

VOOL: A modular insight-based framework for vocalizing OLAP sessions

Matteo Francia*, Enrico Gallinucci, Matteo Golfarelli, Stefano Rizzi

DISI, University of Bologna, Italy

ARTICLE INFO

Keywords:

Vocalization
OLAP
Business intelligence
Conversational

ABSTRACT

OLAP streamlines the exploration of multidimensional data cubes by allowing decision-makers to build sessions of analytical queries via a “point-and-click” interaction. However, new scenarios are appearing in which alternative forms of user-system communication, based for instance on natural language, are necessary. To cope with these scenarios, we present VOOL, an extensible framework for the vocalization of the results of OLAP sessions. To avoid flooding the user with long and tedious descriptions, we choose to vocalize only selected insights automatically extracted from query results. Insights are quantitative and rich-in-semantics characterizations of the results of an OLAP query, and they also take into account the user’s intentions as expressed by OLAP operators. Firstly, they are extracted using statistics and machine learning algorithms; then an optimization algorithm is applied to select the most relevant insights respecting a limit on the overall duration of vocalization. Finally, the selected insights are sorted into a comprehensive description that is vocalized to the user. After describing and formalizing our approach, we evaluate it from the points of view of efficiency, effectiveness, and operativity, also by comparing it with LLM-based applications.

1. Introduction

The OLAP (On-Line Analytical Processing) paradigm streamlines data exploration by allowing decision-makers – independently of their ICT skills – to interactively build intuitive sessions of queries on multidimensional data cubes; each query in a session is easily formulated by applying an intuitive OLAP operator to the results of the previous query. OLAP tools excellently support this paradigm via a “point-and-click” interaction, which is optimal in traditional business scenarios. However, new scenarios are appearing in which users either request hand-free interfaces [1] or they have specific visual necessities [2], thus calling for alternative forms of communication with the system, for instance using natural language. In general, there is a progressive shift of user-computer communication towards voice interfaces, which enable users to keep using the system even if they are distracted or cannot reach for the screen and the keyboard. Interestingly, translating natural language sentences into OLAP sessions has been thoroughly investigated [3]; however, the issues related to the vocalization of the results of OLAP sessions have been addressed only partially. With this paper, we aim at taking one step forward in this direction. The risk when vocalizing the results of OLAP sessions is to flood the user with long and tedious descriptions; thus, we choose to vocalize only selected insights. An insight is a quantitative and rich-in-semantics characterization of the results of an OLAP query. It is obtained by analyzing these

results through statistics and machine learning algorithms, and can be explained to users by means of a natural language description.

1.1. Requirements for OLAP vocalization

We start by emphasizing that, in OLAP scenarios, each query the user formulates within an OLAP session depends on the data (s)he has found in the cube resulting from the previous query. Indeed, the user can get precious insights by comparing the results of consecutive queries. The insights emerging during analysis can be simple (e.g., min/max) or more complex (e.g., clusters and outliers); they can illustrate any amount of facts in the cube, and could also depend on the applied OLAP operator (for instance, the roll-up operator returns a coarser view of the cube, which is very different from the goal of the slice-and-dice operator, i.e., focusing on a specific part of the cube).

The overall requirements we deem necessary for an OLAP vocalization framework can be listed as follows:

- #1 *Automation*: it must extract data-driven insights with no technical support by ICT specialists.
- #2 *Session-awareness*: it should be aimed at describing not only the results of a single query, but also the comparison of the results of subsequent queries in a session.

* Corresponding author.

E-mail addresses: m.francia@unibo.it (M. Francia), enrico.gallinucci@unibo.it (E. Gallinucci), matteo.golfarelli@unibo.it (M. Golfarelli), stefano.rizzi@unibo.it (S. Rizzi).

<https://doi.org/10.1016/j.is.2024.102496>

Received 21 December 2023; Received in revised form 25 November 2024; Accepted 26 November 2024

Available online 2 December 2024

0306-4379/© 2024 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

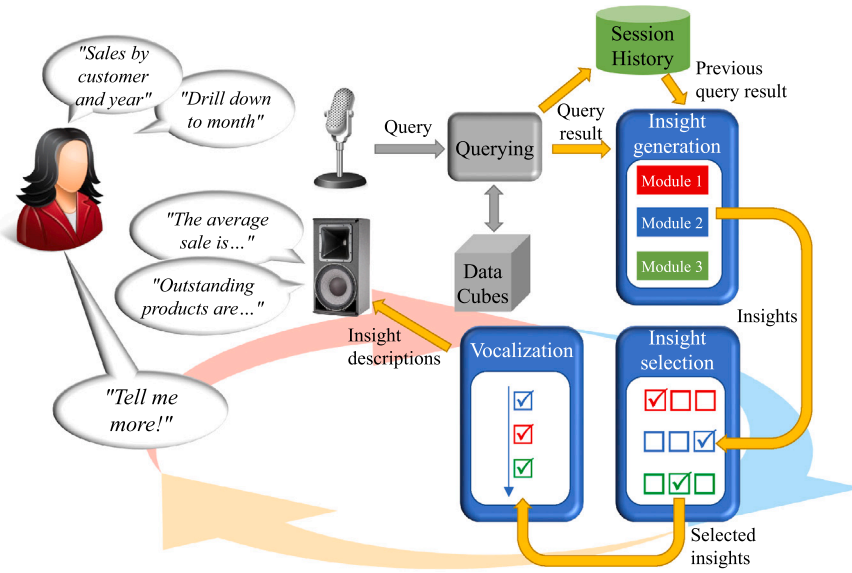


Fig. 1. Functional view of VOOL.

- #3 *Intention-awareness*: it should take the semantics of each OLAP operator into account.
- #4 *Extensibility*: it should allow an easy inclusion of new modules via suitable interfaces.
- #5 *Timeliness*: the vocalization process should return the results to the user without long delays.
- #6 *Conciseness*: the text vocalized should not be needlessly verbose.

1.2. Overview of VOOL

Inspired by the requirements above, in this paper we present a framework for the VOcalization of OLap sessions (VOOL). In Fig. 1 we give a description of VOOL from the functional point of view; the querying component is out of scope for this paper since it has been described by [3].

Vocalization of an initial query. The user first formulates an *initial*, fully-specified OLAP query q_0 (e.g., “Sales by Customer and Year”), which returns a cube C_0 . This cube is sent as input to the *insight generation* process, which executes a set of modules to analyze the cube data so as to obtain different types of insights. Each insight has a natural language description, an interest, a coverage (the number of tuples covered by the description), and a vocalization cost (e.g., the number of words of its natural language description or the duration of its vocalization). For instance, an insight produced by a Top-k module could be described in natural language as “The facts with highest Quantity are Beer with 80, Wine with 70, and Cola with 30”. Any number of modules can be executed, and each of them can return any number of insights; thus, a significantly large number of insights can be returned overall. *Insight selection* applies an optimization algorithm to this set of insights to isolate the most interesting ones based on a given limited budget (related for instance to the total duration of vocalization). *Vocalization* sorts the most interesting insights from the most general ones (i.e., the ones with higher coverage) to the most specific ones and vocalizes them using a comprehensive natural language description.

Vocalization of a refined query. The user can iteratively apply an OLAP operator to formulate a new query q_i , $i = 1, \dots, n$ (*refined query*). The cube C_i resulting from refined query q_i is sent as input to insight generation together with the one resulting from q_{i-1} , i.e., C_{i-1} . Consistently with requirements #2 and #3 (Session-awareness and Intention-awareness), the insights detected can entail not only the description of C_i , but also its comparison with C_{i-1} . For instance,

in the sales domain, after drilling down sales from product category to product, a user may be interested in excellent products that were previously concealed within average-performing categories. Following the generation of insights, vocalization is done as with an initial query.

Details-on-demand. After the selected insights have been vocalized, the user might ask for more details. To this end, VOOL supports two means of interactions: “Tell me more” and “Tell me more about F ”, where F is a module. The latter is a request for details specific to F (e.g., if we returned the Top-3 products with the highest sales, a user might be interested in expanding the description to the Top-5 products). “Tell me more” is a generic request for additional details; given a new budget, any insight not vocalized yet can be returned to the user.

1.3. Novel contributions and paper outline

A preliminary version of VOOL has been presented by [4]. The new contributions we offer here are listed below:

- We introduce the “Tell me more” interaction mode to let the user ask for further insights.
- We include additional modules in the framework, namely, *assess*, *domain variance*, and *skyline*.
- For all modules we give a formal definition of insight interest to drive the insight selection process.
- We introduce a categorization of modules.
- We formalize the algorithm for insight selection.
- We substantially extend the set of experiments by including more efficiency and user tests, comparing against large language models (LLMs), and investigating the relationship between the vocalization budget and the interest of the insights returned.

The paper structure is as follows. Section 2 formalizes the necessary background. Section 3 introduces the vocalization process, including insight generation (Section 3.1), insight selection (Section 3.2), vocalization (Section 3.3), and details-on-demand (Section 3.4). Section 4 describes the basic modules we implemented to support the VOOL framework. Section 5 presents the results of a comprehensive set of tests aimed at assessing the effectiveness, efficiency, and operativity of VOOL, also by comparing it with LLMs. Section 6 discusses the related approaches and, finally, Section 7 draws the conclusions and envisions the possible evolutions of VOOL.

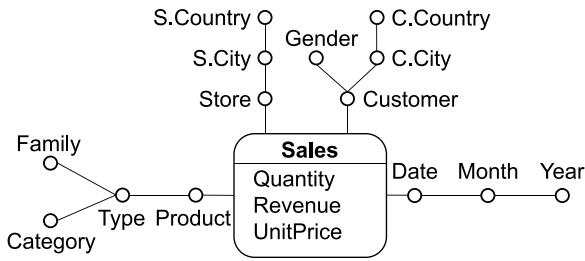


Fig. 2. (Simplified) DFM representation of the Sales cube schema.

2. Formal background

The reader is referred to [3] for a full disclosure of formal definitions. Here, we limit the discussion to the necessary background. We start with the *cube*, which is a multidimensional depiction of a business phenomenon important for decision-making.

Definition 1 (Hierarchy and Cube Schema). A *hierarchy* is a triple $h = (L, \geq, \succeq)$ where:

- (i) L is a set of categorical levels, each level l being coupled with a domain of members, $Dom(l)$;
- (ii) \geq is a roll-up partial order of L ; and
- (iii) \succeq is a part-of partial order of $\bigcup_{l \in L} Dom(l)$.

The part-of partial order is such that, for each couple of levels l and l' such that $l \geq l'$, for each member $u \in Dom(l)$ there is exactly one member $u' \in Dom(l')$ such that $u \succeq u'$. A *cube schema* is a couple $C = (H, M)$ where:

- (i) H is a set of hierarchies;
- (ii) M is a set of numerical measures, each coupled with an aggregation operator $op(m) \in \{\text{sum, avg, min, max}\}$.

Example 1. Let cube schema $Sales = (H, M)$ be our working example; Fig. 2 shows its conceptual representation according to the DFM [5]:

$$H = \{h_{Date}, h_{Customer}, h_{Store}, h_{Product}\}$$

$$M = \{Quantity, Revenue, UnitPrice\}$$

$$Store \geq S.City \geq S.Country,$$

$$Date \geq Month \geq Year,$$

$$\dots$$

We have $op(Revenue) = op(Quantity) = \text{sum}$ and $op(UnitPrice) = \text{min}$. As to the part-of partial order we have, for instance, $Venice \succeq Italy$ and $2024-04-15 \succeq 2024$. \square

Cubes are normally queried by aggregating their data according to a group-by, whose formal definition is given below.

Definition 2 (Group-by and Coordinate). Given cube schema $C = (H, M)$, a *group-by* of C is a tuple G of levels. A *coordinate* of group-by G is a tuple of members, one for each level of G . Let \succeq_H be the partial order of all the possible group-by's of C , induced by the roll-up orders of the hierarchies in H . Given coordinate γ of group-by G and another group-by G' such that $G \succeq_H G'$, we will say that γ *rolls-up* to γ' if γ' is the coordinate of G' whose members are related to the corresponding members of γ in the part-of orders.

Definition 3 (Base Cube). Let G^T be the top group-by in the \succeq_H partial order (i.e., the finest one). A *base cube* over C is a partial function C^T that maps the coordinates of G^T to a numerical value for each measure $m \in M$.

Each coordinate γ that participates in C^T , with its associated tuple of measure values, is called a *fact* of C^T . The value taken by measure m in the fact corresponding to γ is denoted $\gamma.m$. We will also treat a cube as the collection of coordinates corresponding to its facts, albeit with a minor abuse of notation.

Example 2. Three group-by's of Sales are $G^T = \langle \text{Date, Customer, Store, Product} \rangle$, $G_1 = \langle \text{Month, C.City, Gender} \rangle$, and $G_2 = \langle \text{Year} \rangle$, where $G^T \succeq_H G_1 \succeq_H G_2$. Examples of coordinates of the three group-by's are, respectively, $\gamma^T = \langle 2024-04-15, Rossi, BigMart, Beer \rangle$, $\gamma_1 = \langle 2024-04, Rome, Male \rangle$, and $\gamma_2 = \langle 2024 \rangle$, where γ_0 rolls-up to γ_1 and γ_1 rolls-up to γ_2 . \square

Definition 4 ((Cube) Query and (Derived) Cube). Given a base cube C^T over schema C , a *(cube) query* over C is a quadruple $q = (C, G_q, P_q, M_q)$ where:

- (i) G_q is a group-by of C ;
- (ii) P_q is a (possibly empty) set of selection predicates each expressed over one level of H ;
- (iii) $M_q \subseteq M$.

The result of q over base cube C^T , denoted $q(C^T)$, is called a *(derived) cube*, i.e., a partial function that assigns to each coordinate γ of G_q satisfying the conjunction of the predicates in P_q the value computed by applying $op(m)$ to the values of m for all the coordinates of C^T that roll-up to γ .

Note that, as a consequence of this definition, if no level of a hierarchy h is included in a group-by, then facts are completely aggregated along h .

Example 3. Let q a sample query over Sales which returns the total quantity sold by product; its formalization is $q = (Sales, G_q, P_q, M_q)$ where $G_q = \{\text{Product}\}$, $P_q = \emptyset$ (i.e., no selection predicate is applied), and $M_q = \{\text{Quantity}\}$. \square

An *OLAP session* is a sequence of queries, q_0, \dots, q_i ; the first query q_0 is fully stated, whereas the subsequent queries are obtained as refinements by applying an *OLAP operator* to the output of the previous query. We denote with $\omega_i \in \{\text{roll-up, drill-down, slice-and-dice}\}$ the OLAP operator applied to q_{i-1} to obtain q_i . The OLAP operators considered in VOOL are the following:

- *Roll-up*: used to aggregate data (e.g., from Month to Year).
- *Drill-down*: used to disaggregate data (e.g., from Year to Month).
- *Slice-and-dice*: used to filter data on a selection predicate (e.g., Date = 2024-04-15).

3. The vocalization process

The VOOL framework has four steps that are each explained in the subsections below: *Insight generation*, *Insight selection*, *Vocalization*, and *Details-on-demand*.

3.1. Insight generation

At this point, a collection of *modules* (such as the Top-k function or a clustering function) are executed to extract *insights* (such as the top three facts or a pair of clusters) characterizing the results of the query. Each of the *components* that make up an insight describes either a single fact (such as one of the top three facts) or a set of facts (e.g., a cluster).

Definition 5 (Module). Let C^T be a base cube, and q_{i-1} and q_i be two consecutive queries within an OLAP session, the latter being obtained by the former by applying OLAP operator ω_i (with $q_{i-1} = NULL$ when $i = 0$, i.e., the initial query is being vocalized). A *module* is a function $F(C^T, q_{i-1}, q_i, \omega_i) = S^F$, where S^F is a set of insights.

$q_0 = (\text{Sales}, \{\text{Product}\}, \emptyset, \{\text{Quantity}\})$

| Product | Quantity |
|---------|----------|
| Beer | 80 |
| Wine | 70 |
| Cola | 30 |
| Bagel | 8 |
| Pizza | 6 |
| Bread | 5 |

Fig. 3. The cube resulting from the (initial) query q_0 in Example 3, which represents the Quantity sold by Product.

A module is executable only if certain conditions are met (possibly related to the aggregation operator used in the query, the measures included in the query result, and the applied OLAP operators). It should be noted that this specification permits the application of any function capable of extracting insights from one or two cubes, in line with requirement #4 (Extensibility).

Definition 6 (Insight). An insight $s \in F(C^\top, q_{i-1}, q_i, \omega)$ is a set of components; each component $v \in s$ describes a set of facts of $q_i(C^\top)$, denoted as $Desc(v)$. Insight s is characterized as follows:

- (i) $NL(s)$ is the natural language description of s .
- (ii) $int(s)$ is the interest of the insight, i.e., its estimated relevance to the decision-making process, defined as

$$int(s) = \sum_{v \in s} int(v)$$

where $int(v) \in (0, 1]$ is the interest of component v .

- (iii) $cov(s) \in (0, 1]$ is the fraction of cube facts covered by the insight, called *coverage*:

$$cov(s) = \frac{|\bigcup_{v \in s} v|}{|q_i(C^\top)|}$$

- (iv) $cost(s) \in \mathbb{N}$ is the cost related to the vocalization of s , measured as the number of words in $NL(s)$.

The module-specific grammars used to construct the natural language descriptions of insights, $NL(s)$, are predefined. Each module has its own definition for the interest of insight components, $int(v)$ (see Section 4). Intuitively, an insight with greater scope is more general and one with less coverage is more focused.

Definition 7 (Insight Space). In the following, let \mathcal{F} be the set of all modules. Given two consecutive queries q_{i-1} and q_i in an OLAP session, their *insight space* is the set of the sets of insights produced by all modules:

$$S = \{F(C^\top, q_{i-1}, q_i, \omega); F \in \mathcal{F}\} = \{\{s_1^F, \dots, s_n^F\}; F \in \mathcal{F}\}$$

Two assumptions are made on modules and insights in order to enable simultaneous and efficient insight generation and selection:

1. Insights are *self-contained*: given an insight s , $NL(s)$ is a self-standing sentence containing all the information necessary for vocalization. As a consequence, the vocalization of an insight is independent from all the others.
2. Insights produced from the same module F are *incremental*, meaning that they can be organized in a sequence where each insight's description expands on the one before it by adding a

new component. In the following, we will assume that the ordering of indices reflects the resulting inclusion (total) ordering: $S^F = \{s_1^F, \dots, s_n^F\}$, with $cov(s_{u+1}^F) \geq cov(s_u^F)$, $int(s_{u+1}^F) \geq int(s_u^F)$, and $cost(s_{u+1}^F) \geq cost(s_u^F)$ for $1 \leq u \leq n-1$.

Example 4. Table 1 shows some of the insights generated by various modules based on the query result from Fig. 3. It should be noted that an insight may, from an informative standpoint, be an extension of another insight because it contains additional components (for instance, s_1^T is extended by s_3^T with two components—which correspond to facts Wine and Cola—while s_1^C is extended by s_2^C with one component—which corresponds to a cluster that includes two facts). \square

3.2. Insight selection

There must be a selection made on the insights to be vocalized because the insight space S can be quite large. Finding the subset of insights \bar{S} such that (i) the total interest is at its greatest and (ii) the total cost is within a specified time budget t_{voc} is the aim of this step (see requirement #6, Conciseness). Budget definition is simple for users because t_{voc} is expressed in seconds. However, in order to separate the insight cost from its vocalization, the former corresponds to the amount of words in the textual description (depending on the target audience, the optimal speech rate may vary). It is simple to convert t_{voc} into a maximum number of words; for instance, 180 is the typical word rate for English speakers and readers [6].

The one described above is unmistakably an optimization problem, with the two extra factors of non-redundancy and right-time response to be taken into account.

Non-redundancy. Even if it is assumed that distinct modules provide insights with different meanings, the insights from the same module have overlapping material (since they are built incrementally). As a result, just one insight $s_z^F \in S^F$ should be chosen given a module F and its output S^F . The multiple-choice knapsack problem (MCKP), a generalization of the knapsack problem, can thus be used to model insight selection. The MCKP divides the set of items (S) into classes (the S^F 's), and the binary decision of taking or not taking an item is replaced by the selection of no more than one item from each class [7].

Right-time response. Since the modules have varied levels of complexity and execution timeframes and are carried out in a *bag-of-task* fashion, S is progressively populated (see requirement #5, Timeliness). However, to maintain the interactive nature of the OLAP session, vocalization should start right away following the query execution, rather than waiting for the termination of all modules. To this end, only a fixed time t_{gen} is waited to begin insight selection. In the case that some modules take a longer time to finish and add insights to S after t_{gen} , the latter will not be included in the selection process, but will be taken into account if the user requests further insights in the details-on-demand stage.

The MCKP is solved through the Dymer-Zemel greedy algorithm [8], which considers the “slope” of pairs of insights (i.e., the ratio between the increase in interest and the increase in cost). The algorithm, whose pseudocode is shown in Algorithm 1, starts by considering all the insights in S and iteratively drops some insights until the budget is consumed. At first, for each module F , the algorithm builds cost-ordered pairs of insights and discards the dominated insights, i.e., those with higher cost and lower interest than another insight (Lines 4–6). If an insight has no siblings (i.e., $|S^F| = 1$), that insight is selected and copied in \bar{S} (Lines 7–10). Else, the algorithm computes the slope for each pair of insights and the median slope α of all the pairs (Lines 11–14). For each module the algorithm computes the locally optimal slope M , to be refined throughout the iterations (Lines 15–16); then it selects, among the insights with slope M , those with the lowest and highest costs (Lines 17–18). If the optimal split t_{voc} is reached [8] the algorithm stops (Lines 19–20). Otherwise, the insights with too high/low slope are dropped (Lines 21–27).

Table 1
Sample insights that describe the query result shown in Fig. 3.

| Module | Insight | NL | cost | int | cov |
|------------|---------|---|------|-----|-----|
| Statistics | s^p | The average Quantity is 33.2 | 5 | 0.0 | 1.0 |
| Top-k | s_1^T | The fact with highest Quantity is Beer with 80 | 9 | 0.4 | 0.2 |
| | s_3^T | The three facts with highest Quantity are Beer with 80, Wine with 70, and Cola with 30 | 17 | 1.0 | 0.5 |
| Clustering | s_1^C | Facts can be grouped into 2 clusters, the largest one has 4 facts and 12 as average Quantity | 18 | 0.8 | 0.7 |
| | s_2^C | Facts can be grouped into 2 clusters, the largest one has 4 facts and 12 as average Quantity, the second one has 2 facts and 75 as average Quantity | 29 | 1.6 | 1.0 |
| Assess | s_1^A | When compared to the previous query, the Quantity of Pizza is 6, tantamount to the average Quantity of Food that is 6.3 | 22 | 1.0 | 0.2 |

Algorithm 1 Insight selection algorithm

```

1:  $\bar{S} \leftarrow \emptyset$ 
2:  $stop \leftarrow FALSE$ 
3: while ! $stop$  do
4:   for each  $S^F \in S$  do ▷ For each module
5:     for each pair of insights  $(s_u^F, s_z^F)$  s.t.  $cost(s_u^F) \leq cost(s_z^F) \wedge int(s_u^F) \geq int(s_z^F)$  do ▷  $s_u^F$  dominates  $s_z^F$ 
6:        $S^F \leftarrow S^F \setminus s_z^F$  ▷ Delete the dominated insight
7:   if  $S^F = \{s^F\}$  then ▷ If a single insight is left in  $S^F$ ...
8:      $\bar{S} \leftarrow \bar{S} \cup \{s^F\}$  ▷ ... select that insight...
9:      $t_{voc} \leftarrow t_{voc} - cost(s^F)$  ▷ ... decrease the budget...
10:     $S \leftarrow S \setminus S^F$  ▷ ... and remove  $S^F$ 
11:   else
12:     for each  $(s_u^F, s_z^F)$  do
13:        $\alpha_{uz} \leftarrow \frac{int(s_z^F) - int(s_u^F)}{cost(s_z^F) - cost(s_u^F)}$  ▷ Compute the slope of  $s_u^F$  wrt to  $s_z^F$ 
14:    $\alpha \leftarrow median(\alpha_{uz})$ 
15:   for each  $S^F \in S$  do ▷ For each module
16:      $M \leftarrow argmax_{s^F \in S^F} (int(s^F) - \alpha \cdot cost(s^F))$  ▷ Select the insights with the optimal slope
17:      $s_a^F \leftarrow argmin_{s^F \in M} (cost(s^F))$  ▷ Get the insight with the lowest cost
18:      $s_b^F \leftarrow argmax_{s^F \in M} (cost(s^F))$  ▷ Get the insight with the highest cost
19:   if  $(\sum_{S^F \in S} cost(s_a^F)) \leq t_{voc} < (\sum_{S^F \in S} cost(s_b^F))$  then ▷ If optimal split...
20:      $stop \leftarrow TRUE$  ▷ ... stop the algorithm
21:   else
22:     if  $(\sum_{S^F \in S} cost(s_a^F)) \geq t_{voc}$  then
23:       for each  $(s_u^F, s_z^F)$  s.t.  $\alpha_{uz} \leq \alpha$  do
24:          $S^F \leftarrow S^F \setminus s_z^F$  ▷ Prune the insights
25:     if  $(\sum_{S^F \in S} cost(s_b^F)) < t_{voc}$  then
26:       for each  $(s_u^F, s_z^F)$  s.t.  $\alpha_{uz} \geq \alpha$  do
27:          $S^F \leftarrow S^F \setminus s_u^F$  ▷ Prune the insights

```

3.3. Vocalization

The preamble of the vocalization is a description of the query, such as “The query result shows the sum of quantity grouped by product”, and it sets the stage for the insights that follow. No pauses will be noticed by the user in the vocalization if it takes longer to vocalize the preamble than t_{gen} . The insights in \bar{S} are spoken after the preamble. In particular, they are sorted by descending coverage cov (from the broadest to the narrowest), and their natural language descriptions NL ’s are then concatenated and vocalized.

3.4. Details-on-demand

Once the insights in \bar{S} have been vocalized to the user, she might want to get more detail either in general or with reference to a specific module. To this end, VOOL supports two means of interaction: “Tell me more” and “Tell me more about F ”. Let s_z^F be the insight (if any) previously selected among those generated by module F . Clearly, the only insights eligible for subsequent selection and vocalization are those that extend s_z^F , i.e., the s_u^F ’s with $u > z$. For instance, if F = “Top-k”, the Top-3 insight is an extension of the Top-2 insight, hence, it has higher coverage (i.e., informative content) and higher cost (since more facts are described).

- “Tell me more about F ” concerns a specific module F . In response to this request, VOOL selects and vocalizes the insight $s_u^F \in S^F$, $u > z$, with the highest interest among those whose cost is lower than t_{voc} .
- “Tell me more” is a generic request for more insights. It triggers a recomputation of the MCKP on a subset S^* of S defined as follows:

$$S^* = \bigcup_{F \in \mathcal{F}} \{s_u^F \in S^F; u > z\}$$

Example 5. Given the query result shown in Fig. 3, let those in Table 1 be the insights returned by insight generation before t_{gen} expires. The insight space is $S = \{\{s^S\}, \{s_1^T, s_3^T\}, \{s_1^C, s_2^C\}, \{s_1^A\}\}$. Insight selection now takes place; assuming that the user-provided time budget is $t_{voc} = 20$ seconds, the maximum allowed number of words for vocalization is 60. The MCKP is solved and the solution $\bar{S} = \{s^S, s_3^T, s_1^C, s_1^A\}$ with cost 51 is returned. Then, the insights in \bar{S} are sorted by descending coverage, so the sequence to vocalize is $(NL(s^A), NL(s_3^T), NL(s_1^C), NL(s_1^A))$. As to details-on-demand, should the user finally ask “Tell me more about Clustering”, the system will return s_2^C . \square

4. The modules of VOOL

We categorize the modules to be plugged into VOOL along two orthogonal coordinates:

- *Operator-agnostic* vs. *Operator-specific*: modules of the former type extract insights out of the result of a single query q_i , while those of the latter type extract insights by also considering the user’s intention as expressed by the last OLAP operator applied, ω_i (e.g., given the sales by product, characterize them also by store type). Operator-agnostic modules can be applied to both initial and refined queries, while operator-specific modules are always applied to refined queries. This is because operator-specific modules also take into account the semantics of the applied OLAP operator (see requirement #4).
- *Fine-grained* vs. *coarse-grained*: in the first case each insight component describes a single fact (e.g., one of the three facts with the highest quantity sold), in the second it describes one group of facts (e.g., the average quantity of the facts in a cluster).

The main set of modules that the VOOL framework currently implements is summarized in Table 2; their applicability is explained in Table 3. We selected the modules following the *Intentional Analytics Model* (IAM) introduced by [9], which proposes five operators to capture the possible user’s intentions, namely, describe (describes one or more cube measures, possibly focused on one or more level members), assess (judges one or more cube measures with reference to some baseline), explain (reveals some hidden information in the data the user is observing, for instance in the form of a correlation between two measures), predict (shows data not in the original cubes,

Table 2
Modules currently implemented in VOOL.

| | Operator-agnostic | Operator-specific |
|----------------|-------------------|----------------------|
| Fine-grained | Bottom-k | |
| | Outliers | Aggregation variance |
| | Skyline | Assess |
| | Top-k | |
| Coarse-grained | Clustering | Domain variance |
| | Correlation | Slicing variance |
| | Statistics | |

Table 3
Modules and their applicability conditions.

| Module | Conditions |
|----------------------|--|
| Aggregation variance | $\omega \in \{\text{drill-down, roll-up}\}$ |
| Assess | $\omega \in \{\text{drill-down, slice-and-dice}\}$ |
| Bottom-k | $op(m) \neq \max$ |
| Clustering | – |
| Correlation | $ M_q \geq 2$ |
| Domain variance | $\omega \in \{\text{drill-down, roll-up}\}$ |
| Outliers | – |
| Skyline | $ M_q > 1 \wedge op(m) \neq \min$ |
| Slicing variance | $\omega \in \{\text{slice-and-dice}\}$ |
| Statistics | – |
| Top-k | $op(m) \neq \min$ |

derived for instance with regression), and suggest (shows data similar to those similar users, have been interested in). In IAM, each of these operators is associated to a set of model types, e.g., clustering, which we have used in VOOL. Specifically, our *Assess* and *Correlation* modules are related to the assess and explain IAM operators, respectively; all the other modules are related to the describe IAM operator. We have not considered *predict* (because in VOOL the user is not analyzing future data) and *suggest* (because in VOOL the user, not the system, is supposed to drive the analysis session).

Some of these modules draw their inspiration from well-known approaches [10–13]; in the majority of situations, all we had to do was coming up with a textual explanation of the insight and/or modifying the returned measurements of interest. As previously stated, this collection can easily be expanded with modules that adhere to the specifications stated in Section 1.

In the remainder of this section we start by describing in detail the Top-k module; then we will briefly describe the other modules, focusing on how they define interest. In the following, the superscript denoting the module will be dropped from the notation of insights for simplicity.

4.1. Top-k

This fine-grained and operator-agnostic module operates on both initial and refined queries and returns the facts that perform best (e.g., the sales with the highest Quantity) [14]. For simplicity, we start by considering the case when the modules operate on a single measure.

Let $q_0 = (C, G_0, P_0, M_0)$ be an initial query, with $M_0 = \{m\}$, and $C_0 = q_0(C^T)$ be the resulting cube. The goal of the Top-k module is to describe the three facts in C_0 having the highest values of m , namely, $\{\gamma_1, \gamma_2, \gamma_3\}$ (we will assume that $\gamma_1.m \geq \gamma_2.m \geq \gamma_3.m \geq \dots$). Three insights including from one to three components are returned:

$$s_1 = \{\{\gamma_1\}\}$$

$$s_2 = \{\{\gamma_1\}, \{\gamma_2\}\}$$

$$s_3 = \{\{\gamma_1\}, \{\gamma_2\}, \{\gamma_3\}\}$$

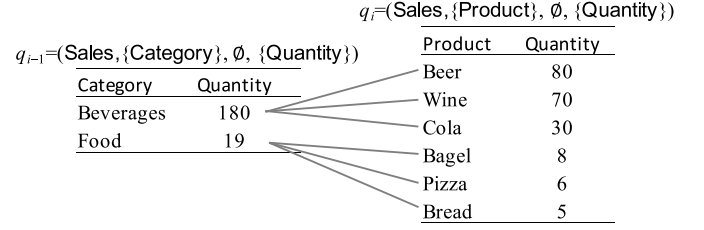


Fig. 4. Two consecutive queries, q_{i-1} and q_i , where the latter drills-down from Category to Product; gray lines highlight the correspondence between the facts on the two query results.

These insights are characterized as follows:

$$NL(s_k) = \begin{cases} \text{“The fact with highest } m \text{ is } \gamma_1 \text{ with } \gamma_1.m\text{”}, & \text{if } k = 1; \\ \text{“The two facts with highest } m \text{ are } \gamma_1 \text{ with } \gamma_1.m \text{ and } \\ \gamma_2 \text{ with } \gamma_2.m\text{”}, & \text{if } k = 2; \\ \text{“The three facts with highest } m \text{ are } \gamma_1 \text{ with } \gamma_1.m, \\ \gamma_2 \text{ with } \gamma_2.m, \gamma_3 \text{ with } \gamma_3.m\text{”}, & \text{if } k = 3; \end{cases}$$

$$cov(s_k) = \frac{k}{|C_0|}$$

As to the interest, for each component $v_k = \{\gamma_k\}$ it is

$$int(v_k) = \frac{\gamma_k.m - \bar{\gamma}_k.m}{\sum_{j=1}^{\bar{k}} (\gamma_j.m - \bar{\gamma}_k.m)}$$

where $\bar{k} > 3$. While the coverage formula is clear, the interest of s_k is the percentage of m that is retained by the Top-k tuples (for example, the total Quantity retained by the Top-3 products relative to the overall units sold by the Top- \bar{k} facts). Instead of adding up all the query results, the denominator is restricted to the Top- \bar{k} facts in order to prevent the *int()* function from having any meaning if there is a long tail of low values. In contrast, the interest function accurately reflects how high the Top-3 are relative to the next ones by only taking into account the highest non-top values (in our implementation, we set $\bar{k} = 6$). In order to deal with the situation of negative values (e.g., if the measure expresses a temperature), all measure values are shifted by $\bar{\gamma}_k.m$.

Regarding refined searches, while *NL* and *cov* remain constant, a component's interest varies based on the outcome of the prior query. Given two consecutive queries q_{i-1} and q_i , a fact in $C_i = q_i(C^T)$ is deemed interesting (in the sense of *peculiar*) if its measures differ considerably from those in the corresponding fact(s) of $C_{i-1} = q_{i-1}(C^T)$ [12]. This is based on the concept of prior belief [15]; specifically, the interest is the difference in belief for corresponding facts in the cubes before and after the application of an OLAP operator, as stated in [15]. After drilling down from Category to Product, for example, the more the Quantity of Beer deviates from the Quantity of Beverages, the higher its peculiarity; in other words, a user is less likely to anticipate that a product with exceptional sales will come from a category with average sales. This method of measuring interest necessitates the definition of the “corresponding fact(s)” in C_{i-1} for each fact in C_i . In order to achieve this, we employ a *proxy function* called $proxyc_{i-1}(\gamma)$ (with $\gamma \in C_i$) that establishes a one-to-many (many-to-one) mapping in the case of drill-down (roll-up) and a one-to-one mapping in the case of slice-and-dice or addition/removal of a measure (see Fig. 4 for an example). According to intuition, when an OLAP operator modifies the group-by, the corresponding fact(s) of C_{i-1} are determined using the roll-up order; when the operator modifies the selection predicate, the corresponding facts of C_{i-1} are one-to-one

mapped to the facts of C_i ; and when the operator modifies the measure, the corresponding facts are the empty set. For the formal definition of proxy and peculiarity $pec()$, we refer the reader to [13]. Finally, the interest of component $v_k = \{\gamma_k\}$ describing the results of a refined query is defined as for initial queries, but weighing measure values on fact peculiarity:

$$int(v_k) = \frac{(\gamma_k.m - \gamma_{\bar{k}}.m) \cdot pec(\gamma_k)}{\sum_{j=1}^{\bar{k}} (\gamma_j.m - \gamma_{\bar{k}}.m) \cdot pec(\gamma_j)}$$

Example 6. As already shown in Table 1, if C_i is the cube in Fig. 3 resulting from an initial query, examples of insights are

$$s_1^T = (NL = \text{“The fact with higher Quantity is Beer with 80”},$$

$$cost = 9, int = 0.44, cov = 0.20)$$

$$s_3^T = (NL = \text{“The three facts with higher Quantity are Beer with 80,$$

$$\text{Wine with 70, and Cola with 30”},$$

$$cost = 17, int = 0.98, cov = 0.50)$$

On the other hand, if C_i is the result of a drill-down from Category to Product as in Fig. 4, the interest changes as follows:

$$int(s_1^T) = \frac{(80 - 5) \cdot 0.21}{64.33} = 0.24$$

$$int(s_3^T) = \frac{(80 - 5) \cdot 0.21 + (70 - 5) \cdot 0.36 + (30 - 5) \cdot 1.0}{64.33} = 0.98$$

According to the previous belief principle, Beer is less interesting than Cola (which is the worst-selling beverage) even though it is the most popular product in its category. \square

In case two or more measures are included in M_i , the three top facts for each $m \in M_i$ are selected, sorted by their interest, and progressively added to the components returned. Thus, for instance, an insight could be vocalized as “The two facts with higher Quantity are Beer with 80 and Wine with 70, the fact with higher Revenue is Truffle with 300”.

4.2. Aggregation variance

This fine-grained, operator-specific module returns the facts with the highest variation in the values of a measure after a roll-up or drill-down operator (similarly to [10]; e.g., after a roll-up from Product to Category, it returns the categories showing the highest variation in the products’ Quantity). An example of vocalization is “The categories with the highest degree of variation of product quantity are Beverages and Food”. In statistics, this sparsity can be measured as the number of standard deviations ($\frac{std(v)}{avg(v)}$).

Let q_{i-1} and q_i be two consecutive queries, C_{i-1} and C_i be the cubes they return, and m be a measure in C_i . In case of a roll-up (and symmetrically for a drill-down), for component $v_j = \{\gamma_j\}$ we define

$$int(v_j) = \min\left(1, \frac{std_{\gamma \in proxy_C(\gamma_j)}(\gamma.m)}{|avg_{\gamma \in proxy_C(\gamma_j)}(\gamma.m)|}\right)$$

4.3. Assess

This fine-grained, operator-specific module evaluates the facts resulting from the current query using those of the previous query as a benchmark [12]. More specifically, (i) if the last OLAP operator applied is a slice-and-dice, the comparison is made against other sibling slices (e.g., to label the sales of a product as better or worse than those for other products of the same type); (ii) if the last applied OLAP operator is a drill-down, the comparison is made against more aggregated facts (e.g., to label the sales in a given month as good or bad based on the yearly average). An example of vocalization is “When compared to the previous query, the quantity of Pizza is 6, tantamount to the average quantity of Food which is 6.3”.

The interest of components is measured as the absolute difference between a fact and its parent (or siblings) from the previous query (normalized by the maximum deviation). For component $v_j = \{\gamma_j\}$ and with $M_i = \{m\}$ we define

$$int(v_j) = \frac{|\gamma_j.m - avg_{\gamma \in proxy_C(\gamma_j)}(\gamma.m)|}{\max_{\gamma' \in C'} |\gamma'.m - avg_{\gamma \in proxy_C(\gamma')}(\gamma.m)|}$$

4.4. Bottom-k

This fine-grained, operator-agnostic module returns the worst performing facts (e.g., sales with lowest Revenue) [14]. Symmetrically to Top-k, the most interesting insights describe the facts that retain the highest percentage of a measure m among the Bottom-k tuples. An example of vocalization is “The three facts with the lowest quantity are Bread with 5, Pizza with 6, and Bagel with 8”.

Specifically, three insights are returned, including an increasing number of components, each corresponding to one of the Bottom-3 facts. Let $\{\gamma_1, \gamma_2, \gamma_3\}$ be these facts, assuming that $\gamma_1.m \leq \gamma_2.m \leq \dots$. The interest of the component $v_j = \{\gamma_j\}$ corresponds to the percentage of m that is retained by γ_j with respect to the Bottom-k tuples; formally

$$int(v_j) = \frac{\gamma_j.m - \gamma_{\bar{k}}.m}{\sum_{u=1}^{\bar{k}} (\gamma_u.m - \gamma_{\bar{k}}.m)}$$

where $\bar{k} > 3$.

4.5. Clustering

This coarse-grained, operator-agnostic module returns groups of facts that maximize intra-group similarity and minimize inter-group similarity (e.g., facts with similar Quantity) [16]. The most interesting insights describe groups of facts characterized by similar measure values. An example of vocalization is “Facts can be grouped into 2 clusters; the largest one has 4 facts and 12 as average quantity”.

For this module each component is a cluster, described by its cardinality and by the average of the measure values for the facts it includes. The total number of clusters is determined using the Elbow method [17]. An internal clustering validation metric is used to determine the interest of each cluster [18]. In our implementation, the interest of a cluster is defined as its silhouette [19]: interesting clusters are compact and well separated from the others.

4.6. Correlation

This coarse-grained, operator-agnostic module computes the degree of Pearson correlation between pairs of measures (e.g., how Quantity and Revenue correlate). The most interesting insights describe pairs of measures characterized by (inversely) proportional values. An example of vocalization is “Quantity and revenue show strong correlation”.

Let $M_i = \{m_1, m_2\}$; a single insight including one component $v = C_i$ is returned, described by its degree of Pearson correlation $corr(m_1, m_2)$. Its interest is $int(v) = |corr(m_1, m_2)|$.

4.7. Domain variance

This coarse-grained, operator-specific module computes the degree of variation in the cardinality of the level domains after a roll-up or drill-down (e.g., after a roll-up from Product to Category, it describes whether Products are uniformly distributed among the categories). An example of vocalization is “There is a high degree of variation in the number of products for each category”.

A single insight including one component $v = C_i$ is returned by this module, described by the number of standard deviations in the cardinality of the level domains after a roll-up or drill-down. In case of a roll-up, we define

Table 4
Evaluating the user experience.

| User group | Q1 | Q2 | Q3 | Q4 | Q5 |
|------------|-------------|-------------|-------------|-------------|-------------|
| Expert | 4.36 ± 0.63 | 3.76 ± 0.85 | 3.54 ± 0.97 | 3.77 ± 1.01 | 4.07 ± 0.62 |
| Non-expert | 4.30 ± 0.67 | 3.75 ± 0.82 | 3.79 ± 1.14 | 3.87 ± 0.81 | 4.30 ± 0.48 |

$$int(v) = \min\left(1, \frac{std_{\gamma \in C_i}(|proxy_{C_{i-1}}(\gamma)|)}{avg_{\gamma \in C_i}(|proxy_{C_{i-1}}(\gamma)|)}\right)$$

Intuitively, this insight is interesting if the members in C_{i-1} are non-uniformly aggregated in C_i . Symmetrically for a drill-down.

4.8. Outliers

This fine-grained, operator-agnostic module returns the facts whose measure values deviate from the data distribution (e.g., anomalous sales) [20]. The most interesting insights describe the idiosyncratic facts (i.e., outlier facts that are not similar to any others). An example of vocalization is “Beer with 80 and Bread with 5 are two outlier facts”.

For this module each component v_j represents an outlier fact, and its interest is defined as the anomaly score [20].

4.9. Skyline

This fine-grained, operator-agnostic module, applied to at least two measures, returns the facts that “are not worse than any other”, i.e., those characterized by measure values that are not dominated (e.g., sales that represent good combinations of values for Quantity and Revenue) [13]. Specifically, a fact dominates another one if it is as good or better in all measure values and better in at least one measure value [21]. An example of vocalization is “The facts whose quantity and revenue are both higher than those of all other facts are Beer and Wine”.

For this module, each component corresponds to a fact belonging to the skyline. For component $v_j = \{\gamma_j\}$ we define

$$int(v_j) = avg_{m \in M_i}(\min MaxNorm(\gamma_j.m))$$

where $\gamma_j.m$ is min-max normalized between 0 and $max_{\gamma \in C_i}(\gamma.m)$.

4.10. Slicing variance

This coarse-grained, operator-specific module computes the degree of correlation between the values of a measure in the cubes before and after the application of a slice-and-dice operator, to describe whether the previous and current measure values are (inversely) proportional (e.g., how Quantity by Product changes after applying the selection predicate StoreCity = ‘Rome’). Assuming that a user is switching the selection predicate from Country = ‘Italy’ to Country = ‘France’, an example of vocalization is “After slicing on country, quantity from France and Italy are inversely correlated”.

Given measure m , we denote with $C_{i-1}.m$ and $C_i.m$ the values it takes within, respectively, C_{i-1} and C_i . A single insight including one component $v = C_i$ is returned, described by the degree of Pearson correlation $corr(C_{i-1}.m, C_i.m)$. Its interest is $int(v) = |corr(C_{i-1}.m, C_i.m)|$.

4.11. Statistics

This coarse-grained, operator-agnostic module returns general statistics on the overall result (e.g., the average value of the Quantity measure and its skewness). An example of vocalization is “The average quantity is 33.2 and ranges between 5 and 80”.

In this case two insights are returned: the first one includes one component $v_1 = C_i$ described by the average value of each measure $m \in M_i$ over the whole C_i ; the second one also includes component $v_2 = C_i$ described by the range of values for each $m \in M_i$. For both components, we define

$$int(v_j) = \max\left(0, 1 - \frac{std_{\gamma \in C_i}(\gamma.m)}{|avg_{\gamma \in C_i}(\gamma.m)|}\right)$$

5. Experimental evaluation

For evaluation purposes, a prototype has been developed and implemented in Python and Java. We import from the scikit-learn library the required mining models, and we rely on Google APIs to vocalize insights through the text-to-speech functionalities. All tests were made against the Foodmart¹ cube.² Our implementation and experimental setup can be found at <https://github.com/big-unibo/conversational-olap>.

5.1. User evaluation

A group of 25 users (mainly data science master students) has been put together to assess the effectiveness of VOOL. Users were divided into two groups depending on their knowledge of data warehousing and business intelligence. Non-expert and expert users amounted respectively to 42% and 58% of the users. Users received a brief introduction to vocalization and VOOL; no tutorial was required because VOOL’s use is simple. Then, three OLAP sessions with various analysis objectives were assigned to the users (e.g., “As a shop owner, you are analyzing the sum of quantity sold in each product department”) and the prototype vocalized the query results.

The users were asked to provide open-ended feedback and to answer the following questions.

- Q1 How familiar are you with the English language?
- Q2 How accurately does the vocalization describe the query result?
- Q3 How interesting are the insights produced?
- Q4 To what extent does the vocalization highlight aspects that you think are important?
- Q5 How would you rate overall your user experience with VOOL?

Table 4 reports the average results on a scale from 1 (very poor/bad) to 5 (very high/good). Both groups had the same English knowledge (Q1) and gave the same score to the quality of the vocalization provided by each insight (Q2). As to Q3-5, although all the scores show promising results, it turned out that non-expert users appreciated the interest, salience, and user experience slightly more than expert users. This shows that VOOL not only provides good results for both groups, but also that its added value in supporting non-expert data scientists is relevant.

Table 5 focuses on Q3 detailing the average appreciation of the modules whose insights were returned by VOOL during the three OLAP sessions. The majority of modules scored higher than 4 (out of 5), with the exception of Statistics and Clustering. Thanks to the open-ended feedback, we acknowledged that the former was sometimes too simplistic in describing the query result, while the latter was deemed to be repetitive over an OLAP session.

Finally, we quantitatively assessed the added value of automatic insight generation also in terms of the user effort it saves; to this end,

¹ <https://github.com/julianhyde/foodmart-data-mysql> (accessed on 2024-07-12)

² We emphasize that the modules implemented in VOOL are dataset-agnostic, hence we tested them against a single cube.

Table 5

User evaluation of the single modules.

| Module | User evaluation |
|----------------------|-----------------|
| Correlation | 4.62 ± 0.65 |
| Skyline | 4.46 ± 0.59 |
| Aggregation variance | 4.29 ± 0.89 |
| Domain variance | 4.17 ± 1.17 |
| Slicing variance | 4.12 ± 0.99 |
| Outlier detection | 4.06 ± 0.87 |
| Statistics | 3.15 ± 1.20 |
| Clustering | 3.11 ± 1.21 |

Table 6

Effort in terms of time (minutes) and code length (number of characters).

| Student | Skill | Models | Time | Length |
|---------|-------|------------|------|--------|
| STUD-1 | ADV | Clustering | 45 | 3480 |
| STUD-2 | ADV | Both | 50 | 1780 |
| STUD-3 | INT | Outliers | 25 | 940 |
| STUD-4 | ADV | Both | 60 | 1150 |
| STUD-5 | ADV | Outliers | 90 | 2630 |

we measured the time and complexity for manually reproducing the results of a small subset of modules without using VOOL. Specifically, we gave five Ph.D. students in computer science 90 min to implement the outliers and clustering modules and extract insights from our cube (we considered only these modules for the sake of time). Students were allowed to use any library from the Python ecosystem (e.g., Sklearn, Numpy, and Pandas). For each student, Table 6 shows the respective Python skills (beginner, intermediate, or advanced), the implemented modules, the time (in minutes) to complete the task, and the length of the written code measured in characters length; the quality of the modules/insights extracted is disregarded. Remarkably, implementing both modules and extracting the insights required quite a long time, even for skilled students, which ended up writing substantial Python programs (even though they were asked to compute two modules only).

5.2. Budget setting

Given the insights \bar{S} selected by the MCKP algorithm with budget t_{voc} for a given query, we define the associated total interest int_{tot} as

$$int_{tot} = \frac{\sum_{s \in \bar{S}} int(s)}{\sum_{S^F \in S} \operatorname{argmax}_{s \in S^F} (int(s))}$$

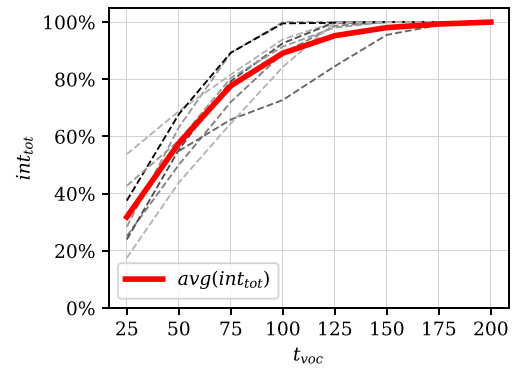
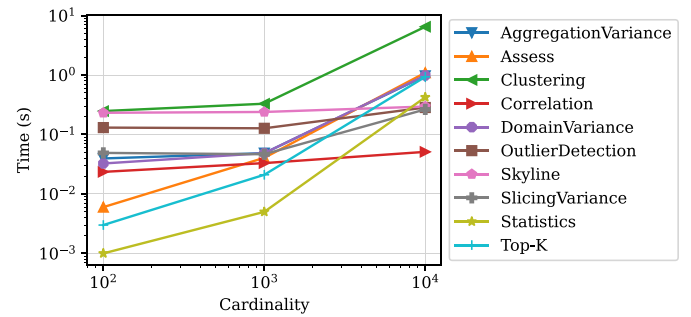
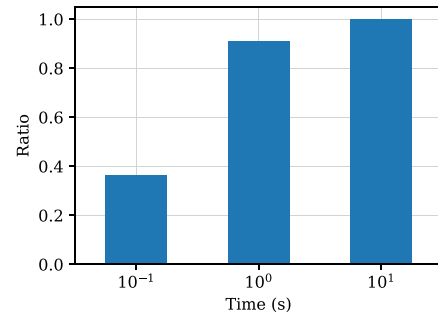
The numerator is normalized by the maximum interest (i.e., when the most interesting insight is selected for each module); this is necessary to compare the insights from different query results.

We ran 10 OLAP sessions, each including 3 OLAP operations, with varying combinations of modules (each module being invoked in a minimum of one session). Fig. 5 depicts how int_{tot} increases by increasing the budget t_{voc} (number of words). Dashed lines represent int_{tot} for a sample of queries,³ while the red line shows the average. Since the shape of int_{tot} is similar for all queries, we refer to the averaged int_{tot} .

Note that, since int_{tot} is monotonically non-decreasing and necessarily reaches a plateau (the possible insights are a finite set), the following comments can be generalized to different schemata/cubes/workloads.

The first comment is that, clearly, the lower the budget, the higher the importance of the details-on-demand interaction since it will return insights that sensibly increase the total interest. Roughly, for the scenario in Fig. 5, we can say that the details-on-demand interaction is fruitful when the budget is below 75 words.

Remarkably, this result also enables an adaptive management of the budget. As the users develop OLAP sessions to VOOL, the Elbow method [22] can be applied (as done in k-means clustering to select

**Fig. 5.** Trade-off between budget and total interest.**Fig. 6.** Performance scalability of the modules.**Fig. 7.** Percentage of completed modules over time.

the best number of clusters) to int_{tot} to suggest a vocalization budget. The Elbow-method heuristic returns a cutoff budget where the increase in the total interest is no longer worth an additional budget, which in this case turns out to be $t_{voc} = 100$. Of course, VOOL should fall back to a predefined budget when the recommended t_{voc} is too long; then, additional insights can be retrieved by asking “tell me more”.

5.3. Efficiency

The efficiency of VOOL is evaluated by assessing (i) how each module scales in terms of performance in relation to the query result cardinality and (ii) the overall responsiveness of the vocalization process. We consider the same OLAP sessions generated in Section 5.2. The tests have been conducted on a PC with 8 GB of RAM and i7-6700 CPU @3.40 GHz. The reported results are obtained as the average of 10 executions. Fig. 6 shows, for each module, the scalability in execution time in relation to cardinalities of query results the increase up to 10^4 . Since the visualization and interaction metaphors adopted by users typically constrain the analyzed results [1], a cardinality of 10^4 can be

³ We have not plotted int_{tot} for all the 30 queries for the sake of readability.

safely considered unrealistic for OLAP. Notably, the processing of all the modules takes less than one second for query results containing 10^4 facts. Clustering is the lone exception, and it takes an average of 7 s for query results with a cardinality of 10^4 .

Fig. 7 shows the percentage of modules completed within a certain time for query results including 10^4 facts. Almost 90% of the modules are completed within a few seconds, and all the insights are found within 10 s. Since the average length of the preamble is 15 words, and considering 180 as the average number of words per minute for English speakers/readers [6] (i.e., 3 words per second), the preamble lasts 5 s on average. Hence, we assumed $t_{gen} = 5$ seconds. With 10^4 facts, all modules but Clustering are completed within t_{gen} , and the results of Clustering will be available in details-on-demand mode.

5.4. Comparison against LLMs and LLM-based applications

In this section we compare VOOL against three possible competitors: GPT4o [23], Data Scientist [24], and Data Analyst [25]; the last two are ChatGPT-based applications that perform data analysis (see Section 6 for a comprehensive discussion) and embed the capabilities to generate and execute the code necessary for the elaboration of data and the application of machine learning algorithms, as well as to describe the retrieved insights. The experiment setup is the same as in Section 5.1: we consider two OLAP sessions, each starting with an initial query and proceeding with two refined queries that involve different OLAP operators (roll-up, drill-down, slicing). We incrementally prompt these sessions to GPT4o, Data Scientist, and Data Analyst.

All prompts are organized according to the guidelines from OpenAI.⁴ In particular, each prompt asks each model to adopt a specific role (i.e., data scientist), clearly indicates where the query result starts, and specifies the desired length of the output (i.e., the word budget). We do not prescribe the specific steps to complete the task, since VOOL itself has no constraints on the insight to extract. Also, we test the behavior of the competitors with and without examples.

Example 7. Here are the prompts of an initial query:

“You are a data scientist describing the highlights of query results. Given the following query result in CSV format, return the most interesting quantitative insights describing it. You can use any algorithm to compute the insights (e.g., the ones from scikit-learn). The highlights must be 100 words at most. [Query result is added here]”

and the one of a refined query obtained by drilling down the previous one:

“The following is the result of a drill-down of the previous data. Given the result in CSV format, return the most interesting quantitative insights describing it also in relationship with the previous result. You can use any algorithm to compute the insights (e.g., the ones from scikit-learn). The highlights must be 100 words at most. [Query result is added here]”

In the following we discuss how some well-known issues of LLMs problems can arise in different settings. We preliminarily recall that the output of LLMs may be affected by external knowledge (learned during their previous training phase) that is not present in the prompt. This can introduce biases that are not supported by the data, which in our setting results in claims inconsistent with the query results [26,27]. Besides, the output of LLMs is often unpredictable; different runs may lead to completely different insights and errors, depending not only on the LLM temperature parameter but also on the content of the current user session. The complete test setup, which relies on the Foodmart

Table 7
Excerpt of a refined query result.

| product Department | Gender | Quantity |
|---------------------|--------|----------|
| Produce | M | 19076 |
| Produce | F | 18711 |
| Alcoholic Beverages | F | 16939 |
| Snack Foods | M | 16859 |
| Snack Foods | F | 16186 |
| Alcoholic Beverages | M | 15399 |
| Household | M | 13757 |
| Frozen Foods | M | 13644 |
| Household | F | 13278 |
| Frozen Foods | F | 13011 |
| Baking Goods | M | 10404 |
| Baking Goods | F | 9841 |
| ... | ... | ... |
| Dairy | M | 8571 |
| Dairy | F | 8500 |
| Beverages | M | 6811 |
| Beverages | F | 6775 |
| Seafood | F | 947 |
| Seafood | M | 817 |
| Carousel | M | 473 |
| Carousel | F | 368 |

dataset to ensure reproducibility and fact-checking, is available in the above-mentioned GitHub repository.

Data input: CSV vs. prompt. Query results (i.e., tables such as Table 7) can be given in input to the LLM either as part of the prompt or as an external CSV file. In the former case, the insights provided by GPT4o, Data Scientist, and Data Analyst are enriched by the semantic capabilities of the LLM (see Table 8 for some examples). However, the insights are often quantitatively incorrect, subject to hallucinations, and affected by (biased) knowledge that is not present in the query result. For instance, males actually purchase more products from the “Produce” department than females, but all three competitors state the opposite. Besides, the statement “females lean towards Dairy and Household items” produced by Data Analyst (Table 8) is wrong (it is not a top-k) and could result from a bias created in the competitors during their training (they are all based on GPT).

On the other hand, when uploading the query result as a CSV file, all three competitors tend to generate Python code (mainly using Pandas and scikit-learn libraries) that reads that file and applies machine learning algorithms (note that these features are available without time constraints only by subscribing to ChatGPT plus). The insights (e.g., averages and quantiles computed from the data) are quantitatively correct when the competitors use the outcome of Python code, but still they incorporate hallucinations and errors when the competitors either generate semantically-wrong code or when they freely interpret the result (as in the previous case). For instance, here is the answer given by Data Analyst for the same query of Table 8 but using a CSV file to input the query result (errors are highlighted in red):

“The drill-down data provides product department quantities segmented by gender. Here are some quantitative insights. **Gender Distribution:** Female consumers **dominate** with significant quantities in key departments like Produce (18,711), Snack Foods (16,186), and Household (13,278). **Top Department Shift:** Produce surpasses Dairy, becoming the highest with 18,711 units when considering gender. **Male vs Female:** **Male representation is significantly lower, with the highest quantity being Carousel at just 473 units.** **Consistency:** **Dairy remains a top department** in overall quantity despite gender segmentation. **Total Female Quantity:** Female purchases total approximately **58,282 units**, indicating their dominant shopping pattern”.

⁴ <https://platform.openai.com/docs/guides/prompt-engineering> (accessed on 2024-07-12)

Table 8

Insights for a refined query obtained by drilling down from departments to departments and genders (query result fed via prompt; in red hallucinations and errors, see Table 7 for fact checking).

| Approach | Returned insights |
|----------------|---|
| VOOL | “Grouped by product department and sex the average quantity is 6699. All genders have similar quantity values. Facts can be grouped into 3 clusters, the largest has 22 facts and 1803 as average quantity, the second has 12 facts and 8187 as average quantity, the third has 10 facts and 15686 as average quantity. The 2 anomalous facts are (Produce, M) with 19076, and (Produce, F) with 18711.” |
| Data Analyst | “The drill-down reveals distinct gender preferences within product departments. Females predominantly purchase Produce (18,711), Snack Foods (16,186), and Alcoholic Beverages (16,939), aligning with high overall department quantities. Males, although less dominant in these categories, show significant purchases in Frozen Foods (13,644) and Household items (13,757). The notable split in Beverage purchases, with males and females contributing almost equally, reflects balanced gender interest. Males prefer Baking Goods and Alcoholic Beverages more than other categories, while females lean towards Dairy and Household items. This gender-based purchasing behavior can guide targeted marketing strategies and inventory decisions.” |
| Data Scientist | “The drill-down data provides gender-based insights. Female Dominance in Key Departments: Females purchase significantly more in departments like Produce (18,711), Snack Foods (16,186), and Alcoholic Beverages (16,939), aligning with overall high quantities in these categories from the previous dataset. Male Preferences: Males prefer Frozen Foods (13,644), Alcoholic Beverages (15,399), and Household items (13,757). Low-Performing Departments: Carousel and Seafood have low quantities, with slightly higher male engagement (473 vs. 368 in Carousel). Balanced Purchases: Departments like Canned Foods and Baking Goods show relatively balanced quantities between genders. These insights suggest tailored marketing strategies could target specific genders in top departments to boost engagement.” |
| GPT4o | “Using a k-means clustering algorithm, we observe significant gender-based differences in quantity distribution. Females contribute more to high-demand departments like Produce (18,711), Snack Foods (16,186), and Alcoholic Beverages (16,939), aligning with overall high-demand trends. Males also show a strong presence in these categories but with slightly lower quantities (Produce: 19,076, Snack Foods: 16,859, Alcoholic Beverages: 15,399). Low-demand departments like Carousel remain low for both genders. Notably, Frozen Foods exhibit balanced high demand across genders (F: 13,011, M: 13,644). These insights can refine gender-targeted marketing and inventory strategies to enhance sales efficiency.” |

The errors in this answer are mainly due Data Analyst generating Python code that extracts insights only on the first 5 tuples of the query results.

Domain: known vs. unknown. The Foodmart cube stores sales data of products sold by stores and purchased by customers. Since the product sales domain is known to LLMs, they also add knowledge that they learned during their training but is not explicitly present in the query result or even inconsistent with it. Conversely, if the dataset is altered using fake strings and column names, then the competitors only describe the distribution of the data. Here is an example from GPT4o:

“The query result contains two columns: columnA and columnB. The columnB values range from 841 to 37787. The highest value is 37787, associated with the entry “OOO” in columnA. On the other hand, the lowest value is 841, associated with the entry “VVV” in columnA. The average value in columnB is 11552.35, with a standard deviation of 10947.48. The distribution of values in columnB is slightly positively skewed, indicating that there are more lower values than higher values. The top three values in columnB are 37787, 33045, and 32338, associated with entries “OOO”, “ZZZ”, and “QQQ” in columnA, respectively”.

Prompt engineering: with vs. without examples. To ease the task of the competitors, we added examples to their prompts to guide their answers. Specifically, we use the insights provided by VOOL for the queries of one OLAP session as examples for the queries of the other session. The first line of Table 8 shows one of the examples provided. Unfortunately, it turns out that by providing these outputs to the competitors, they merely mimic VOOL and do not explore other algorithms/modules to compute their own insights. This suggests that, if we provided hundreds of examples to the competitors, eventually we would faithfully replicate the behavior of VOOL.

Additional considerations. Besides being subject to quantitative errors and hallucinations, the competitors also showed the following issues:

- (i) All of them generate their insights using only a few modules (top-k, bottom-k, clustering, outlier detection), providing quite repetitive descriptions. The tuning of cluster analysis is often sub-optimal (i.e., the competitors do not return the “best” clusters according to the elbow method).
- (ii) When the prompt is too long, the competitors seem to forget the word budget and tend to return longer answers.

- (iii) When computing their insights, the competitors do not consider the semantics of the OLAP operator connecting one query to the following one within an OLAP session. Conversely, VOOL returns operator-specific insights.

6. Related work

Our contribution intersects with two main areas of research, namely, *exploratory data analysis* and *conversational systems* (we allocated an additional section for LLMs).

6.1. Exploratory data analysis

Exploratory data analysis is an important process of knowledge discovery, where the user explores datasets in sessions by concatenating operations (e.g., filtering and aggregation) in sequence. In this context, *recommendation* and *insight extraction* are two interesting research directions.

As to recommendation, many studies are based on data profiling and users’ preferences learning techniques to recommend paths to explore [28]; for instance, discover and recommended the parts of the cube that can surprise the user in a following query by applying the principle of maximum entropy [29]. This line of work is orthogonal to VOOL, as our goal is not to support users in building an exploratory session, but to provide users with concise insights on the data returned by their own sessions (also taking the users’ intentions into account).

As to insight extraction, insight operators are classified in the following categories [30]: *coverage* (that return insights that cover tuples with certain values), *information* (that return insights providing information about the distribution of measure values), and *contrast* (that return insights occurring with some values but not the others). Examples of operators for each category are given below.

- Coverage: *Data Auditor* [31] verifies whether data constraints hold or fail to a specified degree on some fraction of the dataset. *MRI* [10] finds patterns that cover a user-specified fraction of the data and satisfy additional properties related to the variance of the measure attribute within each pattern (e.g., verify whether reviews are characterized by the gender and age of the reviewers).
- Information: *Describe* [13] computes multiple mining models (e.g., top-k and clustering) to return and highlight the most

relevant ones. *Explain* [32] uses machine learning to find relationships among numerical variables. In [11], user insights' accuracy (e.g., mean, shape, variance, ranking, and correlation of a query result) is extracted and confirmed by the authors.

- *Contrast*: *DIFF* [33] finds the rows that better explain an increase/decrease in quantities aggregated at a coarser aggregation level. *Assess* [34] enables custom comparison of OLAP cubes and labels the comparison result. In [35], the authors output the combinations of attribute-value pairs whose diversity is above a threshold.

A recent proposal in the same direction is the Intentional Analytics Model [9], which envisions the coupling of OLAP and analytics into a new paradigm. There, users explore data by communicating their analytical intentions [12,13,36], and cubes annotated with knowledge insights are returned. All the above-mentioned insight operators are complementary to our proposal since they can be potentially plugged into VOOL, just as we did for [10–13].

Some recent approaches aim to automatic extraction of insights to aid data exploration. *QuickInsights* [37] and *MetaInsight* [38] extract insights (e.g., top-k and outliers) under the white-box assumption: knowing how each insight is computed, both frameworks apply pruning strategies for pattern computation and highlight selection. However, while the white-box assumption eases the computation of the insights, it limits expressiveness. Indeed, both frameworks require mapping the subspaces covered by the insights. Defining such spaces is easy when insights are computed on single cubes but not when multiple cubes (with different granularities) are considered. For instance, this is necessary to compute comparison insights during an OLAP session and not on single queries, as done by *Assess* [34]. Finally, *Cinecubes* [39] has a hard-coded mechanism to analyze query results: it compares the result of a given query to results obtained over sibling values or drill-downs to produce insight-based descriptions of such comparisons (similarly to a recommendation system); additionally, it does not propose interest measures to score the insights nor an optimization mechanism to balance the vocalization cost and the interest of the returned insights.

Overall, (i) the insights computed by *QuickInsights* [37] and *MetaInsight* [38] are pluggable into VOOL at the cost of losing some pruning efficiency (if the module implementation is a white box and known to the framework, pruning techniques can be applied to constrain the search of the insights); (ii) we provide guidelines to plug (even black-box) modules into the insight generation process; and (iii) we introduce an optimization approach to select the best insights and to sort them into an organized description.

6.2. Conversational systems

To enable conversational analytics on multidimensional data (e.g., in hand-free scenarios), three main issues have to be addressed: (i) natural-language query formulation, (ii) summarization, and (iii) vocalization of query results.

As to query formulation, natural language interfaces to operational databases enable users with no prior training on formal programming languages (e.g., SQL) and software to specify complex queries; a comprehensive survey on this subject is provided by [40]. Approaches that obtain formal SQL/OLAP queries by translating natural language [3,41–43] are out of scope since they work in the text-to-query direction rather than in the query-to-text one.

As to summarization, besides the approaches discussed in the previous subsection, other approaches have been proposed to reduce the cardinality of the returned results while retaining as much information as possible. In particular, [1,44] exploit the multidimensional and multilevel nature of query results to further shrink the returned data. These approaches can be reused in the VOOL framework, for instance, wrapped in a module that vocalizes a small number of facts representative of a subset of data (e.g., if the sales by product have similar

quantities for the 'Beverages' category, return their average quantity by category).

As to vocalization, the constraints of spoken presentation are well-understood [45,46], and some approaches have been introduced for the vocalization of single query results. [47] uses a set of rules and templates to translate the subset of a database into a narrative synthesizing its contents. [48] leverages the provenance of tuples in the query result to detail not only the results, but also their explanations; the authors map the query in natural language to a dependency tree, whose structure is used to provide the answer to the user through the replacement of words in the question with values from the result and provenance. A few works are then discussed in relation to OLAP and multidimensional data. [2] samples the database to assess various speech fragments; OLAP queries are not fully analyzed and sampling concentrates on result features that are important for voice output. While an end-to-end dialog system is proposed by [49], the vocalization technique is constrained (for example, the returned answer contains only the count of rows if the number of rows or columns to be returned is too high).

Overall, it seems that the path to fully-fledged conversation-driven OLAP has not yet been built in light of the aforementioned contributions. Indeed, users that are not familiar with SQL language and OLAP tools can be aided in data exploration activities through conversational interfaces and personal assistants. Nonetheless, in the area of analytical sessions over multidimensional data, end-to-end conversational frameworks are not offered. [2] is the closest contribution, however: (i) its vocalization does not compare the query results sequentially emerging from OLAP sessions, but it copes only with stand-alone queries; (ii) custom user-specific modules cannot be plugged; and, (iii) vocalization is not based on insights interest.

6.3. LLMs and LLM-based applications

Generative AI and Large Language Models (LLMs) [50,51] have recently witnessed a huge hype even in the data science community. LLMs are autoregressive machine learning models that act as statistical next-word predictors [52] after being trained on huge datasets [26]. The applications leveraging LLMs, such as ChatGPT, are usually oriented to general-purpose information retrieval and are not specific for extracting, ranking, and selecting the most interesting insights out of multidimensional query results. Thus, although LLMs are powerful tools for broad natural language applications, they are not always the best choice for data-intensive tasks for two main reasons.

- *Data volume and cost*. Since multidimensional data usually contain sensitive business information, they are stored in private repositories (such as data warehouses) unknown to LLMs during their training [51]. To feed data to an LLM, the main possibility is to use the prompt. Depending on the specific model, prompts have limits in the number of tokens shared between input and answer, with different models having different limits⁵ [51,53] and pricing⁶ on input and output tokens. Recently, LLMs began to allow users to attach files to the prompt [51], and some LLM-based applications can even extract succinct summaries from these files and use them in place of the whole file content. On the one hand, this can overcome the limits and cost of tokens. On the

⁵ <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them> (accessed on 2024-07-12)

⁶ The OpenAI price calculator in Microsoft Azure⁷ estimates that using GPT-4-32K costs around \$0.06 per 10³ input tokens and \$0.12 per 10³ output tokens (as of 2024-06-20), where 1000 tokens correspond to almost 750 words. For instance, prompting a table with 10⁵ tuples (if feasible at all) could easily reach the cost of several dollars per execution.

⁷ <https://azure.microsoft.com/en-us/pricing/calculator/?service=openai-service> (accessed on 2024-07-12)

other hand, if the summary process is incorrect, it will add errors and bias to the final answer.

- *Running algorithms.* Algorithmic tasks such as SQL querying and data mining are better handled by DBMSs and query engines since (i) they are optimized for these types of operations (e.g., using R-trees to speed up clustering in Euclidean spaces); (ii) their answers are correct, consistent, and reproducible; and (iii) they do not have hallucinations as LLMs do [51] (see Section 5.4).

Overall, we conclude that while (plain) LLMs have many strengths, they are not typically suitable for data-intensive tasks. Several LLM-based applications have been deployed to overcome these limitations. For instance, there are applications for data analysis that are capable of generating and interpreting code (mainly Python), analyzing datasets, and supporting data scientists in doing data elaborations and analyses. Applications can be directly deployed by LLM providers (e.g., Data Analyst [25] by ChatGPT) or can be deployed by third-party users (e.g., Data Scientist [24]). Intuitively, these applications allow the underlying LLM to invoke functions from external libraries when appropriate (e.g., Python’s pandas, scikit-learn, and scipy) and to generate and execute external code. Data Scientists and Data Analyst are indeed closer to VOOL than plain LLMs, but they bear the following differences.

- *Interpretability.* Thanks to our structured approach, different modules can be transparently plugged into the VOOL framework. In LLMs, the process that leads to the computation of the insights and their interest remains hidden and not interpretable to the end-user, who is not aware of how insights are extracted (e.g., if the model has added some distortion to the data before applying a machine learning model). Indeed, the term *language model* mainly refers to systems trained on the prediction of a token given its surrounding context [26] without any guarantees on interpretability, correctness, and quality [54].
- *Domain-specific data.* The insights returned by LLM-based applications strongly depend on the context since LLMs also leverage domain knowledge for their “reasoning”; hence, it is usually recommended to use semantic-rich column names to enhance data analyses (see Section 5.4). On the one hand, this domain knowledge can introduce bias and inconsistencies with the data at hand. On the other hand, for very specific domains and data types, a lack of knowledge can cause issues in the interpretation of the result. For instance, in precision agriculture, having low temperatures could be bad for the production rates, but good in terms of pest control and water management. In VOOL, no attempt is made to give an interpretation of the result (which could be subjective, domain-dependent, and misleading), but rather extract objective and peculiar insights out of the data.
- *Domain-specific modules.* Should additional modules be necessary to produce domain-specific insights, such modules should be embedded into the LLM through prompting or calls to (external) third-party libraries (e.g., function calling in ChatGPT⁸). To this end, the end-user should declare to the LLM the behavior of each module as well as the interest function to rank the insights—which would mean re-implementing a VOOL-like tool.
- *Multidimensionality.* While applications such as Data Scientist normally work on machine learning datasets – typically, denormalized data stored in flat files, for instance in CSV format –, VOOL is meant to operate on multidimensional data stored in data warehouses, whose metadata express a lot of precious knowledge (attribute hierarchies, primary/foreign keys, semantics of OLAP operator, etc.) that cannot be expressed in plain CSV files [55]. Remarkably, VOOL natively exploits this knowledge without the

need for prior training, for instance to compute operator-specific insights and their interest.

We highlight that LLMs are also exposed to a couple of additional issues.

- *Libraries.* LLMs produce good results with well-known libraries such as Python’s scikit-learn and pandas. When less-known libraries are required, LLMs can fail to produce the necessary code.
- *Development.* Many LLM-based applications are commercial (or handcrafted) and not associated with research papers. Hence, following their development and deeply understanding their capabilities is often hard, also due to a lack of documentation. On the one hand, this makes non-empirical comparisons between LLMs unfeasible. On the other, exactly reproducing the results of empirical tests may be impossible.

7. Conclusion and future work

Conversational systems play a significant role in enhancing data analysis capabilities by providing user-friendly interfaces for interacting with data. We have presented VOOL, an approach for selecting and vocalizing interesting insights out of the results of an OLAP session, respecting a user-defined vocalization budget. Overall, the experiments confirm the efficiency and effectiveness of our approach, showing that (i) its performances are fully compatible with the level of interaction required by an OLAP session, (ii) users appreciate the provided experience, and non-expert users also consider VOOL as an added value to their analysis, and (iii) the returned insights are generally marked as interesting, confirming that our interest function can appropriately select interesting insights.

In this paper we specifically discussed the differences between VOOL and LLMs. Based on that discussion, our claim is that applications such as Data Scientist and Analyst, as well as GTP4o (or other LLMs), are, at the state of the art, valid tools to support data scientists (but not to automatically pilot the extraction of insights, as much as Copilot is an incredible tool to support software engineers but not to replace them [56]). However, due to hallucinations, quantitative errors, and to the need for continuous fact-checking, their adoption as reliable end-to-end tools for data exploration and insight extraction is still limited [57]. Conversely, VOOL focuses on automatically providing quantitatively correct and interesting insights to describe and augment the query results. This is important in domains such as Business Intelligence, where query results are the core of data-driven decisions.

Besides refining and extending the modules, other directions that can be envisioned for future research are the following. Tracking the insights vocalized during a single OLAP session to avoid repeating them after different queries in the session. Tuning insight selection depending on the adoption of VOOL: if adopted atop a visualization tool such as Tableau or PowerBI, users could prefer the highlight of idiosyncratic patterns rather than a comprehensive description of the query result. Finally, extending the interest of an insight also to its overlapping with the previously selected ones.

CRedit authorship contribution statement

Matteo Francia: Writing – original draft, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Enrico Gallinucci:** Writing – original draft, Software, Conceptualization. **Matteo Golfarelli:** Writing – original draft, Supervision, Methodology, Investigation, Conceptualization. **Stefano Rizzi:** Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization.

⁸ <https://platform.openai.com/docs/guides/function-calling> (accessed on 2024-11-22).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

- [1] M. Francia, M. Golfarelli, S. Rizzi, A-BI⁺: A framework for augmented business intelligence, *Inf. Syst.* 92 (2020) 101520.
- [2] I. Trummer, Y. Wang, S. Mahankali, A holistic approach for query evaluation and result vocalization in voice-based OLAP, in: *Proc. SIGMOD*, Amsterdam, The Netherlands, 2019, pp. 936–953.
- [3] M. Francia, E. Gallinucci, M. Golfarelli, COOL: A framework for conversational OLAP, *Inf. Syst.* (2021).
- [4] M. Francia, E. Gallinucci, M. Golfarelli, S. Rizzi, Insight-based vocalization of OLAP sessions, in: *Proc. ADBIS*, Turin, Italy, 2022, pp. 193–206.
- [5] M. Golfarelli, D. Maio, S. Rizzi, The dimensional fact model: A conceptual model for data warehouses, *Int. J. Cooperative Inf. Syst.* 7 (2–3) (1998) 215–247.
- [6] M. Brysbaert, How many words do we read per minute? A review and meta-analysis of reading rate, *J. Mem. Lang.* 109 (2019) 104047.
- [7] H. Kellerer, U. Pferschy, D. Pisinger, The multiple-choice knapsack problem, in: *Knapsack Problems*, Springer, 2004, pp. 317–347.
- [8] H. Kellerer, U. Pferschy, D. Pisinger, Multidimensional knapsack problems, in: *Knapsack Problems*, Springer, 2004, pp. 235–283.
- [9] P. Vassiliadis, P. Marcel, S. Rizzi, Beyond Roll-Up's and Drill-Down's: An intentional analytics model to reinvent OLAP, *Inf. Syst.* 85 (2019) 68–91.
- [10] M. Das, S. Amer-Yahia, G. Das, C. Yu, MRI: Meaningful interpretations of collaborative ratings, *Proc. VLDB Endow.* 4 (11) (2011) 1063–1074.
- [11] E. Zgraggen, Z. Zhao, R.C. Zeleznik, T. Kraska, Investigating the effect of the multiple comparisons problem in visual analysis, in: *Proc. CHI*, Montreal, QC, Canada, 2018, p. 479.
- [12] M. Francia, M. Golfarelli, P. Marcel, S. Rizzi, P. Vassiliadis, Assess queries for interactive analysis of data cubes, in: *Proc. EDBT*, Nicosia, Cyprus, 2021, pp. 121–132.
- [13] M. Francia, P. Marcel, V. Peralta, S. Rizzi, Enhancing cubes with models to describe multidimensional data, *Inf. Syst. Front.* 24 (1) (2022) 31–48.
- [14] Z.W. Luo, T.W. Ling, C.H. Ang, S.Y. Lee, B. Cui, Range top/bottom k queries in OLAP sparse data cubes, in: *Proc. DEXA*, 2001, pp. 678–687.
- [15] T.D. Bie, Subjective interestingness in exploratory data mining, in: *Proc. IDA*, 2013, pp. 19–31.
- [16] A. Likas, N. Vlassis, J.J. Verbeek, The global k-means clustering algorithm, *Pattern Recognit.* 36 (2) (2003) 451–461.
- [17] L. Rokach, O. Maimon, Clustering methods, in: *Data Mining and Knowledge Discovery Handbook*, Springer, 2005, pp. 321–352.
- [18] Y. Liu, Z. Li, H. Xiong, X. Gao, J. Wu, Understanding of internal clustering validation measures, in: *2010 IEEE International Conference on Data Mining*, IEEE, 2010, pp. 911–916.
- [19] P.J. Rousseeuw, Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, *J. Comput. Appl. Math.* 20 (1987) 53–65.
- [20] F.T. Liu, K.M. Ting, Z. Zhou, Isolation forest, in: *Proc. ICDM*, Pisa, Italy, 2008, pp. 413–422.
- [21] S. Borzsony, D. Kossmann, K. Stocker, The skyline operator, in: *Proc. ICDE*, IEEE, 2001, pp. 421–430.
- [22] V. Satopaa, J.R. Albrecht, D.E. Irwin, B. Raghavan, Finding a “Kneedle” in a Haystack: Detecting knee points in system behavior, in: *Proc. ICDCS*, 2011, pp. 166–171.
- [23] OpenAI, GPT4o, 2024, <https://openai.com/index/hello-gpt-4o/> (Accessed 12 July 2024).
- [24] P.C. Sekha, Data scientist, 2024, <https://chatgpt.com/g/g-UgvIRTBjV-data-scientist> (Accessed 12 July 2024).
- [25] ChatGPT, Data analyst, 2024, <https://chatgpt.com/g/g-HMncP6w7d-data-analyst> (Accessed 12 July 2024).
- [26] E.M. Bender, T. Gebru, A. McMillan-Major, S. Shmitchell, On the dangers of stochastic parrots: Can language models be too big? in: *Proc. FAccT*, 2021, pp. 610–623.
- [27] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P.S. Yu, Q. Yang, X. Xie, A survey on evaluation of large language models, *ACM Trans. Intell. Syst. Technol.* 15 (3) (2024) 39:1–39:45.
- [28] L. Song, J. Gan, Z. Bao, B. Ruan, H.V. Jagadish, T. Sellis, Incremental preference adjustment: A graph-theoretical approach, *VLDB J.* 29 (6) (2020) 1475–1500.
- [29] S. Sarawagi, User-adaptive exploration of multidimensional data, in: *Proc. VLDB*, Cairo, Egypt, 2000, pp. 307–316.
- [30] L. Golab, D. Srivastava, Exploring data using patterns: A survey and open problems, in: *Proc. DOLAP@EDBT/ICDT*, Nicosia, Cyprus, 2021, pp. 116–120.
- [31] L. Golab, H.J. Karloff, F. Korn, D. Srivastava, Data auditor: Exploring data quality and semantics using pattern tableaux, *Proc. VLDB Endow.* 3 (2) (2010) 1641–1644.
- [32] M. Francia, S. Rizzi, P. Marcel, Explaining cube measures through intentional analytics, *Inf. Syst.* 121 (2024) 102338.
- [33] S. Sarawagi, Explaining differences in multidimensional aggregates, in: *Proc. VLDB*, Edinburgh, Scotland, 1999, pp. 42–53.
- [34] M. Francia, M. Golfarelli, P. Marcel, S. Rizzi, P. Vassiliadis, Suggesting assess queries for interactive analysis of multidimensional data, *IEEE Trans. Knowl. Data Eng.* 35 (6) (2023) 6421–6434.
- [35] F. Abuzaid, P. Kraft, S. Suri, E. Gan, E. Xu, A. Shenoy, A. Ananthanarayan, J. Sheu, E. Meijer, X. Wu, J.F. Naughton, P. Bailis, M. Zaharia, DIFF: A relational interface for large-scale data explanation, *VLDB J.* 30 (1) (2021) 45–70.
- [36] M. Francia, S. Rizzi, P. Marcel, The whys and wherefores of cubes, in: *Proc. DOLAP@EDBT/ICDT*, Ioannina, Greece, 2023, pp. 43–50.
- [37] R. Ding, S. Han, Y. Xu, H. Zhang, D. Zhang, QuickInsights: Quick and automatic discovery of insights from multi-dimensional data, in: *Proc. SIGMOD*, Amsterdam, The Netherlands, 2019, pp. 317–332.
- [38] P. Ma, R. Ding, S. Han, D. Zhang, MetaInsight: Automatic discovery of structured knowledge for exploratory data analysis, in: *Proc. SIGMOD*, 2021, pp. 1262–1274.
- [39] D. Gkesoulis, P. Vassiliadis, P. Manousis, CineCubes: Aiding data workers gain insights from OLAP queries, *Inf. Syst.* 53 (2015) 60–86.
- [40] K. Affolter, K. Stockinger, A. Bernstein, A comparative survey of recent natural language interfaces for databases, *VLDB J.* 28 (5) (2019) 793–819.
- [41] F. Li, H.V. Jagadish, Understanding natural language queries over relational databases, *SIGMOD Rec.* 45 (1) (2016) 6–13.
- [42] D. Saha, A. Floratou, K. Sankaranarayanan, U.F. Minhas, A.R. Mittal, F. Özcan, ATHENA: An ontology-driven system for natural language querying over relational data stores, *PVLDB* 9 (12) (2016) 1209–1220.
- [43] M. Francia, E. Gallinucci, M. Golfarelli, Towards conversational OLAP, in: *Proc. DOLAP@EDBT/ICDT*, Copenhagen, Denmark, 2020, pp. 6–15.
- [44] M. Golfarelli, S. Graziani, S. Rizzi, Shrink: An OLAP operation for balancing precision and size of pivot tables, *Data Knowl. Eng.* 93 (2014) 19–41.
- [45] V. Demberg, A. Winterboer, J.D. Moore, A strategy for information presentation in spoken dialog systems, *Comput. Linguist.* 37 (3) (2011) 489–539.
- [46] O.B. El, T. Milo, A. Somech, Towards autonomous, hands-free data exploration, in: *Proc. CIDR*, Amsterdam, The Netherlands, 2020.
- [47] A. Simitsis, G. Koutrika, Y. Alexandrakis, Y.E. Ioannidis, Synthesizing structured text from logical database subsets, in: *Proc. EDBT*, Nantes, France, 2008, pp. 428–439.
- [48] D. Deutch, N. Frost, A. Gilad, Explaining natural language query results, *VLDB J.* 29 (1) (2020) 485–508.
- [49] G. Lyons, V. Tran, C. Binnig, U. Çetintemel, T. Kraska, Making the case for query-by-voice with EchoQuery, in: *Proc. SIGMOD*, ACM, New York, NY, USA, 2016, pp. 2129–2132.
- [50] T.B. Brown, et al., Language models are few-shot learners, in: *Proc. NeurIPS*, 2020.
- [51] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F.L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., GPT-4, Technical Report, 2023, arXiv:2303.08774.
- [52] S.R. Bowman, Eight things to know about large language models, 2023, arXiv:2304.00612.
- [53] T.B. Brown, Language models are few-shot learners, 2020, arXiv:2005.14165.
- [54] C. Spiess, D. Gros, K.S. Pai, M. Pradel, M.R.I. Rabin, A. Alipour, S. Jha, P. Devanbu, T. Ahmed, Calibration and correctness of language models for code, 2024, arXiv:2402.02047.
- [55] M. Golfarelli, S. Rizzi, A methodological framework for data warehouse design, in: *Proc. DOLAP*, 1998, pp. 3–9.
- [56] A. Ziegler, E. Kalliamvakou, X.A. Li, A. Rice, D. Rifkin, S. Simister, G. Sitampalam, E. Aftandilian, Measuring GitHub Copilot's impact on productivity, *Commun. ACM* 67 (3) (2024) 54–63.
- [57] P.J. Denning, Can generative AI bots be trusted? *Commun. ACM* 66 (6) (2023) 24–27.