Discovering Functional Dependencies: Can We Use ChatGPT to Generate Algorithms?

Loredana Caruccio[1,*,†], Stefano Cirillo[1,2,†], Tullio Pizzuti[1,†] and Giuseppe Polese[1,†]

[1]University of Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano, Salerno, Italy
[2]Hasso Plattner Institute, University of Potsdam, Germany

Abstract

The establishment of Large Language Models allowed people to interact with tools capable of answering in a natural language many kinds of questions on even very large sets of topics. Although the natural language generation processes have to address several issues (e.g., providing focused content w.r.t. queries, composing texts without ambiguities, and so forth), models and tools are becoming more and more capable of providing answers with a syntactically and semantically correct form, independently from both topics and languages. This led to enabling *an algorithm to become capable of writing algorithms* together with their implementation, so tackling an even more complex task since programming languages are more rigid and precise, and the generated code should also embrace the reasoning underlying methodologies used to solve problems at different levels of complexities. At present, the most representative example of such a tool is given by ChatGPT. Based on the GPT-3.5 model and trained over more than 300 Billion tokens, ChatGPT obtained high notoriety and is starting to impact society due to its wide usage in the daily life of people. This paper aims at evaluating to what extent ChatGPT and its underlying model are capable of generating algorithms for the discovery of Functional Dependencies (FDs) from data. The latter represents a very complex problem to which the scientific literature has devoted much effort. The inference of a correct, minimal, and complete set of FDs, holding on a given dataset, defines the main constraints guaranteeing literature solutions to be considered effective, leading to questioning if also solutions generated from ChatGPT can satisfy them. In particular, by following a prompt-based approach, we enabled ChatGPT to provide 7 different solutions to the FD discovery problem and measured their results in comparison with the ones provided by the HyFD discovery algorithm, one of the most efficient solutions provided in the literature.

Keywords

Data Profiling, Functional Dependency, Large Language Model, ChatGPT

1. Introduction

In recent years, the diffusion of large language models (LLMs) has led to a revolution in the area of natural language processing, also affecting the activities of many other research areas. These models have demonstrated a remarkable ability to understand, generate and manipulate natural language mainly due to the large amounts of text they have been trained on. These also entailed the introduction of bots and tools that can be used by users, such as ChatGPT[1], Google

ITADATA2023: The 2nd Italian Conference on Big Data and Data Science, September 11–13, 2023, Naples, Italy
*Corresponding author.
†These authors contributed equally.
lcaruccio@unisa.it (L. Caruccio); scirillo@unisa.it (S. Cirillo); t.pizzuti1@studenti.unisa.it (T. Pizzuti); gpolese@unisa.it (G. Polese)
0000-0002-2418-1606 (L. Caruccio); 0000-0003-0201-2753 (S. Cirillo); 0000-0002-8496-2658 (G. Polese)

[1]www.chat.openai.com

Bard[2], and so forth, for querying LLMs with specific and/or generic questions. In fact, besides their potential in machine translation, text generation, and language processing, LLMs are also finding applications in several new domains, such as security [1], healthcare [2], the Internet of Things (IoT) [3].

The applicability of LLMs, in the area of algorithm and code generation, requires the generated code to incorporate the underlying reasoning and methodologies employed to address problems at various levels of complexity. Thus, to evaluate the effectiveness of LLMs in tackling with complex tasks, in this paper, we focus on possible solutions in the data profiling research area. The latter defines the set of activities and processes to examine, create, and extract useful information from the data. This information permits to identify data quality issues, risks, and overall trends by using profiling metadata, among which the simplest are the average, minimum, maximum, frequency of values, and some more complex ones, such as functional dependencies and inclusion dependencies. Identifying such metadata from data is an extremely complex problem, especially when the data is very large and/or when it evolves over time [4]. In fact, new profiling methodologies continue to be investigated to enable the efficient profiling of ever-increasing amounts of data and the application of metadata in new scenarios.

LLMs have the capabilities to offer new perspectives in the area of data profiling, leading to the definition of advanced data analysis processes, paving the way towards the definition of new data and metadata interpretation methodologies, and offering valuable support for the management and analysis of heterogeneous and complex data. Furthermore, the use of LLMs could lead to the definition of new metadata discovery algorithms based on new methodologies generated by their knowledge. To this end, in this paper, we propose a comparative evaluation of FD discovery algorithms generated by one of the most well-known LLMs, i.e., ChatGPT. To perform this evaluation, we asked ChatGPT to generate the executable source code of several algorithms by means of a new prompt engineering strategy. The results of the algorithms generated by ChatGPT have been evaluated on real-world datasets and their results have been compared with those achieved by HyFD [5], one of the best-performing FD discovery algorithms in the literature.

The rest of the article is organized as follows: Section 2 surveys relevant related works in the literature; Section 3 provides preliminary notions about the definition of FDs and the discovery problem; Section 4 shows the new Prompt Template Engineering approach for generating FD discovery algorithms together with algorithms generated by ChatGPT; Section 5 shows the experimental evaluation and provides a discussion of results; Conclusions and future directions are discussed in Section 6.

2. Related Work

The Data Profiling area aims to process and analyze data to evaluate their quality, structure, and characteristics, through the extraction and analysis of different metadata from the data [6]. This metadata enables to identify patterns, anomalies, and inconsistencies with the aim to define advanced techniques for supporting different operations, such as Data Integration [7], Query Processing [8], and Data Cleaning [9]. In fact, when data is created or shared, it

[2]www.bard.google.com

often has missing values, inaccuracies, and/or errors of various kinds. Often the datasets used in the analysis processes have different formats or inconsistencies that can negatively affect the performance of the AI models [10]. In this scenario, the Data Preparation tasks permit to collect, structure, and organize data to improve the quality and make them suitable for analytical and predictive models [10]. Therefore, these profiling metadata can allow the extraction of meaningful information and enhance preparation and analytical processes.

Among the different types of profiling metadata, Functional Dependencies (FDs) are widely adopted for different data preparation tasks [11], so requiring the design of efficient discovery algorithms capable of identifying all FDs holding on a given dataset. In the literature, three different types of algorithms have been proposed, i.e., column-based, row-based, and hybrid, respectively. Column-based algorithms rely on a lattice structure to efficiently explore the search space and to generate candidate FDs according to the Apriori search strategy [6], which enables pruning the search space and reducing the number of FDs to be validated [12, 13]. Instead, the row-based algorithms generate candidate FDs from two attribute subsets, namely *agree-set* and *difference-set*, which are built by comparing the values of attributes between all possible combinations of tuples pairs [14, 15]. Recently, new hybrid algorithms have been proposed, which exploit the advantages of both row-based and column-based methodologies, to improve the FD discovery process [5].

The approaches mentioned above represent pillars for the very complex FD discovery task. They preserve the requirements of correctness, completeness, and minimality of results. Nevertheless, data scientists could draw inspiration from LLM-generated codes to consider alternative FDs discovery approaches. In fact, the spread of LLMs has led to taking advantage of the ability of these LLMs to easily generate parts of code or entire algorithms to speed up their work. For instance, a recent study has proposed an extended version of GPT, trained for the automatic generation of source codes written in Python [16]. The results were evaluated through a new HumanEval dataset is composed of 164 programming problems with associated unit tests [16]. Starting from this dataset, authors in [17] have proposed a new framework to generate different inputs for unit tests in order to increase the tests to be performed on each generated algorithm. Recently, a new LLM has been proposed, namely PolyCoder [18], which has been trained on a large set of open-source codes from GitHub with the aim to automatically generate source codes in 12 different programming languages. The performance of PolyCoder was compared with some of the major LLMs trained for code generation [16]. However, it is necessary to evaluate the correctness of these source codes and algorithms before adopting them in real scenarios.

Through this study, we evaluated if the knowledge underlying an LLM permits the generation of algorithms to solve a specific task, such as the FD discovery one, which would represent, to the best of our knowledge, the first comparative study on FDs discovery algorithms generated by LLM models.

3. Preliminaries

One of the main data profiling tasks is the FD discovery one. Formally, given a relation schema R, and an instance r of it, an FD holding on r is a statement $X \rightarrow Y$ (X implies Y), with X and Y attribute sets of R, such that for every pair of tuples (t_1, t_2) in r, whenever $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$. X and Y are also named Left-Hand-Side (LHS) and Right-Hand-Side (RHS),

respectively, of the FD. An FD is said to be *non-trivial* if and only if $X \cap Y = \emptyset$. Moreover, an FD is said to be *minimal* if and only if there is no attribute $B \in X$ such that $X \backslash B \rightarrow Y$ holds on r.

Discovering FDs from data is an extremely complex problem, due to the exponential number of column combinations to be analyzed [19]. Formally, given a relation instance r with n tuples and M attributes, and considering w.l.o.g. candidate FDs with a single attribute on the RHS, it is necessary to validate each possible candidate FD. Thus, the FD discovery problem has to consider 2^M possible attribute combinations, each employing n^2 iterations in a brute-force approach for validating them, since all pairs of tuples must be compared to verify the property. This leads to a complexity's upper bound that is $O(n^2(\frac{M}{2})^2 2^M)$, with $\frac{M}{2}$ attribute combinations representing the average number of attributes involved in an FD [20].

Data profiling algorithms deterministically perform such an analysis where the properties of correctness, completeness, and minimality are guaranteed on the discovery results. However, they typically exploit FDs already validated and the FD inference rules to prune the search space in order to limit the number of candidate FDs to be validated. Nevertheless, in the analyzed scenario, it is necessary to verify if the previously mentioned properties are guaranteed by FD discovery algorithms generated through ChatGPT. In particular, although failures in the satisfiability of completeness and minimality can be tolerated, the correctness should be always preserved to avoid the consideration of not holding FDs in the result set. To this end, it is necessary to introduce the concept of *specialization* and *generalization* of a given FD $\varphi : X \rightarrow Y$.

More formally, given φ, an FD φ': $X \backslash B \rightarrow Y$ represents a *generalization* of φ for any $B \in X$. Instead, an FD φ'': $X \cup B \rightarrow Y$ represents a *specialization* of φ for any $B \notin X \cup Y$. Consequently, to compare results between different algorithms we also consider these concepts to measure the quality of discovery results w.r.t. a correct, complete, and minimal set of FDs holding on a given relation r.

4. Generation of FD Discovery Algorithms

In this section, we first introduce a new prompt engineering approach that we have defined for generating FD discovery algorithms, and then we provide an overview of the algorithms provided by ChatGPT for the purpose of this study.

4.1. Prompt engineering approach

We used ChatGPT, which is based on one of the best-known large language models, to generate seven FD discovery algorithms that have been successively evaluated on real-world datasets. In particular, the interaction with ChatGPT required the definition of multiple prompts to collect discovery algorithms of different natures. To this end, we have designed an approach to generate prompts for ChatGPT based on the automatic Prompt Template Engineering strategies [21]. As shown in Figure 1, the approach starts by asking ChatGPT the following prompt: $p_1 =$ "*What type of functional dependency discovery algorithms do you know?*". After the response of the ChatGPT, the approach asks ChatGPT which of the algorithms of the previous answer can provide a source code, exploiting the prompt $p_2 =$ "*Which of these Algorithms can you provide me with a source code?*". This strategy allowed us to focus our study only on the algorithms of which ChatGPT has been able to provide us with at least an executable code. Starting from the list of algorithms provided by p_2, we have defined an ad-hoc prompting functions $x' = f_{\text{algorithm-filling}}(x)$, where x represents the input of the prompt composed by the type of

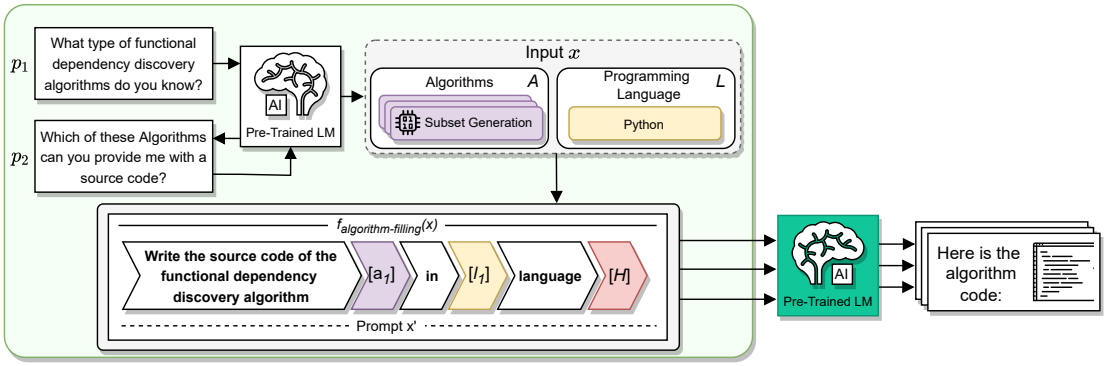

Figure 1: Overview of the prompt engineering methodology.

algorithm to be generated and its programming language, and x' is the prompt sentence to be submitted to the LLM for generating the FD discovery algorithm. The template for this prompt has been defined as follows

"**Write the source code of the functional dependency discovery algorithm** [A] **in** [L] **language** [H]"

where [A] is the slot containing the type of algorithm, [L] is the slot containing the programming language of the algorithm, and [H] is the answer slot that will be filled after the generation of the source code issued by the LLM. More formally, the slot [A] is the set $A = \{a_1, a_2, \ldots, a_n\}$ of the types of algorithms of which the LLM can generate the source code, and [L] is the set $L = \{l_1, l_2, \ldots, l_m\}$ of the programming language. As an example, the first prompt submitted to ChatGPT was: "*Write the source code of the functional dependency discovery algorithm Apriori in Python language*".

In our study we only consider a single source code at a time for each algorithm (a_1), i.e., the first one provided by ChatGPT, and a single programming language (l_1). Notice that, LLMs have the potential to generate several source codes for the same algorithm. Thus, our approach has been designed to be easily generalizable for the request of multiple algorithms at once and written in different programming languages.

In what follows, we discuss each algorithm generated by ChatGPT, by also providing an overview of their methodologies.

4.2. Discovery Algorithms

We have generated seven different algorithms through ChatGPT, and each of them consisted of a single procedure, without any module for reading the datasets subject of the discovery and/or for standardizing the results. In what follows, we provide an overview of the discovery strategies underlying the generated algorithms.

Apriori: This algorithm relies on the Apriori search strategy, (e.g., a traditional approach in column-based discovery algorithms), in order to identify the set of FDs valid a dataset. The algorithm uses the concept of frequent itemsets to identify attributes that appear with significant frequency, as generally used for the association rule mining task.

The algorithm starts by generating the frequent itemsets of size 1, i.e., those containing a single attribute. Then, it extends the frequent itemsets computing those with sizes from $k-1$ to k. Starting from frequent itemsets, the algorithm validates the candidate FDs using

the confidence measure. Given a FD $\varphi : X \rightarrow Y$, the confidence establishes the conditional probability of having a certain value combination on attributes Y given a set of determining attributes X. Moreover, the validation function reads as an input a minimum threshold that allows to constrain the validation of a candidate FD, i.e., the latter is valid iff its confidence exceeds the input threshold. For example, let us consider an FD $A, B \rightarrow C$. The Apriori algorithm computes the frequency of occurrence of the itemset (A, B, C) w.r.t. the itemset (A, B). The confidence of the FD is then calculated as: $confidence = \frac{frequency\_of(A,B,C)}{frequency\_of(A,B)}$. If this value exceeds the predefined threshold then the candidate FD is valid.

Subsetgen: This algorithm relies on the subset generation approach to extract the FDs holding on a given dataset. The algorithm generates all possible subsets of attributes and computes the closure of each subset. The closure represents the full set of attributes determined by the subset of attributes considered. For example, let us consider a dataset of 4 attributes (A, B, C, D), the subsets generated are: $(A), (B), \ldots, (A, B), \ldots, (A, B, D), \ldots, (A, B, C, D)$. If we consider the subset (A, B), its closure is the set of all attributes determined by (A, B). After calculating the closure for each subset, for each FD $\varphi : X \rightarrow Y$, the algorithm computes the confidence of the set of attributes on the LHS and RHS, and if the confidence is equal to 1, then the FD is valid.

Anova: This algorithm relies on the analysis of variance (Anova) to identify FDs valid in a dataset. The algorithm evaluates the variation of an attribute Y based on the other attributes in X, which are considered determinants. In particular, the algorithm computes the one-way ANOVA for determining the statistical differences between the means of multiple data, computing their p-value. The latter is a parameter used to discriminate a hypothesis test, which is used by the algorithm to check whether all means of the projections of the attributes of the FD on the dataset are equal. A p-value less than 0.05, i.e., the standard p-value, indicates that the FD analyzed is valid on the dataset.

Linear Regression: This algorithm relies on the linear regression algorithm for discovering FDs holding in a dataset. The algorithm starts by defining the set of candidate FDs to validate considering a single attribute on the RHS and all the other attributes of the dataset on the LHS. In this way, the algorithm evaluates only the candidate FDs with the highest number of attributes on the LHS. Starting from this, for each attribute on the RHS, a linear regression model is built that tries to evaluate the variation of the attribute based on the values of the attributes on the LHS. In particular, the values of the attributes on the LHS of each candidate FD are used as the training data of the linear regression model, while the values of the attribute on the RHS as their target values. The algorithm trains a linear regression model for each FD to be validated and extracts the regression coefficient representing the relative importance of the determining attribute. If the coefficient exceeds the value 0.95 defined by default then the candidate FD is considered valid.

Correlation & Regression: This algorithm relies on the Pearson correlation coefficient to validate FDs on a given dataset. The first step of the algorithm requires replacing all values that are not numbers with a numerical representation, since it works only with numerical values. Then, in its main step, it trains a linear regression model on every possible pair of attributes, i.e., allowing for testing the validation on only the most general FDs. Among the resulting outputs provided by the linear regression, the algorithm considers the Pearson correlation coefficient to

validate the FDs. Thus, an FD is valid iff there is a high degree of positive correlation between the attribute pairs.

Pair-wise Algorithm: This algorithm relies on a pair-wise comparison of the attributes in the dataset to validate candidate FDs. The algorithm has been designed and developed to consider only FDs with a single attribute on the LHS and one on the RHS. For each dependency $X \rightarrow Y$, check if every subset of tuples having equal value on attribute X always has the same value for attribute Y.

TANE-based Algorithm: The latest algorithm is based on TANE, which is one of the pioneering algorithms in the literature[13]. TANE relies on Stripped Partitions, which are partitions of value combinations built based on the equality constraint and having more than one element within the structure. This structure makes it easy to check if a dependency holds by the *partition refinement* property [13]. TANE explores the search space of all possible candidate FDs and thanks to pruning methods it is able to reduce the number of FDs to validate. Specifically, the source code provided by ChatGPT lacks some of the core parts of TANE, such as pruning strategies, the lattice structure, and the validation based on the refinement property. In fact, the algorithm erroneously reverses the RHS and LHS of an FD, issuing errors when generating the possible combinations of attributes. Nevertheless, each FD is validated by means of a property that checks if the values of the tuples for all the attributes involved in a candidate FD form a primary key.

5. Experimental Evaluation

The experimental evaluation has been conducted by comparing the results achieved by the seven algorithms generated by ChatGPT[3] with those achieved by HyFD [5], one of the best-performing algorithms for FD discovery. Specifically, this comparative evaluation allowed us to measure the suitability of GPT-generated code with the complex problem of discovering FDs form data.

5.1. Experimental Settings

The experiments were performed on different real-world datasets previously used for evaluating FD discovery algorithms. Table 1 shows the characteristics of the datasets involved in the evaluation. All the experiments have been executed on a computer with an Intel Xeon W at 2.3 GHz, 18-core, and 128GB of memory, running macOS Ventura 13.0.1 and Python 3.10 as the execution environment.

The discovery algorithms have been generated in Python language and we kept their structures in order to preserve strategies behind them and evaluate the capability of ChatGPT of generating FD discovery algorithms. However, we have only introduced the procedures for reading the datasets and standardizing the results. To evaluate the results achieved by the generated algorithm, we compare them with those achieved by HyFD with the aim to evaluate their performances in terms of Precision, Recall, and F1-Score. In particular, we considered two distinct cases: i) the number of minimal FDs correctly discovered by the generated algorithms w.r.t. those discovered by HyFD (namely, *Common Case*), and ii) the number of both minimal

[3]The source code of the FD discovery algorithms generated is available on the repository https://github.com/DastLab/FDDiscoveryChatGPT

ID	Dataset	Columns	Rows	#FDs
D1	*Chess*	7	28,056	1
D2	*Abalone*	9	4,177	137
D3	*Nursery*	9	12,960	1
D4	*Electricity Normalizer*	9	45,312	61
D5	*Customer Shopping Data*	10	99,457	21
D6	*Fraudfull*	11	98,439	39

ID	Dataset	Columns	Rows	#FDs
D7	*Poker Hand*	11	1,025,010	1
D8	*Firewall*	12	65,532	88
D9	*Adult*	15	32,562	78
D10	*Letter*	17	20,000	61
D11	*Sgemm Product Rounded*	18	241,600	4

Table 1
Characteristics of the real-world datasets used for the experimental evaluation.

and valid FDs also considering the case in which the generated algorithms validate specialized FDs w.r.t. those discovered by HyFD (namely, *Specialization Case*).

In the *Common Case*, we considered the FDs identified by ChatGPT algorithms that also occur in HyFD results as True Positives (i.e., TP_{com}); the FDs validated by ChatGPT algorithms that do not occur in HyFD results as False Positives (i.e., FP_{com}), and the FDs validated by HyFD that have not been discovered by ChatGPT algorithms as False Negatives (i.e., FN_{com}).

In the *Specializations Case*, we considered both the common and specialized FDs, extracted by generated algorithms w.r.t. those occurring in HyFD results, as True Positive (i.e., TP_{spec}); the FDs discovered by HyFD for which there are no equal or specialized FDs within the results of the generated algorithms as False Negatives (i.e., FN_{spec}), and the FDs discovered by the generated algorithms for which neither equal nor generalized FDs within the results of HyFD can be found as False Positives (i.e., FP_{spec}). It is important to notice that in both cases we do not consider True Negatives (i.e., TN), since they are the FDs that are invalid and whose details are not provided by HyFD.

5.2. Experimental Results

Table 2 shows the results achieved by all the generated ChatGPT algorithms, respectively. In particular, such tables split the results according to the two above specified cases, i.e., the *Common Case*, shortened with *com*, and the *Specialization Case*, shortened with *spec*.

In addition, for each algorithm, two types of charts were constructed (see Figure 2). Given an algorithm, the chart on the left shows the resulting minimal and valid FDs compared to the solutions identified by HyFD. On the other hand, the chart on the right side shows the number of incorrect FDs and generalizations identified by the generated algorithms with respect to the number of discovered FDs.

Results show that the algorithms, on most datasets fail in identifying minimal or valid dependencies. For example, the *Apriori* and *Subset Generation* algorithms identified only a subset of minimal or valid dependencies. However, many incorrect dependencies were also generated, as can be seen in Figure 2. In fact, also Tables 2 report very low values for all metrics.

Specifically, the *Correlation & Regression* algorithm failed to identify any minimal or valid dependencies. On 4 out of 11 datasets, it produced no result, while on the remaining ones it identified only incorrect dependencies or generalizations (see Figure 2). According to these results, its metrics are all equal to 0. This behaviour can be due to the fact that Pearson correlation coefficient only takes into account the correlation degree, e.g., positive or negative. However, a strong positive correlation between two attributes does not imply the existence of a FD. Similarly, the *Linear Regression* algorithm does not detect any FD that is also minimal. However, it achieved some good results on 5 datasets considering valid dependencies (i.e., also in specialized form). It also generated only a few not holding FDs and no generalized ones.

		Anova		Apriori		Correlation		Linear Reg.		Subset Gen.		Pair-wise		Tane	
		com	*spec*	*com*	*spec*	*com*	*spec*	*com*	*spec*	*com*	*spec*	*com*	*spec*	*com*	*spec*
Precision	D1	0	0	0	0	0	0	0	0	0	0	0	0	0.14	0.14
	D2	0	0	0.06	0.06	0	0	0	1.00	0	0	0	0	0	1.00
	D3	0	0	0	0	0	0	0	0	0	0	0	0	0.11	0.11
	D4	0	0	0.04	0.04	0	0	0	1.00	0	0	0	0	0	0.97
	D5	0.33	0.33	0	0	0	0	0	0	0	0	1.00	1.00	0	1.00
	D6	0	0	0	0	0	0	0	0.96	0	0	0.47	0.47	0	0.97
	D7	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0.09
	D8	0	0	0	0	0	0	0	0.98	0.01	0.01	0	0	0	0.95
	D9	0.01	0.01	0	0	0	0	0	0	0	0	1.00	1.00	0	0.90
	D10	0	0	0	0	0	0	0	0	0	0	0	0	0	0.79
	D11	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0.22
Recall	D1	0	0	0	0	0	0	0	0	0	0	0	0	1.00	1.00
	D2	0	0	0.68	0.68	0	0	0	0.82	0	0	0	0	0	1.00
	D3	0	0	0	0	0	0	0	0	0	0	0	0	1.00	1.00
	D4	0	0	0.74	0.74	0	0	0	0.44	0	0	0	0	0	1.00
	D5	0.09	0.09	0	0	0	0	0	0	0	0	0.95	0.95	0	1.00
	D6	0	0	0	0	0	0	0	0.63	0	0	0.24	0.24	0	1.00
	D7	0	0	0	0	0	0	0	0	0	0	0	0	1.00	1.00
	D8	0	0	0	0	0	0	0	0.96	0.09	0.09	0	0	0	1.00
	D9	0.02	0.02	0	0	0	0	0	0	0	0	0.02	0.02	0	1.00
	D10	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00
F1-Score	D1	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0.25
	D2	0	0	0.10	0.10	0	0	0	0.90	0	0	0	0	0	1.00
	D3	0	0	0	0	0	0	0	0	0	0	0	0	0.20	0.20
	D4	0	0	0.07	0.07	0	0	0	0.61	0	0	0	0	0	0.98
	D5	0.15	0.15	0	0	0	0	0	0	0	0	0.97	0.97	0	1.00
	D6	0	0	0	0	0	0	0	0.76	0	0	0.31	0.31	0	0.99
	D7	0	0	0	0	0	0	0	0	0	0	0	0	0.17	0.17
	D8	0	0	0	0	0	0	0	0.97	0.01	0.01	0	0	0	0.97
	D9	0.01	0.01	0	0	0	0	0	0	0	0	0.05	0.05	0	0.94
	D10	0	0	0	0	0	0	0	0	0	0	0	0	0	0.88
	D11	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0.36

Table 2
Results metrics related to *common* and *specialization* cases of the ChatGPT algorithms.

As for the *Anova* algorithm, it generated many not holding dependencies (also in generalized form) on all considered datasets. This leads to very low performances in all considered metrics, also due to the fact that the number of correct FDs is very low. Like other generated algorithms, even the *Pairwise* algorithm discovered FDs that contain a single attribute on the left-hand side, but in this case, its validation strategy turns out to be better than other algorithms, among those working with the most generalized candidate FDs as possible, and only on one of the datasets, it discovers incorrect FDs. In addition, by looking at Table 2, for datasets on which the algorithm is able to find holding FDs, the metrics results are satisfactory.

Finally, the Tane algorithm generated many specializations but at the same time many incorrect dependencies. Although its results on minimal dependencies are low, if we go to consider valid dependencies we have a significant increase even compared to all other algorithms. This implementation always generated specialized dependencies of the maximum size, i.e., you have $n - 1$ attributes on the left side and only one attribute on the right side, yielding the most specialized candidate FDs as possible. In fact, from Figure 2 we see that it has many

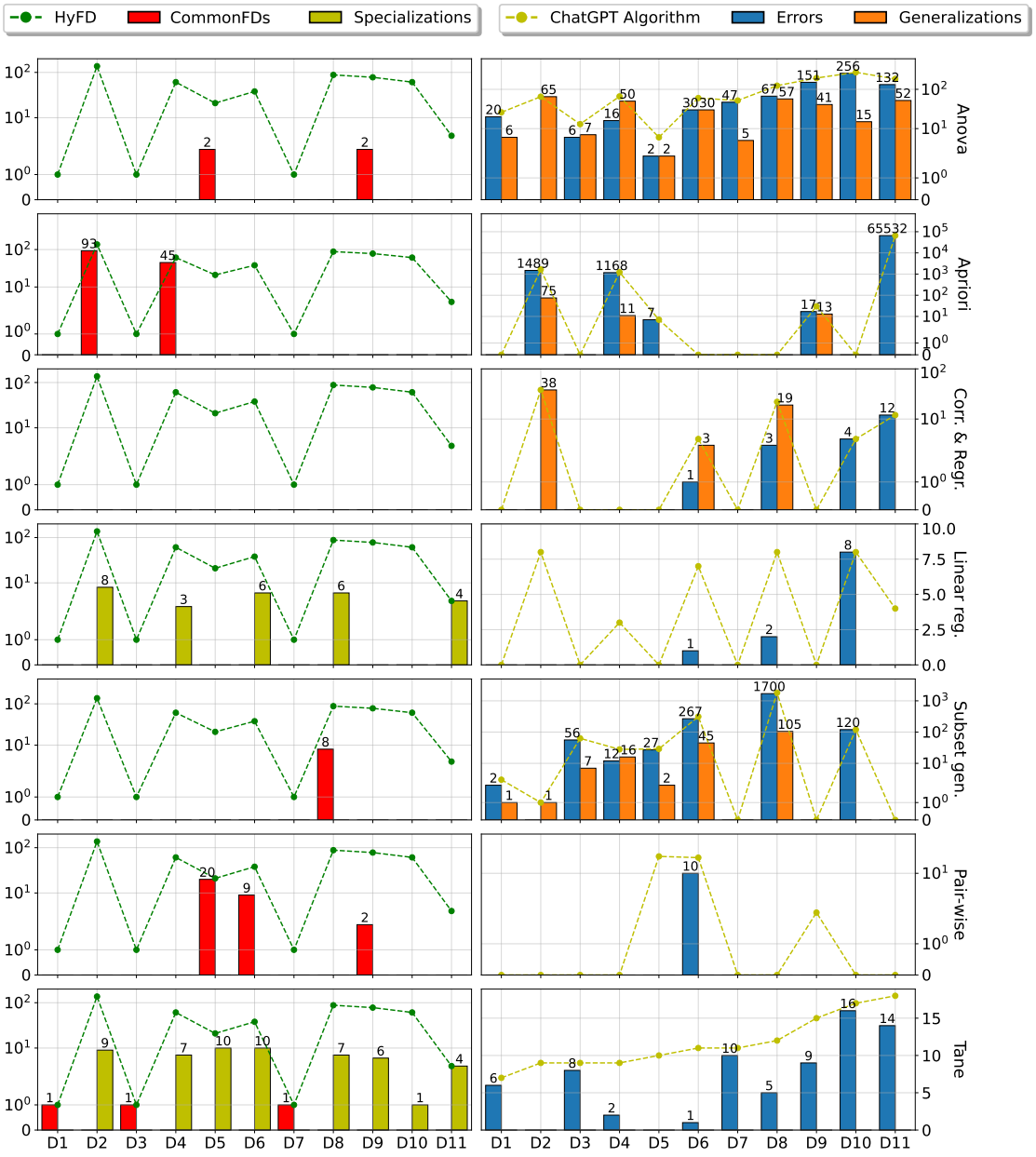

Figure 2: Characterization of discovered FDs by generated algorithms w.r.t. oracle results.

specializations and no generalizations.

6. Conclusion and Future Directions

In this paper, we considered the possibility to entrust LLMs for the automatic generation of algorithms in the area of Data Profiling. To the best of our knowledge, it represents the first proposal trying to exploit the potential of LLMs for the definition of possible discovery algorithms in the data profiling area. Specifically, we obtained seven different FD discovery

algorithms by means of a prompt learning strategy with ChatGPT. Then, we evaluated the effectiveness of such algorithms by comparing their discovery results with respect to the ones of the HyFD algorithm. Evaluation results demonstrate that most of the statistical or ML-based ChatGPT-generated algorithms, i.e., *Correlation & Regression* and *Linear Regression*, limit the exploration of the search space to the most generalized candidate FDs as possible, leading to having very few valid FDs discovered. Instead, the other algorithm in this category, i.e., *Anova* can consider more, randomly generated, candidate FDs, but it is not able to improve performances due to the high number of generalizations and/or errors in results. On the contrary, the other two generated algorithms, i.e., *Apriori* and *Subset Generation*, that perform an exhaustive exploration of the search space, in general, achieve a number of FDs much higher. However, the validation method of both turned out to be fallacious, leading to obtaining many errors in the discovery results. The generated *Pairwise* algorithm includes a good validation method, but also, in this case, the evaluation of the only most generalized candidate FDs, results in obtaining few discovered FDs. Finally, the *Tane* algorithm represents the most complete generated discovery algorithms, since it is related to a literature approach. Nevertheless, the generated code presents many errors in the algorithm logic, yielding the algorithm considering only the most generalized FDs as possible. In fact, results appear good, but also in this case the correctness of discovered FDs is not preserved. The latter consideration applies to all generated algorithms, allowing us to state the used general-purpose LLM has not been able to generate solutions guaranteeing correctness in discovery results, a fundamental property to preserve to make discovery results useful in application scenarios.

In the future, it would be useful to investigate new prompting approaches for enabling LLMs to refine the generated algorithms, involving domain experts in revising the methodologies of the generated algorithms or using existing discovery algorithms for guiding LLMs in the direct validation of FDs. The latter could involve the usage of multimodal prompts to process entire datasets. In fact, current open LLMs are limited to a maximum number of tokens that can be processed and only allow us to handle datasets through batching strategies. Finally, we would like to extend our evaluation by also considering specialized LLMs for algorithm generation.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

References

[1] G. Sandoval, H. Pearce, T. Nys, R. Karri, S. Garg, B. Dolan-Gavitt, Lost at c: A user study on the security implications of large language model code assistants, arXiv preprint (2023).

[2] A. Arora, A. Arora, The promise of large language models in health care, The Lancet 401 (2023) 641.

[3] O. Zheng, M. Abdel-Aty, D. Wang, Z. Wang, S. Ding, ChatGPT is on the horizon: Could a large language model be all we need for intelligent transportation?, arXiv preprint (2023).

[4] B. Breve, L. Caruccio, S. Cirillo, V. Deufemia, G. Polese, IndiBits: Incremental discovery of relaxed functional dependencies using bitwise similarity, in: Proceedings of the 39th IEEE International Conference on Data Engineering, ICDE, 2023.

[5] T. Papenbrock, F. Naumann, A hybrid approach to functional dependency discovery, in:

Proceedings of the 2016 International Conference on Management of Data, SIGMOD, 2016, pp. 821–833.
[6] Z. Abedjan, L. Golab, F. Naumann, Profiling relational data: a survey, VLDB J. 24 (2015) 557–581.
[7] M. Lenzerini, Data integration: A theoretical perspective, in: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS, 2002, p. 233–246.
[8] L. Caruccio, S. Cirillo, V. Deufemia, G. Polese, R. Stanzione, REQUIRED: A tool to relax queries through relaxed functional dependencies, in: Proceedings of the 26th International Conference on Extending Database Technology, EDBT, 2023, pp. 823–826.
[9] B. Breve, L. Caruccio, V. Deufemia, G. Polese, RENUVER: A missing value imputation algorithm based on relaxed functional dependencies, in: Proceedings of the 25th International Conference on Extending Database Technology, EDBT, 2022, pp. 52–64.
[10] A. Jain, H. Patel, L. Nagalapatti, N. Gupta, S. Mehta, S. Guttula, S. Mujumdar, S. Afzal, R. Sharma Mittal, V. Munigala, Overview and importance of data quality for machine learning tasks, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD, 2020, pp. 3561–3562.
[11] A. A. A. Fernandes, M. Koehler, N. Konstantinou, P. Pankin, N. W. Paton, R. Sakellariou, Data preparation: A technological perspective and review, SN Comput. Sci. 4 (2023) 425.
[12] H. Yao, H. J. Hamilton, C. J. Butz, FD_Mine: Discovering functional dependencies in a database using equivalences, in: Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM, 2002, pp. 729–732.
[13] Y. Huhtala, J. Kärkkäinen, P. Porkka, H. Toivonen, TANE: An efficient algorithm for discovering functional and approximate dependencies, Comput J. 42 (1999) 100–111.
[14] S. Lopes, J. Petit, L. Lakhal, Efficient discovery of functional dependencies and armstrong relations, in: Proceedings of the 7th International Conference on Extending Database Technology, EDBT, volume 1777, 2000, pp. 350–364.
[15] P. A. Flach, I. Savnik, Database dependency discovery: A machine learning approach, AI Commun. 12 (1999) 139–160.
[16] M. Chen, J. Tworek, H. Jun, Q. Yuan, et al., Evaluating large language models trained on code, arXiv preprint (2021).
[17] J. Liu, C. S. Xia, Y. Wang, L. Zhang, Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation, arXiv preprint (2023).
[18] F. F. Xu, U. Alon, G. Neubig, V. J. Hellendoorn, A systematic evaluation of large language models of code, in: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, MAPS, 2022, p. 1–10.
[19] T. Bläsius, T. Friedrich, M. Schirneck, The complexity of dependency detection and discovery in relational databases, Theor. Comput. Sci. 900 (2022) 79–96.
[20] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, F. Naumann, Functional dependency discovery: An experimental evaluation of seven algorithms, Proc. VLDB Endow. 8 (2015) 1082–1093.
[21] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, G. Neubig, Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing, ACM Comput. Surv. 55 (2023) 1–35.