BABOONS: Black-Box Optimization of Data Summaries in Natural Language

Immanuel Trummer
Cornell University
Ithaca, NY
itrummer@cornell.edu

ABSTRACT

BABOONS (BlAck BOx Optimization of Natural language data Summaries) optimizes text data summaries for an arbitrary, user-defined utility function. Primarily, it targets scenarios in which utility is evaluated via large language models. Users describe their utility function in natural language or provide a model, trained to score text summaries in a specific domain.

BABOONS uses reinforcement learning to explore the space of possible descriptions. In each iteration, BABOONS generates summaries and evaluates their utility. To reduce data processing overheads during summary generation, BABOONS uses a proactive processing strategy that dynamically merges current with likely future queries for efficient processing. Also, BABOONS supports scenario-specific sampling and batch processing strategies. These mechanisms allow to scale processing to large data and item sets. The experiments show that BABOONS scales significantly better than baselines. Also, they show that summaries generated by BABOONS receive higher average grades from users in a large survey.

PVLDB Reference Format:
Immanuel Trummer. BABOONS: Black-Box Optimization of Data Summaries in Natural Language. PVLDB, 15(11): 2980 - 2993, 2022.
doi:10.14778/3551793.3551846

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/itrummer/BABOONS.

1 INTRODUCTION

Data are often summarized as text. The best way to summarize data depends entirely on the context. Who is the target audience? What is the purpose of the summary? The answers to those questions determine which facts to include into a summary, and which ones to neglect. The BABOONS (BlAck BOx Optimization of Natural language data Summaries) system gives users maximal flexibility in specifying a utility function, used to score data summaries. Its primary use case are scenarios where users evaluate (not generate) data summaries via large language models, i.e. complex neural networks used for text analysis. Users can specify a model, trained to score data summaries in a particular domain, to use as utility function. Alternatively, users specify requirements on a data summary in natural language. Then, a language model can be used to compare candidate summaries against user instructions.

BABOONS generates comparative data summaries. A comparative summary describes differences between a data subset and the full data. For instance, a data subset may be associated with a specific product to advertise. A summary consists of facts. Each fact outlines a deviation in average values between the data subset and the full data. This comparison may only consider a subset of rows with certain properties, described via equality predicates. The following example illustrates those concepts, a corresponding demonstration video is available online: https://youtu.be/ssGwZcUkMKA.

Example 1.1. A shop owner writes short advertisement texts for laptops described in Table 1. It is recommended to integrate concrete and accurate statistics into such texts [1, 5]. To use BABOONS, the owner first specifies the search space for data summaries. This entails specifying the number of facts per summary and the number of properties (i.e., equality predicates and aggregates) per fact. Furthermore, the owner specifies dimension columns (on which equality predicates are placed) and aggregation columns. Next, the owner specifies one text template for each dimension and aggregation column. A dimension template expresses an equality predicate in natural language and contains a placeholder for the constant. For aggregation columns, templates express a comparison with a placeholder for the relative average. Finally, users specify a preamble that prefixes each fact text. Table 2 shows text templates and search space specification for the laptops data (it links to symbols introduced in Section 2). In addition, the shop owner specifies a model used to evaluate candidate summaries (e.g., a model for sentiment analysis to favor summaries with positive sentiment). Now, the shop owner can use the system to generate summaries with positive sentiment for specific laptops or specific laptop brands. The comparison focus is defined by specifying an SQL predicate, identifying the data subset to describe. For instance, specifying the predicate ID=4 may result in the summary shown in Table 3.

BABOONS requires users to specify one text template for each column in the data set. As shown in the example, those templates are relatively simple. They can be reused for different summaries within the same scenario (e.g., for describing different laptops in Example 1.1). Hence, the overheads for users are moderate. The advantage of using template-based text generation (as opposed to a neural approach [21, 35, 72]) is robustness: assuming accurate templates, each generated summary is correct and understandable. Post-processing may be required to identify cases where the model does not accurately evaluate utility. This can be done by the user

Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551846

Table 1: Example data describing laptop models.

ID	Brand	CPU	Disk	Rating	Price	Battery
1	IBM	2.4 GHz	500 GB	2	1,999	6
2	Mac	2.6 GHz	1 TB	4	2,499	12
3	Mac	2.2 GHz	128 GB	5	1,999	16
4	Dell	3.2 GHz	1 TB	4	1,999	3

Table 2: Description of data source and summary template for example data (Table 1). [P] represents placeholders that are replaced by dimension values or aggregates.

Symbol	Semantics	Value
$\mathcal{D}.dims$	Dimensions	$\{Brand, CPU, Disk, Rating\}$
$\mathcal{D}.aggs$	Aggregation	$\{Price, Weight\}$
$\mathcal{T}.n_F$	Nr. facts	1
$\mathcal{T}.n_P$	Nr. properties	3
$\mathcal{T}.preamble$	Preamble	Among all laptops
$\mathcal{T}.predtxt_{Brand}$	Brand text	of [P] brand
$\mathcal{T}.predtxt_{CPU}$	CPU text	with [P] CPU
$\mathcal{T}.predtxt_{Disk}$	Disk text	with [P] disk space
$\mathcal{T}.predtxt_{Rating}$	Rating text	with [P] / 5 stars
$\mathcal{T}.aggtxt_{Price}$	Price text	its discounted price is [P] than average
$\mathcal{T}.aggtxt_{Battery}$	Battery text	its battery life is [P] than average

Table 3: Example fact about laptop with ID 4 (substitutions for placeholders in text templates are marked in bold font).

Part	Text
Preamble	Among all laptops ...
Predicate 1	with **1 TB** disk space ...
Predicate 2	with **4** / 5 stars ...
Aggregate	its discounted price is **11% lower** than average.

for small item sets or via crowd sourcing for larger ones (typically for a fee of few cents per verification).

BABOONS uses a deep reinforcement learning approach. Reinforcement learning is a popular method for black box optimization as it does not place constraints on the nature of the utility function. Guided by reinforcement learning, BABOONS iteratively selects data subsets and aggregates, generates corresponding text summaries, and evaluates them via the user-defined utility function. The summary with highest utility is ultimately returned to the user.

Generating summaries requires data processing to calculate associated aggregates. This becomes expensive for large data sets, in particular as BABOONS queries the data iteratively. To scale to large data sets, BABOONS uses several problem-specific mechanisms that increase efficiency. First, BABOONS uses a proactive cache manager. This cache manager predicts likely future queries, based on the state of the reinforcement learning algorithm, that can be easily merged with queries scheduled for execution. The results of likely queries are cached and can be retrieved in future iterations. Second, BABOONS supports sampling strategies that evaluate data summaries on data samples. This becomes challenging as the utility function may be non-linear (i.e., small deviations between sample estimates and accurate aggregates may cause significant differences in utility). BABOONS deals with this challenge by calculating error margins for aggregates within which the text description does not change. Finally, BABOONS supports multiple batch processing mechanisms to scale to large numbers of items.

The experiments simulate an advertisement scenario where items of different categories (ranging from laptops to airlines) need to be advertised, based on data sets. The experiments show that BABOONS is able to scale to large data sets as well as to large item batches. Furthermore, the experiments show that BABOONS performs significantly better than baselines. The experiments also include the results of a survey, asking hundreds of human users to evaluate advertisements generated by different approaches. It turns out that descriptions generated by BABOONS are preferred over summaries generated by baselines. In summary, the original scientific contributions in this paper are the following:

- It introduces the problem of optimizing comparative data summaries according to black-box utility functions.
- It describes the design of BABOONS, a system that solves the aforementioned problem via reinforcement learning.
- It experimentally compares BABOONS to baselines.

The remainder of this paper is organized as follows. Section 2 introduces the formal problem model and associated terminology. Section 3 gives an overview of the BABOONS system. Section 4 describes the summary optimizer and its reinforcement learning model in more detail. Section 5 describes the system's proactive caching strategy. Section 6 discusses the sampling component while Section 7 provides details on the batch processor. Section 8 reports experimental results. Finally, Section 9 discusses prior work.

2 FORMAL MODEL

This section introduces terminology used throughout the paper.

Definition 2.1. We describe data in a **Data Source** $\mathcal{D}$, characterized by a relational **Table** ($\mathcal{D}.table$), a set of **Dimension Columns** ($\mathcal{D}.dims$), and a set of **Aggregation Columns** ($\mathcal{D}.aggs$). Aggregation columns must be of type integer or float (to enable aggregation via averaging). Dimension columns can be of any type. Aggregation and dimension columns may overlap. Additionally, the table may have columns that serve to identify items (discussed later).

Given a data source $\mathcal{D}$, we optimize over summary sketches.

Definition 2.2. A **Summary Sketch** S is a list $\langle t_1, \ldots, t_n \rangle$ of **Topics**. Each topic t is characterized by a **Data Scope**, $t.scope$, and an **Aggregate**, $t.agg$. The data scope is a set of equality predicates on dimension columns (i.e., $t.scope = \{\langle c_i, v_i \rangle\}$ with $c_i \in \mathcal{D}.dims$ and v_i is a value from the domain of c_i), representing their conjunction. The aggregate is one of the aggregation columns (i.e., $t.agg \in \mathcal{D}.aggs$). The generic term **Property** refers to both, predicates and aggregates of a topic.

We instantiate summary sketches for concrete items.

```
<Summary>::=<Fact>+
<Fact>::=<Preamble><Predicate>*<Aggregate>
```

Figure 1: Comparative summary structure in EBNF.

Definition 2.3. An **Item** i is characterized by an SQL predicate P_i that defines a data subset (the **Item Data**). Let a_I be the average in $t.agg$ over rows satisfying all scope predicates in $t.scope$ and the item predicate P_i. The item predicate may reference additional columns, beyond dimensions and aggregates, that serve to identify items. Let a_G be the average in $t.agg$ over rows satisfying all scope predicates. We call the triple $\langle t, i, a_I/a_G \rangle$ a **Fact**, instantiating topic t for item i by describing the relative average for the item within the topic scope.

Users restrict the search space for summaries by templates.

Definition 2.4. A **Summary Template** $\mathcal{T}$ specifies the number of facts ($\mathcal{T}.n_F$) as well as the maximal number of properties per fact ($\mathcal{T}.n_P$). It also specifies how abstract facts translate into natural language text by providing text snippets (with placeholders for values) for predicates ($\mathcal{T}.predtxt$) and aggregates ($\mathcal{T}.aggtxt$), as well as a preamble ($\mathcal{T}.preamble$), prefixing each fact text. For predicates, $\mathcal{T}.predtxt$ contains one text snippet for each dimension column (the snippet for dimension d is denoted by $\mathcal{T}.predtxt_d$). For aggregates, $\mathcal{T}.aggtxt$ contains one snippet for each aggregation column ($\mathcal{T}.aggtxt_a$ denotes the snippet for column a).

BABOONS outputs a summary in natural language.

Definition 2.5. A comparative **Summary** is the text representation that instantiates a summary sketch for a specific item. Figure 1 shows the corresponding grammar in Extended Backus-Naur Form (EBNF). A summary is a sequence of facts. Each fact text starts with the preamble ($\mathcal{T}.preamble$), optionally narrows the comparison scope via predicates, and finally reports a relative average. Each predicate restricting the value in column C to value V is represented by instantiating the text snippet $\mathcal{T}.predtxt_C$ by replacing a placeholder by V. Each aggregate on column A is represented using $\mathcal{T}.aggtxt_A$, replacing a placeholder by the relative average description (a rounded percentage, followed either by the keyword "lower" or "higher"). We evaluate summaries by a black-box **Model**. A model maps a text description to a numerical utility value.

We propose a system that solves the following problem.

Definition 2.6. An instance of **Data Summarization with Black-Box Utility** is defined by a triple $\langle \mathcal{D}, \mathcal{T}, \mathcal{M}, I \rangle$, where $\mathcal{D}$ is a data source, $\mathcal{T}$ a summary template, $\mathcal{M}$ a black-box utility model, and I a set of items. The goal is to find summaries for I, based on $\mathcal{D}$ and $\mathcal{T}$, that maximize mean utility over I according to $\mathcal{M}$.

3 SYSTEM OVERVIEW

Figure 2 shows an overview of the BABOONS system. Following Definition 2.6, the input consists of a set of items to compare, a text template for summary texts, a black-box utility function evaluating summaries, and a relational data source used to generate statistics. The output is a comparative summary that maximizes the score assigned by the utility model.

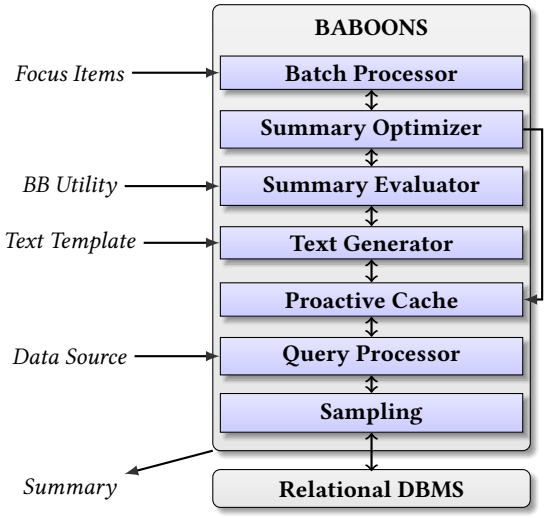

Figure 2: Overview of BABOONS system.

BABOONS generates facts about items using the Query Processor (using SQL queries). Facts compare aggregates for a data subset, associated with an item, to the entire data set. The Text Generator uses such facts as input to generate natural language data summaries (using simple text templates). The generated summaries are evaluated (by the Summary Evaluator component) via a user-provided utility function. This function is in general complex and considered a black box (e.g., the current implementation allows users specifying utility functions that are based on models for text analysis from Huggingface [65]).

The goal of the Optimizer is to find summaries maximizing the utility function. Internally, it uses deep reinforcement learning for optimization. This method is frequently used for black box optimization [36] (as approaches such as gradient descent are inapplicable). In the reinforcement learning model, an agent learns to apply actions to maximize rewards obtained in a (possibly) unknown environment. Here, the search space ("state space") is defined by summary sketches (defining data subsets and aggregates to consider). Actions change the summary sketches and rewards are proportional to utility values. Section 4 describes the Optimizer component in more detail.

Reinforcement learning is an iterative process, meaning that many summaries must be created and evaluated. One of the primary design goals of BABOONS is to minimize query evaluation overheads (thereby making summary optimization on large item and data sets practical). To increase efficiency, BABOONS uses proactive caching to reduce query processing overheads. It predicts likely future queries that can be merged cheaply with queries already scheduled for processing, caching their results. For large data sets, BABOONS can seamlessly reduce processing overheads via a customized sampling strategy (implemented in the Sampling component). In addition, BABOONS features a functionality for batch processing of large item sets. Sections 5, 6, and 7 describe those components in more detail.

4 THE SUMMARY OPTIMIZER

We treat the selection of data statistics for a comparative summary as a combinatorial optimization problem. The search space (i.e., the space of sketches) is discrete. The function to optimize is complex and we consider it a black box. Furthermore, if using data samples for evaluation (see Section 6), the utility function is subject to random noise (i.e., evaluating the same sketch twice may yield different results). We therefore use Reinforcement learning [55] which has been used successfully in similar scenarios [36].

Reinforcement learning applies to scenarios described as Markov Decision Processes (MDPs). We model summary optimization as an MDP $\langle \mathbb{S}, \mathbb{A}, \mathbb{T}, \mathbb{W}, \mathbb{O} \rangle$ where $\mathbb{S}$ is the state space, $\mathbb{A}$ the action set, $\mathbb{T}$ the transition function, $\mathbb{W}$ the reward function, and $\mathbb{O}$ the observation function. The state space represents summary sketches. It is a matrix whose dimensions are defined by the summary template $\mathcal{T}$: $\mathbb{S} = \mathbb{N}^{\mathcal{T}.n_F \cdot \mathcal{T}.n_P}$. Each integer value represents the ID of a property (i.e., a possible aggregate or a possible predicate). For predicates, there is a special ID value that represents the absence of a predicate (following Definition 2.4, users specify only the *maximal* number of properties per fact).

Each action changes the value of one field in the state matrix. We bound the branching factor of the search space by only allowing changes that replace a property by a *similar* property (we show that this restriction improves performance in Section 8).

Definition 4.1. We define **Similarity** between two properties as the cosine similarity of their embedding vectors, generated by encoding their textual representation via BERT [6].

We identify admissible changes via the **Property Graph**.

Definition 4.2. Given a data source $\mathcal{D}$, the **Property Graph** maps properties (i.e., aggregates and predicates) to a limited number of similar elements. It consists of two disconnected graph components associated with aggregates and predicates. Each node represents an aggregate from $\mathcal{D}.aggs$ or an equality predicate on one of the dimension columns $\mathcal{D}.dims$. The graph associated with predicates contains one node representing the absence of a predicate (the associated embedding vector used to calculate similarity between properties is the embedding of an empty string). The degree of the property graph is bounded by a constant d, limiting the number of outgoing edges per node.

The action space is defined by the space of triples from $\mathbb{A} = \{1, \ldots, \mathcal{T}.n_F\} \times \{1, \ldots, n_P\} \times \{1, \ldots, d\}$ where the first component selects the topic to change, the second component the property to change, and the last component selects the edge to follow in the property graph (starting from the node representing the current property). The definition of the transition function $\mathbb{T} : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{S}$ follows naturally as it maps a state and a state change to the state that results after the change. Finally, we map states to observations by concatenating the BERT [7] embedding vectors of all properties that appear in the current state.

Algorithm 1 is the algorithm executed by the optimizer. Given an instance of data summarization with black-box utility, it first creates the property graph (which requires processing data to determine all possible predicates). Next, it initializes a summary sketch and iteratively changes it until a timeout. In each iteration, it selects the next action to execute (i.e., a change to the current sketch) via reinforcement learning, More precisely, it uses a variant of the A3C (Asynchronous Advantage Actor-Critic) algorithm [40]. The change is executed and evaluated using the Step function, discussed in more detail in the following. The internal statistics of the learning algorithm are updated (Line 15) and influence choices in the next iterations. The result is the best sketch found until the timeout.

Algorithm 1 Algorithm executed by the summary optimizer.

```
1:  // Generate summaries for items I, using data D and text
2:  // template T, maximizing utility according to model M.
3:  function Optimize(D, T, M, I)
4:      // Generate property graph
5:      G ←PropertyGraph(D)
6:      // Initialize summary sketch
7:      S ←InitSketch(T)
8:      // Iteratively improve summary
9:      while no timeout do
10:         // Select change to summary sketch
11:         ⟨t, p, n⟩ ←ChooseAction(S)
12:         // Execute change and evaluate result
13:         ⟨S, r⟩ ←Step(D, T, M, G, S, t, p, n, I)
14:         // Update reinforcement learning model
15:         UpdateStats(S, r)
16:     end while
17:     return S
18: end function
```

Algorithm 2 Step function for reinforcement learning.

```
1:  // Given data D, template T, model M, and property
2:  // graph G, update sketch S by replacing p-th property
3:  // of t-th topic by n-th neighbor. Evaluate for items I.
4:  function Step(D, T, M, G, S, t, p, n, I)
5:      // Change property p of topic t
6:      S[t, p] ← G(S[t, p], n)
7:      // Iterate over items
8:      for i ∈ I do
9:          // Generate facts
10:         i.facts ←GetFacts(D, S, i)
11:         // Generate text
12:         i.text ←GetText(T, S, i.facts)
13:         // Evaluate text using NLP model
14:         i.eval ←Evaluate(M, i.text)
15:     end for
16:     // Return new sketch and reward
17:     return ⟨S, Mean({i.eval|i ∈ I})⟩
18: end function
```

Algorithm 2 shows the implementation of the step function. First, we execute the change described by the input action. We represent the current state as a two-dimensional matrix and use the bracket notation to access specific fields in it. We denote the n-th neighbor of property p in the property graph $\mathcal{G}$ as $\mathcal{G}(p, n)$. Next, we evaluate the changed sketch for all input items. For each item, we first obtain concrete facts for the current item and the sketch topics. Second, we translate those facts into a summary sketch (using the text templates in $\mathcal{T}$). Finally, the resulting text can be evaluated via the

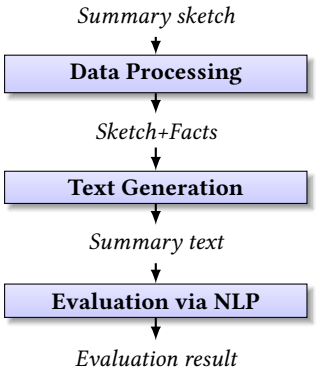

Figure 3: Steps when evaluating summary sketches.

Algorithm 3 Generate item-specific facts for sketch topics.

```
1:  // Using data D, generate facts for topics in sketch S on item i.
2:  function GetFacts(D, S, i)
3:      // Generate queries for each topic
4:      Q ← ∅
5:      for t ∈ S.topics do
6:          // Add query generating fact (simplified!)
7:          Q ← Q ∪ {select (select t.agg from D.table
8:           where SQL(t.scope) and SQL(i))/(select t.agg
9:           from D.table where SQL(t.scope))
10:     end for
11:     // Add results of uncached queries
12:     for q ∈ Q : ¬isCached(q) do
13:         AddToCache(q)
14:     end for
15:     // Retrieve all relevant facts from cache
16:     return {GetFromCache(q)|q ∈ Q}
17: end function
```

user-provided utility model. Figure 3 illustrates the steps taken to evaluate the current sketch.

Algorithms 3 describes the generation of facts from a relational data source in more detail. First, we generate one SQL query for each topic. We currently consider facts about relative averages, comparing item-related data to all data within the topic scope. We use the shortcut `SQL(t.scope)` for the conjunction of scope-related equality predicates. Note that the SQL query represented in Algorithm 3 is simplified, compared to the queries generated by the actual implementation. In particular, handling of special cases (e.g., null values and empty results) is omitted. The system caches query evaluation results to avoid re-evaluating the same query during the same optimization session. Function $isCached(q)$ verifies whether a result for query q is available. A query q is executed and its results added to the cache via the call AddToCache(q) and retrieved by GetFromCache(q).

5 CACHE MANAGER

BABOONS executes SQL queries to generate facts for the current summary. It caches query results to avoid re-executing the same queries. Proactive caching expands the scope for caching. Instead of caching results of queries that must be executed, the system proactively executes queries that may be needed in the future and caches their results. This is beneficial if generating additional query results increases execution cost only marginally. If anticipated queries occur indeed, their cached result can be used.

Algorithm 4 Query execution with proactive caching.

```
1:  // Process current queries Q and proactively cache likely
2:  // future queries under cost threshold α, given current
3:  // sketch S and property graph G.
4:  procedure ProCache(Q, S, G, α)
5:      // Calculate likelihood for future queries
6:      ⟨F, P⟩ ←FutureQueries(S)
7:      // Iterate over current queries
8:      for q ∈ Q do
9:          // Remove queries with cached result
10:         F ← {f ∈ F|¬isCached(f)}
11:         // Is query result in cache?
12:         if ¬isCached(q) then
13:             // Expand by integrating future queries
14:             e ←Expand(F, P, q, α)
15:             // Process expanded query
16:             AddToCache(e)
17:         end if
18:     end for
19: end procedure
```

BABOONS merges multiple queries of the type used in Algorithm 3 into single queries implementing the following template (the template is simplified by omitting handling of special cases such as NULL values or empty results):

```
select (itemSum/itemCnt)/generalAvg, dim1, dim2, ...
from (
 select avg(agg) as generalAvg,
  case when SQL(i) then 1 else 0 end as itemCnt,
  case when SQL(i) then t.agg else 0 end as itemSum,
  dim1, dim2, ...
 from T
 where SQL(scope1) or SQL(scope2) or ...
 group by dim1, dim2, ...
) as T
```

The query above calculates relative averages for one item and one aggregate but for multiple scopes that refer to the same dimension columns. Here, `agg` is the aggregation column, `dim1`, `dim2` etc. are dimension columns restricted by scope predicates, `SQL(i)` is the predicate identifying rows associated with an item, and `SQL(scope1)`, `SQL(scope2)` etc. are conjunctions of equality predicates representing the corresponding scope. The query uses the group-by clause to generate results for different scopes. It is more efficient than executing aggregation queries for each scope separately (e.g., as the input data is only read once). The extension to calculating multiple aggregates instead of one is natural.

Algorithm 4 replaces Lines 12 to 14 in Algorithm 3. Instead of executing only the queries that are currently needed, it expands the query scope to include likely future queries as well. The input

Algorithm 5 Query expansion for proactive caching.

```
1:  // Expand query q by covering possible future queries F with
2:  // probabilities P while increasing cost at most by factor α.
3:  function EXPAND(F, P, q, α)
4:      // Collect similar queries
5:      S ← {f ∈ F | f.dims = q.dims}
6:      // Collect and sort aggregates
7:      A ← {s.agg | s ∈ S} sort by rank(a) = Σ_{s∈S|s.agg=a} P(s)
8:      // Collect and sort conditions
9:      C ← {s.scope | s ∈ S} sort by rank(c) = Σ_{s∈S|s.scope=c} P(s)
10:     // Generate expanded query candidates
11:     E ← ∅
12:     for i ← 1, ..., |A| do
13:         // How many conditions can be considered?
14:         j ← arg max_{j∈1..|C|} C(A_1..A_i, C_1..C_j) ≤ q.cost · α
15:         // Add expanded query to candidates
16:         E ← E ∪ {CMPQUERY(A_1..A_i, C_1..C_j, q.item)}
17:     end for
18:     // Select expansion with highest probability
19:     return arg max_{e∈E} Σ_{s∈e} P(s)
20: end function
```

is the set of currently requested queries Q, the current sketch S, the property graph $\mathcal{G}$, and a cost threshold α. The algorithm expands the scope of queries until estimated processing costs reach a multiple of α, compared to the original query processing costs.

First, Algorithm 4 calculates a probability distribution over future queries via function FUTUREQUERIES (pseudo-code omitted). This calculation is based on the structure of the MDP, discussed in detail in Section 4, which is explored by the reinforcement learning algorithm. Each MDP state describes a summary sketch. Each sketch is associated with queries to evaluate (to calculate relative averages for each topic in the sketch). For a query f, denote by $\mathbb{S}_f$ the set of states that require evaluating that query. In that case, the probability $P(f)$ of having to answer query f in the next k steps of the reinforcement learning algorithm equals $\sum_{i=1..k} \sum_{s \in \mathbb{S}_f} P(s,i)$, where $P(s,i)$ designates the probability of reaching state s in i steps. To calculate $P(s,i)$, the system assumes that transitions are selected with uniform random probability (i.e., each transition from any given state is equally likely). It calculates probabilities for all states (and associated queries) that can be reached with up to $k = 2$ transformations from the current state. Unlike the probability distribution over transitions, the probability distribution over states and queries is not uniform. E.g., some queries are required by multiple states while others are not. While based on simplifying assumptions, the experiments in Section 8 reveal a relatively high cache hit rate. This indicates limited potential for improvement by more sophisticated (and, likely, more expensive) probabilistic models.

Next, Algorithm 4 iterates over queries that must be processed to evaluate the current sketch. For each of those queries, the algorithm tries to expand the query scope to cover likely future queries. The resulting, expanded query is processed and its result is cached (call to ADDTOCACHE in Line 16).

Algorithm 5 expands the scope of a query q, given anticipated future queries F with probabilities P and a cost threshold α for expansions. First, it collects future queries that are *similar* to query q. Similar queries restrict the same dimension columns as q ($q.dims$). Second, it collects aggregates and scope predicates from similar queries. It ranks aggregates and predicates by the accumulated probability of all queries using them. In principle, all combinations of aggregates and predicates could be considered for expansions. This may however lead to significant computational overheads. Instead, Algorithm 5 uses a simple heuristic: it considers aggregates in decreasing order of probability and, for each aggregate, uses the maximal number of predicates (while prioritizing likely predicates) under the cost constraint. This heuristic is simplifying as it is based on marginal probabilities of aggregates and predicates in separation (as opposed to probabilities for specific combinations of aggregates and predicates). Nevertheless, the experimental results indicate that opportunities for improvements in cache hit rate by more sophisticated expansion strategies are limited.

Table 4: Comparison of expansions for example query (cells contain probability in percent that associated topic becomes relevant). Optimal expansion is marked up in green.

Predicates	Price	Weight	Battery
Brand=IBM	100	8	12
Brand=Apple	6	9	7
Display=13"	3	4	
CPU=Intel	2	16	
Display=15"	5		

Example 5.1. Table 4 illustrates query expansions by an example. Aggregates are ordered from left to right in decreasing order of probability (probability to occur in future queries). Predicates are ordered from top to bottom in decreasing order of occurrence probability. Cells contain probabilities expressed as percentages (note that the probability for the current query, in the left upper corner of the table, is at 100%). As the number of considered aggregates increases, the number of predicates that can be processed under the cost budget shrinks (represented by the gray table area). We maximize the expected number of future queries covered by selecting two aggregates (which allows us selecting all but the last predicate). The associated table area is colored in green.

6 SAMPLING

BABOONS supports approximate processing to handle large data sets. If activated, the system selects summaries generated from data samples (as opposed to the full data set). The utility of those summaries is therefore an estimate. After optimization, a few summaries with high utility estimates are regenerated on the full data set. This avoids incorrect claims in the final summary (while sampling may lead to sub-optimal summaries being selected).

In a first step, the system uses a variant of the learning-based algorithm from Section 4 to optimize sketches. The main difference is that Function GETFACTS operates on a data sample, instead of the full data set. By default, the current implementation selects a sample containing at most 5% of the rows in the full data set. During optimization, several metrics are stored about each evaluated sketch. Based on those metrics, the system ranks all encountered sketches. For the top-k sketches (the current implementation selects

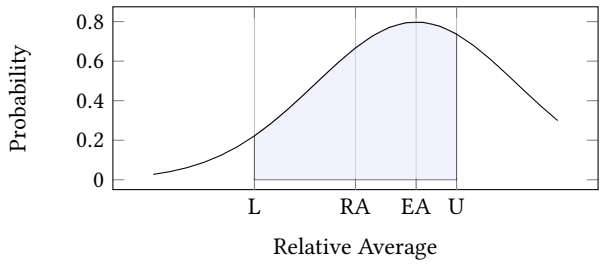

Figure 4: Based on the estimated average (EA), the summary text uses the rounded average (RA). The range of average values for which the text is accurate is colored and bounded by a lower (L) and upper bound (U).

the five highest ranking sketches), BABOONS processes the associated queries on the full data set. Finally, it evaluates the resulting advertisement texts and returns the optimum. Note that considering the single, highest ranking sketch alone is often sub-optimal. As utility results are based on data samples, they may change when re-generating statistics on the full data set. Next, we will see how to rank sketches and how to efficiently re-generate statistics for a batch of sketches.

When ranking sketches, the goal is to maximize expected quality according to the utility model. As facts are generated on data samples, the system does not obtain precise summaries and, as a consequence, no precise utility values. By bounding the probability for significant deviations between averages on samples and full data, it can however obtain estimates. Consider the summary text T_S for the current sketch S and its evaluation result E_S as random variables. They depend on random variables A_i and A_g, representing the item-specific and general average for the relevant aggregate on the data sample (we first assume a single item and aggregate before generalizing). Denote by a_i and a_g the values obtained for A_i and A_g on the current data sample. Further, let t_S and e_S be the text and utility value obtained with averages a_i and a_g.

Utility is calculated by a black-box function. It is unclear how even a small change of numbers in the input affects the final utility value. Therefore, the proposed model pessimistically assumes a utility value of zero if the final summary text does not match the current one. This means, it assumes $\mathbb{E}(E_S) = e_S \cdot \Pr(T = t)$ (where $\mathbb{E}$ denotes expected value). The summary text uses rounded values. Hence, if averages do not deviate too much from their sample values, neither the final advertisement text nor its utility will change.

Denote by U and L upper and lower bounds on the relative average within which the current text t is valid. If the relative average falls outside of that range, the summary text changes. Figure 4 illustrates the situation: the estimated average (EA), based on the data sample, is rounded to the one that appears in text (RA). As long as the real average falls between the bounds (L and U, marked up in blue), the rounded average does not change. Hence, expected quality becomes $\mathbb{E}(E_S) = e_S \cdot \Pr(L \leq A_i/A_g \leq U)$.

The relative, item-specific average is given as A_i/A_g. The first average, A_i, is based on rows within a topic scope that satisfy the item-related predicates. The second average, A_g, is based on all rows within the scope. Hence, confidence bounds calculated according to

Algorithm 6 Merge queries for highly ranked sketches.

```
1:  // Greedily merge group-by queries Q.
2:  function MERGE(Q)
3:      // Collect query templates
4:      T ← {⟨q.agg, q.dims⟩|q ∈ Q}
5:      // Partition queries by template
6:      P ← {{q ∈ Q|q.agg = a ∧ q.dims = d}|⟨a, d⟩ ∈ T}
7:      // Merge query partitions
8:      while |P| > 1 do
9:          // Find cost-optimal merge operation
10:         ⟨p1, p2⟩ ← arg min_{p1,p2∈P} C(p1 ∪ p2) − C(p1) − C(p2)
11:         // Does the merge improve cost?
12:         if C(p1 ∪ p2) − C(p1) − C(p2) > 0 then
13:             // Replace partitions by merged partition
14:             P ← (P \ {p1, p2}) ∪ {p1 ∪ p2}
15:         else
16:             // No improvement, return partitions
17:             return P
18:         end if
19:     end while
20:     return P
21: end function
```

formulas such as Hoeffding's inequality [16] will tend to be tighter for A_g (which is based on a larger sample of rows satisfying a weaker condition) than for A_i. Therefore, we simplify by assuming that $a_g = \mathbb{E}(A_g)$ is accurate while $\mathbb{E}(A_i)$ is uncertain. Hence, we obtain $\mathbb{E}(E_S) \approx e_S \cdot \Pr(L \leq A_i/a_g \leq U)$. We can upper-bound this probability using Hoeffding's inequality, assuming that A_i/a_g takes values from a bounded interval. The current implementation uses the interval $[0, 10]$, assuming that a deviation of more than one order of magnitude between item-specific and general average within a scope is unlikely. If a sketch contains multiple facts, the utility estimate is multiplied by the probability of each individual fact (thereby assuming independence).

The top sketches, according to the aforementioned formulas, are evaluated using the full data set. Knowing multiple sketches to evaluate a-priori creates opportunities to merge related queries. This scenario differs from the one of pro-active caching as there are no choices in terms of which queries are processed (and queries are not associated with probabilities). Algorithm 6 shows how queries are merged. It avoids complex approaches for cost-based query merging [51]. Such strategies can deal with diverse types of queries which is not required in this scenario. Instead, the algorithm uses a simple heuristic that is specialized to the types of queries required by BABOONS. We will see in the experiments that this heuristic leads to significant cost savings.

Algorithm 6 first partitions queries by their aggregates ($q.agg$) and by the dimension columns ($q.dims$) on which they place predicates. All queries in the same partition can be easily merged as demonstrated in Section 5. As the merged queries do not produce any unnecessary results, we always merge queries in the same partition. Merging queries from different partitions may lead to queries producing unnecessary result values. This creates a trade-off between the number of queries executed and their result sizes. Hence, we only merge partitions if it decreases estimated execution

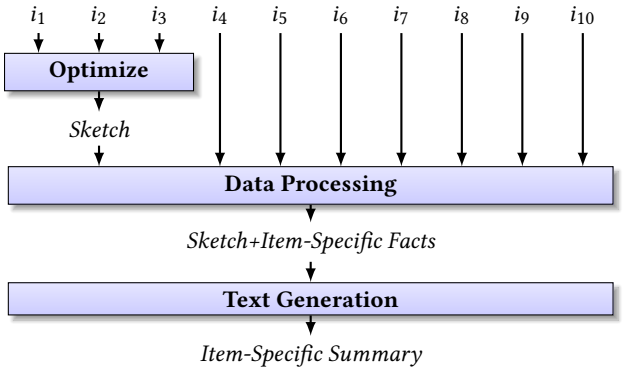

Figure 5: Generating summaries for item batches: BABOONS first selects a summary sketch for a small set of representative items, then instantiates that sketch for all items.

Algorithm 7 Use same sketch for entire item batch.

```
1:  // Given data D, template T, model M, and
2:  // property graph G, summarize data for items I,
3:  // using γ representatives when selecting sketch.
4:  function BatchProcess(D, T, M, G, I, γ)
5:      // Select representative items
6:      R ←RandomChoice(I, γ)
7:      // Select summary sketch for representatives
8:      S ←Optimize(D, T, M, G, R)
9:      // Generate facts for each item
10:     I.facts ←GetFactsBatch(D, S, I)
11:     // Generate summaries for each item
12:     I.text ←GetTextBatch(T, S, I.facts)
13:     // Evaluate each summary using utility model
14:     I.eval ←EvaluateBatch(M, I.text)
15:     return I
16: end function
```

costs (we use the cost model of the query optimizer of the underlying database system, represented by function C in pseudo-code). We merge partitions greedily, repeatedly selecting partition pairs whose merge leads to the largest cost savings. Iterations end once a single partition is left or once no immediate cost improvements can be achieved by any merge operation.

7 BATCH PROCESSOR

Figure 5 illustrates the first of two batch processing strategies. It simplifies optimization by selecting one common summary sketch to summarize all items in a batch. Selecting one sketch for the entire batch, based on utility results for a single item, is however risky. It may lead to topics that are highly item-specific and do not yield high utility for most of the items in the batch. As a compromise, the simple batch processing strategy randomly selects a small set of representative items. Then, it selects the sketch that maximizes the average utility among the representatives. Finally, it uses the best performing sketch to generate summaries for all items in the batch. In doing so, the approach balances overheads for iterative optimization against the quality of evaluations. Algorithm 7 shows the associated pseudo-code. It replaces Function GetFacts with Function GetFactsBatch, compared to Algorithm 1. This function merges queries for generating facts for all items, thereby increasing efficiency. Similarly, Function EvaluateBatch uses batch inference for increased efficiency (on GPU).

Algorithm 8 Iteratively select summaries with maximal marginal improvement of average summary quality.

```
1:  // Given data D, template T, model M, and
2:  // property graph G, summarize data for items I,
3:  // using β iterations and γ representatives per iteration.
4:  function BatchIterative(D, T, M, G, I, β, γ)
5:      // Select and evaluate one summary sketch for all items
6:      I ←BatchProcess(D, T, M, G, I, γ)
7:      // Iteratively improve item-specific summaries
8:      for i ∈ 1, ..., β do
9:          // Select sketch with maximal marginal improvement
10:         J ←BatchProcessMarginal(D, T, M, G, s, γ, I)
11:         // Retain best summary for each item
12:         I ←PruneSummaries(I, J)
13:     end for
14:     // Return best summary for each item
15:     return I
16: end function
```

Algorithm 8 uses Algorithm 7 to generate an initial summary for each item (all following the same sketch). Next, it selects an additional summary sketch in each iteration, using Function BatchProcessMarginal (pseudo-code omitted). This function works as BatchProcess with one difference: it aims at selecting the sketch with maximal marginal improvement over the summaries seen so far. The pseudo-code of Function BatchProcessMarginal equals the one of Function BatchProcess with two exceptions. First, Function BatchProcessMarginal has one additional parameter, storing the best previously generated summary for each item. Second, it uses the optimizer (invocation in Line 8 of Algorithm 7) with a modified reward function. For each representative item considered during optimization, reward is proportional to the utility improvement over the best previously known summary (and zero if the new summary is inferior). The overall reward is the arithmetic average over the reward for each of the γ representative items. After generating new summaries according to the selected sketch, summaries are pruned via Function PruneSummaries (pseudo-code omitted). Given two vectors of item summaries (parameters I and J), this function iterates over items and returns a vector containing the best summary for each item. While seemingly simple, Algorithm 8 guarantees near-optimal summaries, assuming that the reinforcement learning optimizer identifies the locally optimal summary in each iteration. A formal proof of this statement is given in Appendix A.

8 EXPERIMENTAL EVALUATION

Section 8.1 describes the experimental setup. Section 8.2 reports performance results for BABOONS in different configurations and compares to baselines. Finally, Section 8.3 evaluates the quality of summaries generated by different approaches by surveys.

Table 5: Overview of benchmark data sets.

Data Set	Size	#Rows	#Columns	Items
Laptops [22]	30 KB	200	8	134
Developers [23]	10 MB	90 K	14	28
Flights [24]	900 MB	7 M	26	18
Sales [25]	5 GB	20 M	8	8706

8.1 Experimental Setup

The following experiments simulate use cases in advertisement. The goal is to advertise items in different categories. The utility function is implemented by a Roberta model [34], fine-tuned for sentiment analysis on the MNLI benchmark [1]. This model predicts either positive or negative sentiment, together with a confidence score. The confidence score, multiplied by +1 for positive and -1 for negative predicted sentiment, is used as utility value.

This model has not been specialized for evaluating advertisement text. However, the main contribution in this paper is a framework that optimizes data summaries according to generic black box functions. Improving quality of utility measures is an orthogonal research goal. Also, in Section 8.3, we will see that scores assigned by the generic model correlate well with opinions of actual users.

The experiments use four data sets, obtained from Kaggle. Table 5 shows an overview of those data sets. The first data set contains information on laptop models (e.g., price, disk space, and screen size). Here, the goal is to advertise specific models by pointing out advantages to the others. The second data set contains results of a large developer survey. Here, the goal is to advertise courses for specific programming languages, based on arguments comparing developers who know the language versus others (e.g., in terms of average salary). The third data set contains information on flight delays and cancellations. Here, the goal is to advertise specific airlines by comparing statistics (e.g., on average delay in specific regions or seasons) to those of competitors. The final data set contains information on product sales. The goal is to advertise specific products to shop owners by comparing sales numbers or revenue, possibly in specific regions, to other products. Each column was associated with a short text template, used for text generation (e.g., "from [X]" for a column containing flight start airports where [X] is replaced by the column value).

BABOONS is implemented in Python 3. It uses Postgres (version 12) as relational processing engine and the Huggingface Transformers library [66] for NLP. The property graph has degree five ($d = 5$). Proactive caching is tuned for a maximal cost increase of up to 50% for query expansions ($\alpha = 1.5$). When using sampling, the five highest ranked sketches are selected for evaluation on the full data set. In batch processing, $\gamma = 3$ items are used as representatives per partition and partitions are split five times ($\beta = 5$). The A2C implementation of the stable-baselines library [44] (Version 3) is used as reinforcement learning algorithm. Unless noted otherwise, 200 learning steps are executed per test case.

Prior work on optimizing data summaries often places constraints on the type of utility function (see Section 9). Here, the utility function is represented by a large neural network, rendering most prior work inapplicable. The approach by Ziegler et al. [74] is one of the few prior methods that are applicable. It uses reinforcement learning as well and has been used for tasks similar to the ones evaluated here (e.g., generating text summaries that maximizes evaluation by another neural network). It is used as baseline in the following (denoted as "G" in the plots). The "prompt" (i.e., a text preamble that influences subsequent text generation) consists of 10 randomly selected facts (generated by the query processor of BABOONS). This enables the baseline to exploit facts about the current item. The reward function is the same as for BABOONS. Also, "r" and "R" denote two randomized baselines in the following figures. Baseline R iteratively generates and evaluates random fact combinations, using the same amount of time per test case as BABOONS. Baseline r only draws one single random sample. Finally, "V" designates a baseline that uses the Google Vizier platform [12] with default parameter settings. This platform offers blackbox optimization as a service. The baseline models summaries via integer parameters (one parameter per fact and per aggregate or predicate slot, representing no predicate by a value of zero), trying out value combinations suggested by Vizier and reporting back to Vizier on the quality of resulting summaries.

All experiments were executed on a p3.2xlarge instance on the Amazon EC2 Cloud platform. This instance features a Tesla V100 GPU, 61 GB of main memory, and 8 vCPUs. Ubuntu 18 was used as operating system.

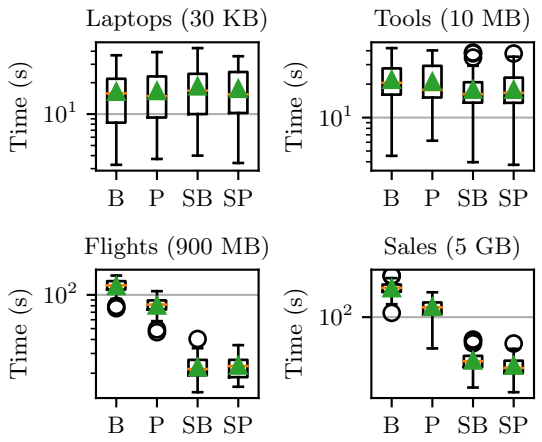

Figure 6: Performance of BABOONS with different configurations.

8.2 Performance Evaluation

First, we compare different configurations of BABOONS. The base configuration ("B") deactivates proactive caching (as well as sampling and batch processing). The configuration "P" uses proactive caching but no sampling, "SB" uses sampling but no proactive caching, and "SP" activates proactive caching and sampling at the same time. Note that all configurations use "reactive" caching (i.e.,

[1] https://huggingface.co/siebert/sentiment-roberta-large-english

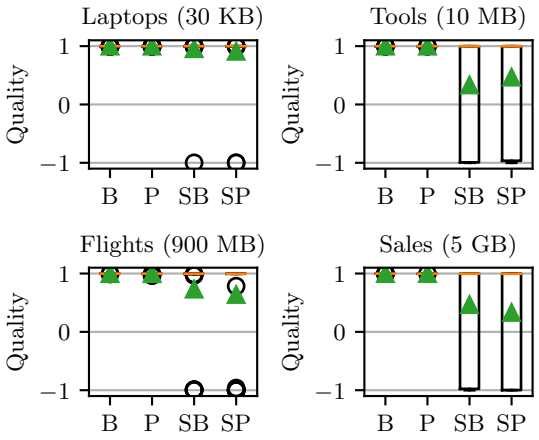

Figure 7: Output quality (utility) of BABOONS with different configurations.

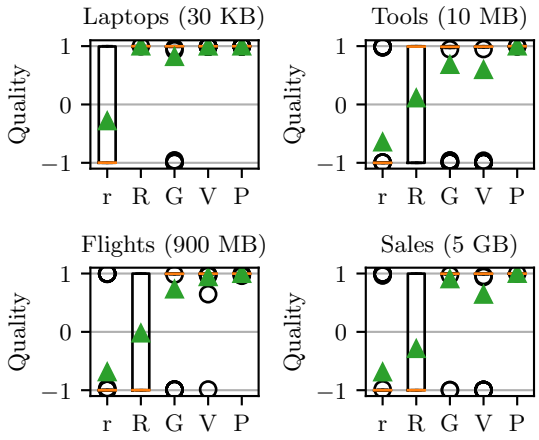

Figure 8: Comparison of output quality for different summarization methods.

each possible query is processed at most once). Figure 6 shows boxplots for average times for different approaches and scenarios (note the logarithmic y-axis). In this and the following figures, each data point corresponds to runs for five randomly selected items. Green triangles mark the arithmetic average while horizontal, orange lines mark the median.

Clearly, both, sampling and proactive caching improve performance for large data sets. Proactive caching is able to predict future queries in most cases, thereby increasing efficiency. It achieves an average cache hit rate of 86% among new queries (i.e., queries that were not encountered before). Hence, the query expansion heuristic presented in Algorithm 5 is already close to the optimum.

Sampling improves performance further but comes with tradeoffs. Figure 7 compares output quality, measured by the language model as a value between -1 and +1, of different configurations. Unlike proactive caching, sampling decreases average output quality (less so for the median). While sampling does not always find optimal summaries, the generated summaries are always accurate. This becomes possible since the highest ranking sketches are processed on the entire data set. Merging queries according to Algorithm 6 decreases processing costs by factor seven in average. Using proactive caching, in combination with sampling, improves performance further. When omitting search space clustering by embedding vectors (see Section 4), average quality decreases by over one percent over all scenarios.

The following experiments scale up the number of items. The goal is to generate summaries for all of the 8,706 items in the largest data set. Nine summaries are generated for each item, considering between one and three facts per summary and between one and three predicates per fact. Averaging over all items and over test cases with different number of predicates, Algorithm 8 achieves utility values of 0.23, 0.17, and 0.03 for summaries with one fact, two facts, and three facts after five iterations. Given the same amount of time, the sampling baseline only generates summaries for 2.2% of items. Also, average utility is worse (-0.96, -0.95, and -0.99 for summaries with one to three facts). Compared to Algorithm 8, Algorithm 7 is about five times faster (it performs only a single iteration) but achieves utility values of only -0.15, -0.31, and -0.28 for summaries with one to three facts respectively.

Table 6: Analysis of results by generative baseline.

Scenario	1F, 1P	2F, 2P	3F, 3P
Laptops	No Statistic	No Statistic	No Statistic
Tools	No Statistic	No Statistic	No Statistic
Flights	Wrong Statistic	No Statistic	No Statistic
Sales	No Statistic	Wrong Statistic	No Statistic

Figure 8 compares BABOONS (with proactive caching) to baselines in terms of output quality (see Section 8.1 for a description of the baselines). All baselines, except for single random selection (r), are iterative. To ensure a fair comparison, each iterative baseline is allocated the same time as taken by BABOONS for each test case. For all but the smallest data set, iterative random selection has a high quality variance and a low average quality (the same is true for single random selection). BABOONS achieves the highest average quality over all scenarios.

Vizier (V) achieves the second highest average quality in two of the four scenarios, the generative baseline (G) in the two other ones. Compared to Vizier, BABOONS finds summaries with higher quality for 83% of test cases (considering all scenarios). For 10% of test cases, Vizier generates summaries with negative quality values (this does not happen for BABOONS). At the time of writing, Vizier charges 1 USD per test case[2]. This corresponds to a cost increase of approximately factor 20, compared to the cost of running the EC2 instance used by BABOONS (given an average time of 61 seconds per test case, the associated cost is 5 cents).

The generative approach is the only baseline not guaranteed to generate truthful summaries. Table 6 reports results of a manual verification. The goal of verification is to determine whether facts

[2]https://cloud.google.com/vertex-ai/pricing#vizier

Table 7: Average rank of summaries generated by baselines.

	r	R	P
Survey Rank	3.5	2.25	2
Model Rank	3.5	1.5	1.25

in the first summary in each scenario are accurate (by running corresponding SQL queries). The baseline succeeds at generating text that maximizes utility according to the language model. However, most of the generated text does not contain verifiable statistics. Also, whenever concrete statistics are specified, they typically do not match the data (this effect is referred to as "hallucination" [74]).

8.3 Survey

Table 7 shows aggregate results of a survey, comparing summaries generated by different baselines (single random selection, iterative random selection, and BABOONS with proactive caching). The survey is based on the Google Surveys [20] platform. All summaries use one fact and up to two predicates per fact. For each of the four scenarios (laptops, tools, flights, and sales), participants rate summaries for the same item on a scale from one to five, according to scenario-specific criteria. For instance, in case of laptops, participants rate whether a summary motivates them to buy the advertised laptop. Table 7 reports the arithmetic average rank of summaries generated by each baseline, averaging over all scenarios. The table reports rank based on the model-based utility estimates as well as ranks based on actual user ratings. The results use answers from participants who evaluated each summary in a specific scenario: 79 participants in the laptops scenario, 68 participants for both flights and tools, and 62 participants for sales. The inter-rater agreement according to the Kappa metric [46] is 0.36 for laptops, 0.73 for tools, 0.48 for flights, and 0.58 for sales (i.e., ranging from "fair" to "substantial" according to recommended terminology [30]). BABOONS ranks best according to estimates and actual user replies.

Table 8 compares summaries generated for the same item by BABOONS and by the Vizier baseline, reporting the model-based quality estimate. The latter baseline performs best among all baselines generating accurate results (the generative baseline generates incorrect summaries and is not considered in the surveys). Figure 9 reports results of a corresponding survey. This survey uses the Amazon Mechanical Turk (AMT) platform to enable direct comparisons between summaries generated by both approaches (Google Surveys imposes a limit of 175 characters per question, disabling this option). Crowd workers select the preferred summary, comparing summaries by both approaches for the same item. The survey covers five randomly selected items for each of the four scenarios. Crowd workers receive five cents per task. Only crowd workers with the "Master" certificate (indicating highly reliable performance) are eligible. Figure 9 reports votes (on which summary is best) for each test case (crowd workers can only vote once per test case). The inter-rater agreement, using the same metric as before, is 0.23 (indicating "fair" agreement [30]). BABOONS generates preferred summaries for 75% of test cases (and the majority in each scenario).

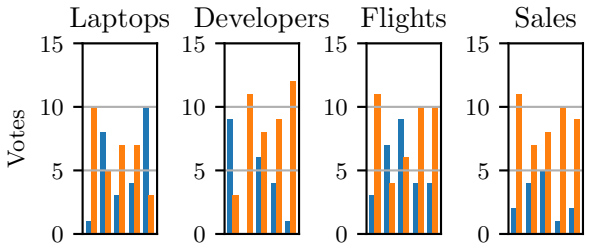

Figure 9: Votes by crowd workers, comparing summaries generated by Vizier (blue) and BABOONS (orange, hashed).

A final experiment demonstrates that BABOONS can indeed use diverse utility functions. For instance, BABOONS can evaluate data summaries in terms of their similarity to a user-defined communication goal. The following experiment focuses on summaries for five randomly selected items from the sales data set. Summary quality is measured as the entailment score between the summary and a natural language communication goal. The score is calculated by a large BART language model [32], trained on the MNLI benchmark [64] (which includes comparisons between sentence pairs). Four communication goals are used, namely "advertisement for customers with small budget", "advertisement for shop keepers who want to maximize sales", "advertise low cost", "advertise high sales". Using proactive caching and the same settings as before, BABOONS generates for instance the summary "Among all liquors , the dollar value per sale is 49% lower than average." with the first communication goal and the summary "Among all liquors sold in Waukee, the number of bottles per sale is 5% higher than average." with the second communication goal for the same item. A small AMT study asked crowd workers with Master certificate whether or not the generated summary satisfies the given communication goal (paying five cents per question). Out of 20 test cases (five items per communication goal), the majority of crowd workers voted "yes" for 18 test cases. Between 16 and 19 answers were received per test case with an average inter-rater agreement of 0.34 (fair).

9 RELATED WORK

The work presented here falls into the broad domain of data-to-text generation [48]. Corresponding approaches derive a textual representation from structured data such as time series [10, 11, 52], tabular data [31, 37], or graphs [26, 57]. BABOONS differs from most work in this domain by addressing efficiency issues when summarizing large data sets (by mechanisms such as caching and sampling). Unlike many recent publications on the topic [3, 35, 38, 41, 45, 72], BABOONS does not require a training corpus that contains data with associated summaries.

BABOONS composes summaries of carefully selected fact sets. In that, it connects to prior work focused on selecting interesting data subsets and aggregates for visualization [50, 56, 61, 67] or for voice output [58–60]. It connects to prior work on finding exceptional facts about entities [4, 14, 18, 54, 69–71, 73]. Prior work typically places restrictions on the utility function (e.g., entropy-based [50], distance-based [58], or monotone [69] utility functions) and exploits

Table 8: Comparison of summaries (slightly shortened) describing the same item with close quality estimates.

Method	Quality	Text
V	0.9983	Among all laptops with Intel UHD Graphics 620 graphics card its discounted price is 8% lower than average.
P	0.9987	Among all laptops its discounted price is 13% lower than average.
V	0.9500	Among all developers who used Bash/Shell/... as programming languages, the salary is about average.
P	0.9982	Among all developers who answered "Yes" when asked if they care for dependents who answered "No" when asked if they feel optimistic, the salary is 17% higher than average.
V	0.9983	Among all flights scheduled to arrive at 847 scheduled to arrive at 847 , the air time is 25% lower than average.
P	0.9984	Among all flights to RAP on 2018-05-19, the taxi time at departure is 62% lower than average.
V	0.9979	Among all liquors sold in Afton, the dollar value per sale is 4% higher than average.
P	0.9984	Among all liquors sold at Shade Tree Liquors, the dollar value per sale is 14% higher than average.

them for pruning. BABOONS does not place any restrictions on the type of utility function and treats it as a black box. It focuses on relational data as opposed to knowledge graphs [71, 73]. BABOONS also connects to prior work on generating text according to complex preference functions [53, 74]. Section 8 compares BABOONS against a corresponding baseline.

BABOONS connects to prior work by the techniques it uses for efficient processing. For instance, the idea of proactive caching appears in various variants [17, 47]. The realization in BABOONS differs by the mechanism used for proactive caching (query expansions) as well as by the selection method (predicting future queries based on the search space structure). Sampling is a popular method to reduce computational overheads [15, 19, 33]. However, the use of sampling in BABOONS differs from prior work by the ranking function (sensitivity of generated text to changes in the relative average) as well as by the processing context (use of sampling to select sketches, followed by query processing on the full data set).

BABOONS selects data subsets for comparisons. In that, it relates to prior work on other problems that involve careful selection of data subsets. Prior work on query result explanation often represents explanations as composite predicates [2, 49, 68], identifying rows with significant impact on results. Instead, the space searched by BABOONS consists of fact combinations where each single fact is defined by a combination of a predicate and an aggregate. Data summarization often refers to the problem of selecting data items to maximize a utility function [13, 39, 62, 63]. Often, work in this domain assumes a sub-modular utility function, motivating the use of greedy algorithms for sub-modular optimization [42]. BABOONS, however, does not assume a sub-modular utility function. It selects aggregate facts, rather than data items. Broadly, BABOONS relates to prior work that analyses or transforms data with the goal of maximizing black-box utility functions. This includes, in particular, recent work on data cleaning with complex or black-box utility functions [9, 28, 29, 43, 62]. Typically, the goal is to select data or cleaning operations to maximize performance of downstream applications such as model training. BABOONS differs from all the aforementioned work by its problem model and the design decisions that derive from it.

10 CONCLUSION

BABOONS automatically generates and selects statistics that optimize an arbitrary utility function. The experiments demonstrate that the approach is efficient and effective.

A FORMAL ANALYSIS OF BATCH PROCESSOR

Algorithm 8 selects one summary sketch in each iteration. Denote by $\mathbb{U} : \mathbb{S} \rightarrow \mathbb{R}$ the function mapping the set $\mathbb{S}$ of selected summary sketches to the average utility over all items (considering for each item the best generated summary which Algorithm 8 retains), i.e. $\mathbb{U}(\mathbb{S}) = Mean(\{\max(\{u_s^i | s \in \mathbb{S}\}) | i \in I\})$ where I is the set of items and u_s^i the utility of the summary sketch s for item i.

Theorem A.1. *Function $\mathbb{U}$ is submodular.*

Proof. For a single item i, Function $\mathbb{U}(\mathbb{S})$ is $\max(\{u_s^i | s \in \mathbb{S}\})$ and the set maximum is submodular (see Page 6 of Ref. [8]). For multiple items, $\mathbb{U}$ is a linear combination of submodular functions with positive weights, therefore submodular [27]. □

The next theorem assumes that Algorithm 8 finds the sketch that maximizes the improvement in average utility in each iteration. We assume that the worst utility value, realized e.g. by an empty summary, is zero (this can be achieved by adding a constant if a lower bound on negative utility values is known).

Theorem A.2. *Algorithm 8 achieves utility of at least $u^* \cdot (e-1)/e$ where u^* is the optimal utility for the selected number of sketches.*

Proof. According to Theorem A.1, average utility over all items is submodular in the set of sketches selected by Algorithm 8. Furthermore, average utility is non-decreasing (since Algorithm 8 prunes sub-optimal summaries for each item) and assumed non-negative. Hence, as Algorithm 8 selects the sketch with maximal marginal improvement of average utility in each iteration, it achieves the postulated guarantees [42]. □

REFERENCES

[1] 2021. https://www.upcounsel.com/what-is-a-product-description.

[2] Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Ananthanarayan, John Sheu, Erik Meijer, Xi Wu, Jeff Naughton, Peter Bailis, and Matei Zaharia. 2021. DIFF: a relational interface for large-scale data explanation. *VLDB Journal* 30, 1 (2021), 45–70. https://doi.org/10.1007/s00778-020-00633-6

[3] Mohiuddin Ahmed. 2019. Data summarization: a survey. *Knowledge and Information Systems* 58, 2 (2019), 249–273. https://doi.org/10.1007/s10115-018-1183-0

[4] Fabrizio Angiulli, Fabio Fassetti, and Luigi Palopoli. 2009. Detecting outlying properties of exceptional objects. *ACM Transactions on Database Systems* 34, 1 (2009). https://doi.org/10.1145/1508857.1508864

[5] B. Buchanan and D. Goldman. 1989. Us vs. them: the minefield of comparative ads. *Harvard Business Review* 67, 3 (1989), 38–40, 42, 44 passim.

[6] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, Vol. 1. 4171–4186. arXiv:1810.04805

[7] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, Vol. 1. 4171–4186. arXiv:1810.04805

[8] Charanpal Dhanjal and Stéphan Clémencon. 2011. Maximising the quality of influence. *Proceedings of the 11th SIAM International Conference on Data Mining, SDM 2011* (2011), 956–967. https://doi.org/10.1137/1.9781611972818.82

[9] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning Matthias. In *NEURIPS*. 1–9.

[10] Dimitra Gkatzia, Helen Hastie, Srinivasan Janarthanam, and Oliver Lemon. 2013. Generating student feedback from time-series data using reinforcement learning. *ENLG 2013 - 14th European Workshop on Natural Language Generation, Proceedings* (2013), 115–124.

[11] Eli Goldberg, Norbert Driedger, and Richard Kittredge. 1994. Using natural-language processing to produce weather forecasts. *IEEE Expert-Intelligent Systems and their Applications* 9, 2 (1994), 45–53. https://doi.org/10.1109/64.294135

[12] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. 2017. Google vizier: A service for black-box optimization. In *SIGKDD*, Vol. Part F1296. 1487–1496. https://doi.org/10.1145/3097983.3098043

[13] Kai Han, Shuang Cui, Tianshuai Zhu, Enpei Zhang, Benwei Wu, Zhizhuo Yin, Tong Xu, Shaojie Tang, and He Huang. 2021. Approximation Algorithms for Submodular Data Summarization with a Knapsack Constraint. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 1 (2021), 1–31. https://doi.org/10.1145/3447383

[14] Naeemul Hassan, Afroza Sultana, You Wu, Gensheng Zhang, Chengkai Li, Jun Yang, and Cong Yu. 2014. Data in, fact out: Automated monitoring of facts by FactWatcher. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1557–1560. https://doi.org/10.14778/2733004.2733029

[15] Joseph M. JM Hellerstein, PJ Peter J. Haas, and HJ Helen J. Wang. 1997. Online aggregation. *SIGMOD Record* 26, 2 (1997), 171–182. https://doi.org/10.1145/253262.253291

[16] W Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association* 58, 301 (1963), 13–30. http://onlinelibrary.wiley.com/doi/10.1002/0470011815.b2a17080/fullhttp://amstat.tandfonline.com/doi/abs/10.1080/01621459.1963.10500830

[17] Haibo Hu, Jianliang Xu, Wing Sing Wong, Baihua Zheng, Dik Lun Lee, and Wang Chien Lee. 2005. Proactive caching for spatial queries in mobile environments. *Proceedings - International Conference on Data Engineering* Icde (2005), 403–414. https://doi.org/10.1109/ICDE.2005.113

[18] Xiao Jiang, Chengkai Li, Ping Luo, Min Wang, and Yong Yu. 2011. Prominent streak discovery in sequence data. In *SIGKDD*. 1280–1288. https://doi.org/10.1145/2020408.2020601

[19] Shantanu Joshi and Christopher Jermaine. 2008. Materialized sample views for database approximation. *ICDE* 20, 3 (2008), 337–351. https://doi.org/10.1109/TKDE.2007.190664

[20] Surveys June and White Paper. 2018. White Paper: How Google Surveys Works. June (2018).

[21] Juraj Juraska and Marilyn Walker. 2021. Attention Is Indeed All You Need: Semantically Attention-Guided Decoding for Data-to-Text NLG. *INLG 2021 - 14th International Conference on Natural Language Generation, Proceedings* September (2021), 416–431. arXiv:2109.07043

[22] Kaggle. 2019. https://www.kaggle.com/ghadahalshehrei/laptops-info.

[23] Kaggle. 2019. https://www.kaggle.com/itrummer/stack-overflow-developer-survey-voice-interface.

[24] Kaggle. 2019. https://www.kaggle.com/yuanyuwendymu/airline-delay-and-cancellation-data-2009-2018.

[25] Kaggle. 2020. https://www.kaggle.com/sibmike/iowaliquorsales2020.

[26] Zdeněk Kasner and Ondřej Dušek. 2020. Data-to-Text Generation with Iterative Text Editing. In *INLG*. 60–67. arXiv:2011.01694

[27] Andreas Krause and D Golovin. 2012. *Submodular function maximization.* Technical Report. http://las.ethz.ch/files/krause12survey.pdf

[28] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, and Eugene Wu. 2017. BoostClean: Automated Error Detection and Repair for Machine Learning. (2017). arXiv:1711.01299 http://arxiv.org/abs/1711.01299

[29] Sanjay Krishnan and Eugene Wu. 2019. AlphaClean: Automatic Generation of Data Cleaning Pipelines. July 2017 (2019). arXiv:1904.11827 http://arxiv.org/abs/1904.11827

[30] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159. https://doi.org/10.2307/2529310

[31] Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *EMNLP*. 1203–1213. https://doi.org/10.18653/v1/d16-1128 arXiv:1603.07771

[32] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. (2020), 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703 arXiv:1910.13461

[33] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. *SIGMOD* 46, 1 (2016), 615–629. https://doi.org/10.1145/2882903.2915235

[34] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv* 1 (2019). arXiv:1907.11692

[35] Joy Mahapatra and Utpal Garain. 2021. Exploring Structural Encoding for Data-to-Text Generation. In *INLG*. 404–415.

[36] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. 2021. Reinforcement learning for combinatorial optimization: A survey. *Computers and Operations Research* 134 (2021). https://doi.org/10.1016/j.cor.2021.105400 arXiv:2003.03600

[37] Kathleen McKeown, Jacques Robin, and Karen Kukich. 1995. Generating concise natural language summaries. *Information Processing and Management* 31, 5 (1995), 703–733. https://doi.org/10.1016/0306-4573(95)00026-D

[38] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. What to talk about and how? Selective generation using LSTMs with coarse-to-fine alignment. In *NAACL*. 720–730. https://doi.org/10.18653/v1/n16-1086 arXiv:1509.00838

[39] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. 2016. FAst coNsTrained submodular maximization: Personalized data summarization. *33rd International Conference on Machine Learning, ICML 2016* 3 (2016), 2042–2054.

[40] Volodymyr Mnih, Adria Puigdomenech Badia, Lehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016* 4 (2016), 2850–2869. arXiv:1602.01783

[41] Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019. Step-by-step: Separating planning from realization in neural data-to-text generation. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference* 1 (2019), 2267–2277. https://doi.org/10.18653/v1/N19-1236 arXiv:1904.03396

[42] GL Nemhauser and LA Wolsey. 1978. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research* 3, 3 (1978), 177–188. http://mor.journal.informs.org/content/3/3/177.short

[43] Felix Neutatz, Binger Chen, Ziawasch Abedjan, and Eugene Wu. 2021. From Cleaning before ML to Cleaning for ML. *Data Engineering* March (2021), 24.

[44] OpenAI. 2021. https://stable-baselines3.readthedocs.io/en/master/.

[45] Ratish Puduppully, Li Dong, and Mirella Lapata. 2019. Data-to-text generation with content selection and planning. In *AAAI*. 6908–6915. https://doi.org/10.1609/aaai.v33i01.33016908 arXiv:1809.00582

[46] Justus J. Randolph. 2005. Free-Marginal Multirater Kappa (multirater K[free]): An Alternative to Fleiss' Fixed-Marginal Multirater Kappa. *Joensuu Learning and Instruction Symposium* (2005). http://eric.ed.gov/?id=ED490661

[47] Weixiong Rao, Lei Chen, Ada Wai Chee Fu, and Yingyi Bu. 2007. Optimal proactive caching in peer-to-peer network: Analysis and application. In *International Conference on Information and Knowledge Management, Proceedings*. 663–672. https://doi.org/10.1145/1321440.1321533

[48] Ehud Reiter and Robert Dale. 1997. Building Applied Natural Language Generation Systems. *Natural Language Engineering* 3, 1 (1997), 57–87. https://doi.org/10.1017/S1351324997001502

[49] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2014), 1579–1590. https://doi.org/10.1145/2588555.2588578

[50] S. Sarawagi. 2000. User-adaptive exploration of multidimensional data. In *VLDB*. 307–316. http://citeseer.ist.psu.edu/sarawagi00useradaptive.html

[51] Timos Sellis and Subrata Ghosh. 1990. On the multiple-query optimization problem. *KDE* 2, 2 (1990), 262–266. https://doi.org/10.1109/69.54724

[52] Pranay Kumar Venkata Sowdaboina, Sutanu Chakraborti, and Somayajulu Sripada. 2014. Learning to summarize time series data. In *LNCS*, Vol. 8403 LNCS. 515–528. https://doi.org/10.1007/978-3-642-54906-9_42

[53] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2020. Learning to summarize from human feedback. *Advances in Neural Information Processing Systems* 2020-Decem, NeurIPS (2020), 1–45. arXiv:2009.01325

[54] Afroza Sultana, Naeemul Hassan, Chengkai Li, Jun Yang, and Cong Yu. 2014. Incremental discovery of prominent situational facts. *Proceedings - International Conference on Data Engineering* December 1992 (2014), 112–123. https://doi.org/10.1109/ICDE.2014.6816644 arXiv:1311.4529

[55] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement learning, second edition: An introduction*. 532 pages. https://doi.org/10.1016/s1364-6613(99)01331-5 arXiv:1603.02199

[56] Bo Tang, Shi Han, Man Lung, Yiu Rui, and Ding Dongmei. 2017. Extracting Top-K Insights from Multi-dimensional Data. In *SIGMOD*. 1509–1524.

[57] Ngan Thi Dong. 2013. *Natural language generation from graphs*. Ph.D. Dissertation. https://doi.org/10.2307/414915

[58] Immanuel Trummer and Anderson Connor. 2021. Optimally summarizing data by small fact sets for concise answers to voice queries. In *ICDE*. 1715–1726.

[59] Immanuel Trummer, Yicheng Wang, and Saketh Mahankali. 2019. A holistic approach for query evaluation and result vocalization in voice-based OLAP. In *SIGMOD*. 936–953.

[60] Immanuel Trummer, Jiancheng Zhu, and Mark Bryan. 2017. Data vocalization: optimizing voice output of relational data. *PVLDB* 10, 11 (2017), 1574–1585.

[61] Manasi Vartak, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. 2014. SeeDB: automatically generating query visualizations. *VLDB* 7, 13 (2014), 1581–1584. https://doi.org/10.14778/2733004.2733035

[62] Tianhao Wang, Yi Zeng, Ming Jin, and Ruoxi Jia. 2021. A Unified Framework for Task-Driven Data Quality Management. (2021), 1–20. arXiv:2106.05484 http://arxiv.org/abs/2106.05484

[63] Yanhao Wang, Yuchen Li, and Kian Lee Tan. 2019. Efficient Representative Subset Selection over Sliding Windows. *IEEE Transactions on Knowledge and Data Engineering* 31, 7 (2019), 1327–1340. https://doi.org/10.1109/TKDE.2018.2854182 arXiv:1706.04764

[64] Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference* 1 (2018), 1112–1122. https://doi.org/10.18653/v1/n18-1101 arXiv:1704.05426

[65] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick Von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M Rush. 2020. Transformers : State-of-the-Art Natural Language Processing. (2020), 38–45.

[66] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *EMNLP*. 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6 arXiv:arXiv:1910.03771v5

[67] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2015. Voyager: exploratory analysis via faceted browsing of visualization recommendations. *Transactions on Visual and Computer Graphics* 22, 1 (2015), 649–658.

[68] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment* 6, 8 (2013), 553–564. https://doi.org/10.14778/2536354.2536356

[69] Tianyi Wu, Dong Xin, Qiaozhu Mei, and Jiawei Han. 2009. Promotion analysis in multi-dimensional space. *Proceedings of the VLDB Endowment* 2, 1 (2009), 109–120. https://doi.org/10.14778/1687627.1687641

[70] You Wu, Pankaj K Agarwal, Chengkai Li, Jun Yang, and Cong Yu. 2012. On "one of the few" objects. In *KDD*. 1487–1495. https://doi.org/10.1145/2339530.2339762

[71] Yueji Yang, Yuchen Li, Panagiotis Karras, and Anthony K.H. Tung. 2021. *Context-aware Outstanding Fact Mining from Knowledge Graphs*. Vol. 1. Association for Computing Machinery. 2006–2016 pages. https://doi.org/10.1145/3447548.3467272

[72] Ruslan Yermakov, Bayer Ag, Nicholas Drago, Bayer Ag, Angelo Ziletti, and Bayer Ag. 2021. Biomedical Data-to-Text Generation via Fine-Tuning Transformers. In *INLG*. 364–370.

[73] Gensheng Zhang, Damian Jimenez, and Chengkai Li. 2018. Maverick: discovering exceptional facts from knowledge graphs. In *SIGMOD*. 1317–1332. www.snopes.com/

[74] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2020. Fine-Tuning Language Models from Human Preferences. *https://arxiv.org/abs/1909.08593* (2020). arXiv:1909.08593 http://arxiv.org/abs/1909.08593