B., Yang, Y., Xiao, X.: Accurate Table Question Answering with Accessible LLMs. In: Proceedings of the 42nd IEEE International Conference on Data Engineering, ICDE 2026 (2026)
29. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with GPUs. IEEE Transactions on Big Data **7**(3), 535–547 (2019)
30. Karagiannis, G., Saeed, M., Papotti, P., Trummer, I.: Scrutinizer: A mixed-initiative approach to large-scale, data-driven claim verification. Proc. VLDB Endow. **13**(11), 2508–2521 (2020), http://www.vldb.org/pvldb/vol13/p2508-karagiannis.pdf
31. Karagiannis, G., Saeed, M., Papotti, P., Trummer, I.: Scrutinizer: Fact checking statistical claims. Proc. VLDB Endow. **13**(12), 2965–2968 (2020). https://doi.org/10.14778/3415478.3415520, http://www.vldb.org/pvldb/vol13/p2965-karagiannis.pdf
32. Katsogiannis-Meimarakis, G., Koutrika, G.: A survey on deep learning approaches for text-to-SQL. VLDB J. **32**(4) (2023). https://doi.org/10.1007/S00778-022-00776-8, https://doi.org/10.1007/s00778-022-00776-8
33. Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K.C., Huang, F., Cheng, R., Li, Y.: Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023 (2023), http://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html

34. Lin, W., Blloshmi, R., Byrne, B., de Gispert, A., Iglesias, G.: An inner table retriever for robust table question answering. In: Rogers, A., Boyd-Graber, J., Okazaki, N. (eds.) Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 9909–9926. Association for Computational Linguistics, Toronto, Canada (2023). https://doi.org/10.18653/v1/2023.acl-long.551, https://aclanthology.org/2023.acl-long.551
35. Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the middle: How language models use long contexts. Transactions of the Association for Computational Linguistics **12**, 157–173 (2024). https://doi.org/10.1162/tacl_a_00638, https://aclanthology.org/2024.tacl-1.9
36. Liu, Y., Yan, M., Xue, J., Ren, W., Ye, Y., Zhou, H., Chen, Z., Li, J.: SPARQ: A Cost-Efficient Framework for Offline Table Question Answering via Adaptive Routing. In: Proceedings of the 42nd IEEE International Conference on Data Engineering, ICDE 2026 (2026)
37. Meta AI: Introducing Meta Llama 3: The most capable openly available LLM to date. https://ai.meta.com/blog/meta-llama-3/ (2024)
38. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: Bengio, Y., LeCun, Y. (eds.) 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings (2013)
39. OpenAI: Documentation of the GPT 3.5 Turbo model. https://platform.openai.com/docs/models/gpt-3.5-turbo
40. OpenAI: Documentation of the GPT 4 "Omni" model. https://platform.openai.com/docs/models/gpt-4o
41. Paton, N.W., Chen, J., Wu, Z.: Dataset discovery and exploration: A survey. ACM Comput. Surv. **56**(4), 102:1–102:37 (2024). https://doi.org/10.1145/3626521, https://doi.org/10.1145/3626521
42. Reimers, N., Gurevych, I.: Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In: Inui, K., Jiang, J., Ng, V., Wan, X. (eds.) Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). pp. 3982–3992. Association for Computational Linguistics, Hong Kong, China (2019). https://doi.org/10.18653/v1/D19-1410, https://aclanthology.org/D19-1410
43. SDMX Community: Sdmx technical specifications. https://sdmx.org/?page_id=5008 (2024)
44. Shraga, R., Roitman, H., Feigenblat, G., Canim, M.: Web table retrieval using multimodal deep learning. In: Huang, J., Chang, Y., Cheng, X., Kamps, J., Murdock, V., Wen, J., Liu, Y. (eds.) Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020. pp. 1399–1408. ACM (2020). https://doi.org/10.1145/3397271.3401120, https://doi.org/10.1145/3397271.3401120
45. Sparck Jones, K.: A statistical interpretation of term specificity and its application in retrieval. Journal of documentation **28**(1), 11–21 (1972)
46. Strötgen, J., Gertz, M.: HeidelTime: High quality rule-based extraction and normalization of temporal expressions. In: Erk, K., Strapparava, C. (eds.) Proceedings of the 5th International Workshop on Semantic Evaluation. pp. 321–324. Association for Computational Linguistics, Uppsala, Sweden (2010), https://aclanthology.org/S10-1071

47. Sui, Y., Zou, J., Zhou, M., He, X., Du, L., Han, S., Zhang, D.: TAP4LLM: table provider on sampling, augmenting, and packing semi-structured data for large language model reasoning. ArXiv preprint **abs/2312.09039** (2023), https://arxiv.org/abs/2312.09039
48. Sun, Y., Yan, Z., Tang, D., Duan, N., Qin, B.: Content-based table retrieval for web queries. Neurocomput. **349**(C), 183–189 (2019). https://doi.org/10.1016/j.neucom.2018.10.033, https://doi.org/10.1016/j.neucom.2018.10.033
49. Wang, Q., Fernandez, R.C.: Solo code repository. https://github.com/thedatastation/solo (2023)
50. Wang, Q., Fernandez, R.C.: Solo: Data discovery using natural language questions via A self-supervised approach. Proc. ACM Manag. Data **1**(4), 262:1–262:27 (2023). https://doi.org/10.1145/3626756, https://doi.org/10.1145/3626756
51. Wang, Z., Zhang, H., Li, C., Eisenschlos, J.M., Perot, V., Wang, Z., Miculicich, L., Fujii, Y., Shang, J., Lee, C., Pfister, T.: Chain-of-table: Evolving tables in the reasoning chain for table understanding. In: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net (2024), https://openreview.net/forum?id=4L0xnS4GQM
52. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K.R., Cao, Y.: React: Synergizing reasoning and acting in language models. In: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net (2023), https://openreview.net/pdf?id=WE_vluYUL-X
53. Ye, Y., Hui, B., Yang, M., Li, B., Huang, F., Li, Y.: Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In: Chen, H., Duh, W.E., Huang, H., Kato, M.P., Mothe, J., Poblete, B. (eds.) Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2023, Taipei, Taiwan, July 23-27, 2023. pp. 174–184. ACM (2023). https://doi.org/10.1145/3539618.3591708, https://doi.org/10.1145/3539618.3591708
54. Zhang, S., Balog, K.: Ad hoc table retrieval using semantic similarity. In: Champin, P., Gandon, F.L., Lalmas, M., Ipeirotis, P.G. (eds.) Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018. pp. 1553–1562. ACM (2018). https://doi.org/10.1145/3178876.3186067, https://doi.org/10.1145/3178876.3186067
55. Zhang, T., Yue, X., Li, Y., Sun, H.: TableLlama: Towards open large generalist models for tables. In: Duh, K., Gomez, H., Bethard, S. (eds.) Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers). pp. 6024–6044. Association for Computational Linguistics, Mexico City, Mexico (2024), https://aclanthology.org/2024.naacl-long.335
56. Zhang, Y., Henkel, J., Floratou, A., Cahoon, J., Deep, S., Patel, J.M.: ReAcTable: Enhancing react for table question answering. Proc. VLDB Endow. **17**(8), 1981–1994 (2024). https://doi.org/10.14778/3659437.3659452, https://www.vldb.org/pvldb/vol17/p1981-zhang.pdf