LINX: A Language Driven Generative System for Goal-Oriented Automated Data Exploration

Tavor Lipman
Tel Aviv University
tavorlipman@mail.tau.ac.il

Tova Milo
Tel Aviv University
milo@cs.tau.ac.il

Amit Somech
Bar-Ilan University
somecha@cs.biu.ac.il

Tomer Wolfson
Tel Aviv University
tomerwol@mail.tau.ac.il

Oz Zafar
Tel Aviv University
ozzafar@mail.tau.ac.il

ABSTRACT

Data exploration is a challenging and time-consuming process in which users examine a dataset by iteratively employing a series of queries. While in some cases the user explores a new dataset to become familiar with it, more often, the exploration process is conducted with a specific analysis goal or question in mind. To assist users in exploring a new dataset, Automated Data Exploration (ADE) systems have been devised in previous work. These systems aim to auto-generate a full exploration session, containing a sequence of queries that showcase interesting elements of the data. However, existing ADE systems are often constrained by a predefined objective function, thus always generating the same session for a given dataset. Therefore, their effectiveness in goal-oriented exploration, in which users need to answer specific questions about the data, are extremely limited.

To this end, this paper presents LINX, a generative system augmented with a natural language interface for goal-oriented ADE. Given an input dataset and an analytical goal described in natural language, LINX generates a personalized exploratory session that is relevant to the user's goal. LINX utilizes a Large Language Model (LLM) to interpret the input analysis goal, and then derive a set of specifications for the desired output exploration session. These specifications are then transferred to a novel, modular ADE engine based on Constrained Deep Reinforcement Learning (CDRL), which can adapt its output according to the specified instructions.

To validate LINX's effectiveness, we introduce a new benchmark dataset for goal-oriented exploration and conduct an extensive user study. Our analysis underscores LINX's superior capability in producing exploratory notebooks that are significantly more relevant and beneficial than those generated by existing solutions, including ChatGPT, goal-agnostic ADE, and commercial systems.

1 INTRODUCTION

Data exploration is the process of examining a dataset by applying a sequence of queries, allowing the user to inspect different aspects of the data. Data exploration can be performed in one of two scenarios. The first, examining an unfamiliar dataset in order to understand its main characteristics. The second, which we refer to as *Goal-oriented Data Exploration* (GDE), is the process of exploring an already familiar dataset in light of a specific analytical goal or question, in order to derive specific, relevant insights.

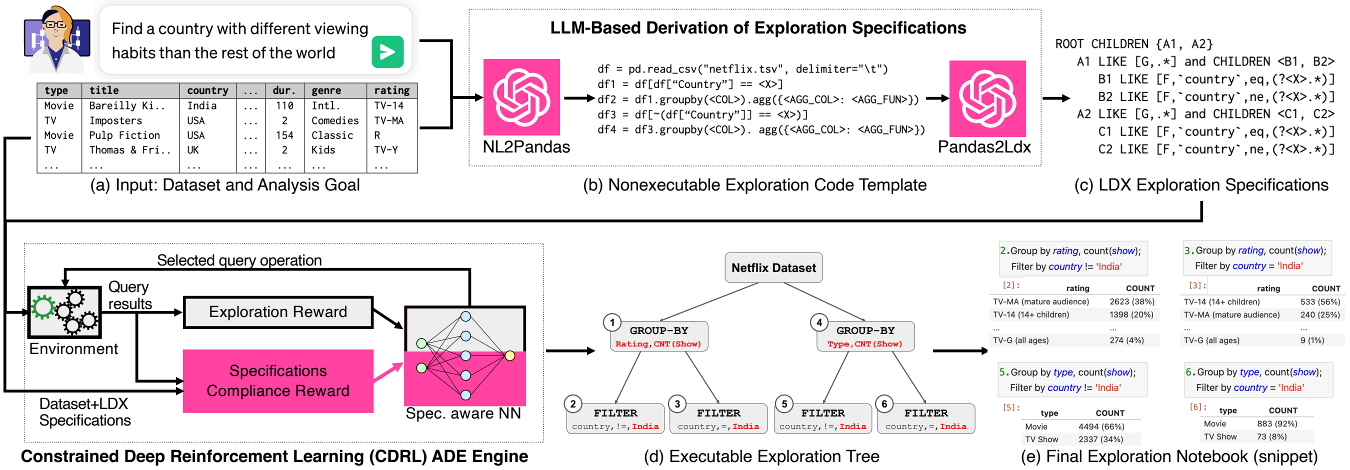
Numerous tools have been devised over the last decade for the purpose of assisting users in the manual, interactive process of data exploration [15, 21, 23, 38, 41, 71]. Most prominently, query recommendation systems and simplified analysis interfaces like Tableau [68] and Power BI [7]. Recently, a new line of work called Automated Data Exploration (ADE), considers data exploration as a multi-step, AI *control problem* [5, 6, 10, 11, 53, 54]. Unlike interactive tools that assist users step-by-step, Automated Data Exploration (ADE) systems take an input dataset and automatically generate a complete session of multiple, interconnected queries. Each query in the session builds on the results of one of the previous queries. The final output session is often displayed in a scientific notebook interface [52], allowing users to quickly gain substantial knowledge on the data before investigating it further.

Importantly, while existing ADE systems have been proven useful in assisting users in examining and familiarizing themselves with a new dataset, they are ineffective for the process of GDE. This is because existing ADE systems solve a predefined optimization problem, with a fixed objective function, thus always generating the same, or similar session for a given dataset. In the case of GDE, users need to answer a specific question, and seek insights that are relevant to their analytical goal. For illustration, consider the following example:

Example 1.1. Data Scientist Clarice, working at a media company, is assigned to analyze the Netflix Movies and TV Shows dataset [34], which contains information about more than 8.8K different titles. Her current assignment is *finding a country with atypical viewing habits, compared to the rest of the world* (to discover new insights that can be utilized to broaden the company's viewership audience). While Clarice is familiar with this dataset, she is tasked with a challenging analytical goal that cannot be answered via a single query. To meet the goal, Clarice needs to examine countries in a trial-and-error manner, comparing them to the rest of the world with different attributes and aggregation functions.

Using the existing ADE system [6], Clarice receives an output exploration notebook, containing query results that imply generic insights such as "*Most Netflix titles originated in the US*". However, these offer no help in respect to Clarice's analytical goal - a specific question about countries with atypical viewing habits.

To this end, we introduce LINX, a *Language-driven generative system for goal-oriented exploration.* LINX is a novel ADE system that receives as input not only the user's dataset, but additionally, a description of the user's *analytical goal* in natural language. LINX then generates a *personalized*, exploratory session, containing queries that are tailored specifically to the dataset and the given goal at hand. LINX follows two steps: First, it uses an LLM-based solution to interpret the input analysis goal and derives from it a set



Find a country with different viewing habits than the rest of the world

type	title	country	...	dur.	genre	rating
Movie	Bareilly Ki..	India	...	110	Intl.	TV-14
TV	Imposters	USA	...	2	Comedies	TV-MA
Movie	Pulp Fiction	USA	...	154	Classic	R
TV	Thomas & Fri..	UK	...	2	Kids	TV-Y
...	...	...	...	...	...	...

(a) Input: Dataset and Analysis Goal

```
df = pd.read_csv("netflix.tsv", delimiter="\t")
df1 = df[df["Country"] == <X>]
df2 = df1.groupby(<COL>).agg({<AGG_COL>: <AGG_FUN>})
df3 = df[~(df["Country"]] == <X>)]
df4 = df3.groupby(<COL>). agg({<AGG_COL>: <AGG_FUN>})
```

(b) Nonexecutable Exploration Code Template

```
ROOT CHILDREN {A1, A2}
  A1 LIKE [G,.*] and CHILDREN <B1, B2>
    B1 LIKE [F,`country`,eq,(?<X>.*)]
    B2 LIKE [F,`country`,ne,(?<X>.*)]
  A2 LIKE [G,.*] and CHILDREN <C1, C2>
    C1 LIKE [F,`country`,eq,(?<X>.*)]
    C2 LIKE [F,`country`,ne,(?<X>.*)]
```

(c) LDX Exploration Specifications

Constrained Deep Reinforcement Learning (CDRL) ADE Engine

Netflix Dataset
1 GROUP-BY Rating,CNT(Show)
4 GROUP-BY Type,CNT(Show)
2 FILTER country,!=,India
3 FILTER country,=,India
5 FILTER country,!=,India
6 FILTER country,=,India

(d) Executable Exploration Tree

2.Group by *rating*, count(*show*); Filter by *country* != 'India'

[2]:

rating	COUNT
TV-MA (mature audience)	2623 (38%)
TV-14 (14+ children)	1398 (20%)
...	...
TV-G (all ages)	274 (4%)

3.Group by *rating*, count(*show*); Filter by *country* = 'India'

[3]:

rating	COUNT
TV-14 (14+ children)	533 (56%)
TV-MA (mature audience)	240 (25%)
...	...
TV-G (all ages)	9 (1%)

5.Group by *type*, count(*show*); Filter by *country* != 'India'

[5]:

type	COUNT
Movie	4494 (66%)
TV Show	2337 (34%)

6.Group by *type*, count(*show*); Filter by *country* = 'India'

[6]:

type	COUNT
Movie	883 (92%)
TV Show	73 (8%)

(e) Final Exploration Notebook (snippet)

Figure 1: An Example LINX Workflow for Auto-Generating Goal-Oriented Exploration Sessions

of specifications for the desired output exploration session. Second, the dataset along with the specifications are transferred to a novel, modular ADE engine which can adapt its output accordingly.

Example 1.2. As depicted in Figure 1, Clarice uploads the Netflix dataset to LINX and types a description of her goal: *"Find a country with different viewing habits than the rest of the world"*. LINX then decides that the output exploration session should contain two comparisons of the same group-and-aggregate queries, one when filtering in on a country, and the second when that country is filtered out. These specifications are then inserted into LINX's modular ADE engine. The engine executes a multitude of exploration sessions until converging to an optimal one: Two group-by operations comparing the *rating* and *show-type* (using a *count* aggregation), where each is employed on the results of two filter queries – one by `Country=India`, and the second by `Country!=India`. Observing the output session notebook (See Fig. 1e for a snippet) she quickly derives insights that are relevant to her goal, illustrating how India differs from the rest of the world: (1) *"While the majority of titles in the rest of the world are rated TV-MA (17+), in India, most titles are rated TV-14 (14+)"* and (2) *"In India, the majority of titles are movies (93%), whereas in the rest of the world, movies comprise only 66% of the titles (with the rest being TV shows)"*.

LINX is able to generate such goal-oriented sessions for various goals described in natural language using two main components: A modular ADE framework that takes into consideration custom specifications, and an LLM-based solution for deriving such specifications from a natural language prompt.

1. Modular ADE framework with a dedicated specification language. Building an ADE framework for goal-oriented exploration requires two significant components lacking in existing ADE systems. First, a means to articulate custom exploration specifications, and second, the ability to integrate these specifications in the ADE optimization process. To this end, we first introduce LDX, a formal, intermediate language for data exploration. LDX allows to define the space of *desired, relevant* exploration sessions with useful constructs for setting the structure, syntax, and the contextual relations between the query operations. Importantly, we devise an efficient *verification* engine for LDX, which quickly determines if an output session is compliant with the specifications or not.

Second, we develop a modular ADE engine that takes into account the input specifications when generating an output exploration session. We base our framework on ATNEA [6], an existing, goal-agnostic ADE system using Deep Reinforcement Learning (DRL). Our modular ADE engine contains two components necessary to support custom specifications: (1) A graduate *LDX-compliance* reward signal, based on multiple variations of the LDX verification engine, used for providing a fine-grained numeric score which increases as the session is closer to satisfying all specifications. (2) A *specification-aware* neural network architecture that derives its final structure from the LDX specifications. Our adaptive architecture is inspired by Constrained Deep Reinforcement Learning (CDRL) solutions [13, 62] where the neural network agent is specifically designed to handle additional requirements, such as safety constraints in autonomous driving frameworks [27]. In such systems, an external mechanism is used to override the agent's actions if they are violating the constraints. In our case, rather than overriding actions externally, we encourage the agent to perform compliant queries by dynamically shifting the action distribution probabilities toward queries that are more likely to be included in a specifications-compliant exploration session.

2. LLM-Based solution for deriving exploration specifications from an analytical goal. As previously mentioned, LINX users specify their goal in natural language, meaning that they do not need to compose LDX specifications, but rather, these are derived directly from the analysis goal description. Our solution receives as input the user's goal as well as a short description of the dataset, and derives from it a syntactically correct LDX specification (This part is crucial as our modular ADE engine utilizes the LDX verification engine). Unlike more common tasks such as Text-to-SQL, for which LLMs demonstrate superior performance, for our task there is hardly any available resources in the LLMs' training data (see discussion in Section 2). To overcome the absence of NL-LDX information in the LLM training data, we use a *few-shot* setting [48, 77], coupled with *intermediate code representation* [12, 45, 79]: Instead of directly instructing the LLM to generate LDX specifications, we adopt a two-stage prompting approach. In the first prompt, the LLM is tasked with expressing the specifications as a non-executable Python Pandas [78] code. In the second stage, an additional prompt instructs the LLM to translate the resulting code into formal LDX specifications. As LLMs are trained on vast amounts of Python code, this intermediary step significantly improves their final performance.

Experimental Evaluation & Benchmark Dataset. To evaluate LINX, we constructed the first benchmark dataset, to our knowledge, for goal-oriented data exploration. Our benchmark contains 182 pairs of analytical goals and corresponding exploratory specifications, over three different datasets. We then conducted a thorough user study involving 30 participants to evaluate the relevance and overall quality of LINX exploration sessions. We compared LINX sessions to ones generated by ATENA [6], to sessions generated directly by ChatGPT [51], as well as to ones generated by the Google Sheets ML-Exploration tool [23]. The results are highly positive: Sessions generated by LINX were considered 1.5-2X more useful and allowed users to derive 3-5X more goal-relevant insights than the other automatic baselines.

A recent demo paper [43] briefly introduces LDX and an earlier prototype of its engine, with a main focus on a web interface for manual specification composition. In our current paper we present an end-to-end, tested solution that only requires the user to describe their analytical goal in natural language.

Paper Outline. We begin by surveying related work (§2), then formally define our problem and provide an example workflow of LINX (§3). Next, we describe the LDX language (§4), our CDRL-based modular ADE framework (§5) and the LLM-based solution for specifications derivation (§6). Finally, we present our experimental evaluation (§7) and provide concluding remarks (§8).

2 RELATED WORK

Assistive Tools for Interactive Exploration (Single Step). Assisting users in data exploration has been the focus of numerous previous works. Examples include simplified analysis interfaces for non-programmers [23, 38], explanation systems for exploratory steps, [15, 41], insights extraction [71], and recommender systems for *single* exploratory steps [16, 17, 20, 21, 31, 47, 60, 84]. While these works facilitate the *interactive* aspects of data exploration, LINX focuses on a complementary dimension – generating full exploratory sessions, joining the more recent line of research on ADE, as described below.

Automated Data Exploration (ADE). Rather than assisting users in formulating a single query, more recent systems such as [5, 10, 11, 53, 54] aim to generate an end-to-end exploratory process, given an input dataset, with the purpose of highlighting interesting aspects of the data, and providing thorough preliminary insights. When presented in a notebook interface, such exemplar exploration sessions are highly useful for analysts and data scientists [36, 52, 58].

Due to the vast domain of possible exploration *sessions*, systems such as [5, 6, 53, 54] use powerful optimization tools, such as dedicated *deep reinforcement learning* (DRL) architectures and mathematical solvers. However, as previously mentioned, these systems are *agnostic* to the user's goal, due to their predefined objective function, which makes them generate the same session per dataset. LINX is the first ADE framework, to our knowledge, designed for *goal-oriented* exploration.

Visualization recommendations & insights discovery Adjacent research fields have concentrated on discovering valuable data visualizations [39, 66, 74, 81] and pattern-based insights [18, 44]. Typically, these approaches define notions of *utility* or *importance* then scan the data to identify top-k visualizations or insights with the highest scores [18, 74, 81]. More recent work [44, 65**?**] has focused on organizing these mined patterns to highlight compound patterns within the data and build *data narratives* [64**?**] Although communicating mined insights and knowledge from data is critically important, it is complementary to our work, which is centered on generating a sequence of interconnected queries that slice and dice the data to answer a specific question or an analytical goals.

In addition, numerous specifications languages exist for data visualizations [29, 61, 82]. Similarly to these languages, the LDX language introduced in this work also uses parametric definitions for query operations. However, whereas visualization languages are used for defining a *single visualization*, LDX focuses on defining a *sequence* of exploration queries.

Text-to-SQL. As previously mentioned, LINX uses an LLM-based solution that derives exploratory specifications from a textual description of the user's analytical goal. This task draws some similarities with the well-studied task of *text-to-SQL*, where a structured query is translated from a natural language (NL) request [1, 37]. Recently, Text-to-SQL via LLMs [42, 55] has shown promising results, nearly comparable to dedicated architectures [90]. This is mainly due to the existing resources used for this task, such as supervised datasets like [42, 86, 90], a plethora of academic papers and books, as well as practical tips in programming internet forums. Differently, solving our new task of NL-to-LDX requires overcoming additional challenges: (1) LLMs are not explicitly trained on vast amounts of exploratory sessions, (2) Our task requires specifying a sequence of interconnected queries, rather than a single SQL query, and therefore more difficult to derive solely based on a description of the task and dataset, and (3) rather than generating the full session, the LLM is tasked with *partially* specifying it, thus leaving some of the query parameters to be discovered by the ADE engine. In Text-to-SQL, the NL request is instantly translated to an executable query. We show in Section 7 that exploratory sessions generated *directly* by ChatGPT are significantly inferior to sessions generated by LINX.

LLM Applications in Data Management. The promising generative results obtained by LLMs paved the path to recent research works investigating how to utilize it for data management tasks. Applications (beyond text-to-SQL) include data integration [4], table discovery [19], columns annotation [69] and even the potential of substituting database query execution engines [49, 59, 72, 73]. These exciting research directions are orthogonal to our work.

3 PROBLEM SETTING, EXAMPLE WORKFLOW

We first define the problem of goal-oriented, automated data exploration, then present an example workflow of LINX.

The goal-oriented ADE problem. Given an input dataset D, and an analytical goal description g, we define the task of automatically generating an exploration "session" – A sequence of N consecutive queries: $q_1, q_2, \dots q_N$ conducted in light of the goal g. Collectively, the results of the queries build upon one another, allowing for the derivation of compound insights that are relevant and aligned with the goal g.

As in standard ADE settings, we assume a predefined set of query operation types. Following [6] we focus on the following *parametric* filter, and group-by query operations: A *filter* operation is defined by `[F,attr,op,term]`, where `attr` is an attribute in the dataset, `op` is a comparison operator (e.g. =, ≥, *contains*) and `term` is a numerical/textual term to filter by. A *Group-and Aggregate* operation is defined by `[G,g_attr,agg_func,agg_attr]`, i.e., grouping on attribute `g_attr`, aggregating on attribute `agg_attr` using an aggregation function `agg_func` (we discuss the support of additional operations below).

We further assume a *tree-based exploration* model, following [6, 47], in which each query operation q_i is represented by a node, and is applied on the results of its *parent* operation. The "root" node of the exploration tree is the original dataset before any operation is applied, and the query execution order corresponds with the tree pre-order traversal (See Figure 1d for an example exploration tree). For dataset D, we denote an exploration tree by T_D.

Now, given a *utility score function* for an exploratory session, denoted U, a goal-agnostic ADE system is tasked to generate a session T_D such that $U(T_D)$ is maximal. Multiple such notions are defined in previous work [6, 10, 54]. In LINX, given a dataset D and the goal g, our objective is to generate a session T_D such that $U(T_D)$ is maximal *and* relevant, w.r.t. analysis goal g. In LINX, the relevance of a session is determined according to a set of exploration specifications Q_X, derived w.r.t. the goal g. If $Q_X(T_D) = True$, i.e., the session is compliant with the specifications, then we say it is *relevant* w.r.t. goal g.

Example workflow. Figure 1 illustrates the detailed architecture and an example workflow of LINX, extending Example 1.2. The user uploads a dataset and an analytical task (Fig. 1a), then LINX works using a two step process: (1) It first derives a set of exploration specifications Q_X that form a "skeleton" for the output session. This skeleton accommodates a variety of compatible instances. In the second step (2), our *Modular ADE engine* generates the full session T_D, which maximizes the exploration score $U(T_D)$ (we use the notion from [6], as explained below) and is also compliant with the derived specifications Q_X.

Step 1: Deriving Exploration specifications w.r.t. the goal. We use an LLM-based solution to derive exploration specifications from D and g, expressed in LDX (described in Section 4).

As mentioned above, we use a two-stage prompting approach, to overcome the absence of relevant explicit knowledge in the LLM training data. First, we prompt the LLM to generate *non-executable Python Code*, as depicted in Figure 1b. Note that this is merely an intermediate gateway for expressing the specifications, as this code cannot be executed. In particular, it contains special placeholders (marked with <>) for query parameters that will be later instantiated by the ADE engine, in a manner that maximizes the general exploration score. As described in Example 1.2, LINX takes the goal of finding an atypical country in the Netflix dataset, and derives that the output session should contain filter operations on the 'Country' column – one for a specific country, and the other for the complement data (i.e., the rest of the world), each followed by the same group-and-aggregate operation. See that the specific country and the group-by parameters are *not* specified. These will be instantiated later by the modular ADE engine, which will discover the instances that maximize the exploration utility. Last, the non-executable code is then translated to LDX via a subsequent prompt, as illustrated in Fig. 1c. Returning syntactically correct LDX is crucial, as the LDX verification engine (Section 4.2) is embedded in the ADE optimization process, as explained in Section 5.

Step 2: Generating a maximal-utility exploratory session, in accordance with the goal-driven specifications. In the second step our modular ADE framework performs a CDRL process, as illustrated in Figure 1 (bottom left): optimizing a generic exploration reward (defined in [6]) while ensuring that the output session is compliant with the input specifications. This is enabled due to our *compliance reward scheme* (Section 5.2) that employs multiple instances of the LDX verification engine, and a novel *specification-aware neural network* which adjusts its structure based on the input specifications (Section 5.3).

After the CDRL process converges, LINX produces an *executable exploration tree* (Fig.1d), consisting of executable query operations that adhere to the input specifications while maximizing the generic exploration score. The query parameters marked in red are the ones discovered by the CDRL engine: the country filter value <X> is *'India'*, and the comparison involves a *count* aggregation over the attributes *rating* and *show type*. This exploratory session is then presented to the user as a *scientific notebook*, as depicted in Fig.1e. The notebook snippet demonstrates that the output exploratory session indeed reveals interesting and relevant insights, illustrating how India differs from the rest of the world in terms of the title *ratings* and *types*, as explained in Example 1.2.

In Section 7 we examine the performance and the quality of notebooks generated by LINX for various different goals (see Table 1 for example instances).

Limitations and Scope. We conclude with three remarks on the scope and limitations of LINX:

1. ADE Vs. interactive exploration. Recall that LINX is an ADE system designed to generate comprehensive exploratory sessions, similar to the approaches in [10, 11, 53, 54]. As discussed in Section 2, ADE systems are not intended to replace interactive exploration tools [21, 47, 84]. Instead, their primary role is to be used *before* users engage in interactive data exploration, offering valuable, thorough preliminary insights. This preparatory step is akin to reviewing *human-generated* exploration notebooks found on platforms like Kaggle and Github [36], providing a solid foundation for subsequent analysis. Naturally, due to the vast search space, the output of ADE systems [6, 10] may take several minutes to generate (see the discussion in Section 7.4). However, this longer running time is acceptable, given that ADE systems are not intended for real-time interaction but for providing thorough, preparatory insights.

2. Supported Query Operations. LINX currently supports filter and group-by-and-aggregate queries, with plans to extend these capabilities in future work. While these represent only two types of operations, it is important to note that there are often over 10,000 unique configurations available (e.g., by varying the filter attribute, comparison operator, token, group-by columns, aggregation functions, etc.). As demonstrated in our experimental evaluations in Section 7.3, when these operations are chosen appropriately by LINX, users are able to identify the insights conveyed in the generated views, and accordingly, they rated LINX notebooks as informative, coherent, and relevant to the analytical goal.

3. Future Extension: Spelled-out Insights and Visualizations. LINX currently outputs a sequence of query operations along with their corresponding result views. To obtain more compelling outputs that may expedite insight discovery, one could apply an insights-mining tool [44, 71] or an auto-visualization tool [39] to the views generated by LINX. We plan to integrate such systems into LINX in the future, as well as explore the use LLMs for providing natural language summaries of the resulting exploratory sessions.

4 EXPLORATION SPECIFICATION LANGUAGE

We first describe LDX, our intermediate language designed for explicitly defining a *sub-space* of exploration sessions that can be relevant for the input analysis goal. Importantly, we further introduce an efficient verification algorithm for LDX, which is embedded in our ADE engine (Section 5.2).

4.1 LDX Language Overview

Recall that an exploration session tree T_D comprises a sequence of query operations, where each query q_i is employed on the results of one of the previous queries $q_j, j < i$. LDX therefore allows posing specifications for (1) the session structure, i.e., the shape of the tree which reflects the execution order and the input data for each query, (2) the parameters and type of the queries, and (3) *continuity variables* for controlling how queries are interconnected. The latter aspect is particularly important for exploration sessions examined by users, as the semantic connection between the queries allows building an exploration *narrative* [36, 52] that gradually leads the viewer to nontrivial insights on the data.

Our specification language LDX extends Tregex [40], a query language for tree-structured data[1]. The basic unit in LDX is a *single node specification*, which addresses a particular node (query operation, in our context). A full LDX specifications query is then composed by conjuncting multiple single-node specifications, interconnected using the *continuity variables.* We begin with a simple "hello world" example, then describe LDX constructs in more detail.

Example 4.1. The following LDX query describes a simple exploration session "skeleton" with two query operations: a group-by, followed by a filter operation, both employed on the full dataset (the root node in T). It also specifies that the filter is to be performed on the same attribute as the group-by. The rest of the parameters are left *unspecified.*

```
ROOT CHILDREN <A,B>
    A LIKE [G,(?<X>.*),.*]
    B LIKE [F,(?<X>.*),.*]
```

The query contains three *named-nodes* – `ROOT`, `A` and `B`, each is differently specified. The `ROOT` node represents the raw dataset, has two immediate children `A` and `B` – the group-by and filter operations (both use it as input data). `A` is a group-by with "free" parameters: unspecified group attribute, aggregation function, and aggregation attribute, and `B` is a filter operation with unspecified operator and term (Recall the parametric definition of queries in Section 3). Unspecified parameters are marked with `*`, but see that the *attribute* parameter in both query operations is marked with `(?<X>.*)`. This means that `X` is a *continuity variable* that ensures that both operations need to use *the same* column parameter.

We next briefly describe the constructs of LDX (Full description and more examples are provided in [56]).

Specifying exploration tree structure. The session structure is specified via tree-structure primitives such as `CHILDREN` and `DESCENDANTS`. For instance, `'A CHILDREN <B,+>'` states that Operation `A` has a subsequent operation named `B`, and at least one more (unnamed) operation, as indicated by the + sign. Importantly, recall that the fact that `B` is a child of `A` not only means that Operation `B` was executed after Operation `A`, but also that `B` is employed on the results of Operation `A` (i.e., rather than on the original dataset).

Specifying query operation parameters. LDX allows for partially specifying the operations using *regular expressions* (regex), as they define match patterns that can cover multiple instances. For example, the expression `'A LIKE [G,'country',SUM|AVG, *]'` specifies that Operation `A` is a *group-by* on the attribute *country*, showing either *sum* or *average* of *some* attribute (marked with $*$).

Continuity Variables. We next introduce the continuity variables in LDX, which allow constructing more complex specifications that *contextually* connect between operations' free parameters once instantiated. LDX allows this using named-groups [2] syntax. Yet differently than standard regular expressions, which only allow "capturing" a specific part of the string, in LDX these variables are used to constrain the operations in subsequent nodes. For instance, the statement '`B1 LIKE [F,'country',eq,.*]`' (taken from the LDX query in Figure 1c) specifies that Operation `B1` is an *equality filter on the attribute 'country', where the filter term is free.* To capture the filter term in a continuity variable we use named-groups syntax: '`B1 LIKE [F,'country',eq,(?<X>.*)]`' – in which the free filter term `(.*)` is captured into the variable `X`. Using this variable in subsequent operation specifications will restrict them to the same filter term (even though the term is not explicitly specified). For instance, as shown in Figure 1c, the subsequent specification is '`B2 LIKE [F,'country',neq,(?<X>.*)]`', indicating that the next filter should focus on all countries *other* than the one specified in the previous operation. In [56] we provide an LDX technical guide, containing full details, as well as additional LDX query examples for various different goals.

4.2 LDX Verification Engine

We next describe our LDX verification engine, which takes an exploration session tree T_D and a LDX specifications query Q_X, and verifies whether T_D is compliant with Q_X. The verification engine, as described in Section 5.2, is integrated into the modular ADE engine of LINX (in several variations) where it provides real-time *compliance* reward on the generated sessions. By optimizing on both the compliance reward *and* the generic exploration reward, the

[1] Tregex natively allows partially specifying structural properties of the tree, as well as the nodes' labels, yet is missing the definitions of *continuity.*

ADE agent is able to produce useful sessions that are also compliant with the LDX specifications derived from the user's analytical goal.

For an input LDX query Q_X, we denote the set of its *named nodes* (e.g., nodes `ROOT`, `A` and `B` in Example 4.1) by $Nodes(Q_X)$, and the set of its continuity variables by $Cont(Q_X)$. We first define an LDX assignment, then describe our verification procedure that searches for valid assignments.

Definition 4.2 (LDX Assignment). Given an LDX query Q_X and an exploration session tree T_D, an *assignment* $A(Q_X, T_D) = \langle \phi_V, \phi_C \rangle$, s.t., (1) ϕ_V is a *node mapping function*, assigning each named node $n \in Nodes(Q_X)$ an operation node $v \in V(T_D)$ in the exploratory session T. (2) ϕ_C is a *continuity mapping function*, assigning each continuity variable $c \in Cont(Q_X)$ a possible value. The initial node mapping is $\phi_V(\texttt{ROOT}) = 0$, i.e., mapping the root node in the LDX query to the root node of T_D.

LDX Verification Algorithm. Recall that an LDX query Q_X comprises a set of *single node specifications*, s.t. each specification $s \in Q_X$ refers to a single named node in Q_X. We denote the named node of s by $Node(s)$, and the (possibly empty) set of continuity variables in s by $Cont(s)$. The LDX verification algorithm, as depicted in Algorithm 1, takes as input an LDX specifications query Q_X, an exploration tree T_D, and the initial assignment A, in which ϕ_V contains the initial root mapping (Definition 4.2) and an empty continuity mapping ϕ_C, and returns `true` if there exists at least one valid assignment $A(Q_X, T_D)$. Note that since Tregex does not support continuity variables, we can only use its node matching function `GetTregexNodeMatch` [22] in our algorithm. This function, as described in [22], takes as input a single specification s, a tree T, and the current node mapping ϕ_V and returns all valid node matches for $Node(s)$, denoted V_T^s, given the current state of the node mapping ϕ_V.

Our verification procedure, as described in Algorithm 1 works as follows. In each recursive call, a single specification s is popped from Q_X (Line 2). Then, s is updated with the continuity values according to ϕ_C (Lines 2-4): if a continuity variable c is already assigned a value in ϕ_C, we update the instance of c in s, denoted $s.c$, with the corresponding value $\phi_C(c)$. Next (Line 5), when all available continuity variables are updated in s, we use the Tregex `GetTregexNodeMatch` function, to obtain a set V_T^s of possible valid assignments for $Node(s)$. Then, for each node $v \in V_T^s$, we first update the node mapping ϕ_V (Line 7) and the continuity mapping ϕ_C (Lines 7-9): we assign each continuity variable c the concrete value of c from v, denoted $v.c$. (Recall that v already satisfies s also w.r.t. ϕ_C, therefore only unassigned variables in $Cont(s)$ are updated.) Once both mappings are updated (denoted ϕ_V^s and ϕ_C^s), we make a recursive call to *VerifyLDX* (Line 10), now with the shorter Q_X (after popping out s) and the new mappings (ϕ_V^s, ϕ_C^s). Finally, the recursion stops in case there is no valid assignment (Line 12) or when Q_X is finally empty (Line 1). In Section 5 we describe how multiple variations of the LDX verification algorithm are used within the optimization process of our modular ADE engine.

5 CDRL FRAMEWORK FOR MODULAR ADE

Recall that ADE systems optimize over the domain of all possible exploration sessions, thus requiring powerful optimization tools [6, 11, 53]. We base our modular ADE engine on the goal-agnostic Deep Reinforcement Learning (DRL) framework for data exploration presented in [6]. In the DRL setting, a neural-network agent produces a maximal-scoring session (using a predefined exploration reward function) by employing a multitude of intermediate sessions, then updating its internal policy according to their obtained scores until converging to an optimal one.

Different than [6], our *modular* ADE framework takes a given dataset D, as well as LDX specifications Q_X, and generates a high-scoring exploration session T_D which is in compliance with Q_X. The main challenge which arises here is to effectively embed the specifications as a part of the optimization process. A naive integration would have been to incorporate, in addition to the generic exploration score, a binary score derived from the result of the verification engine for each generated session (i.e., compliant/non-compliant). However, this naive solution introduces a *reward sparsity* problem [46], a prominent challenge in reinforcement learning arising when the agent scarcely obtains a positive feedback, thus failing to converge. Our experimental evaluation in Section 7.4 indeed shows that such a solution fails to converge on *all* tested LDX queries. We next overview our solution, based on Constrained Deep Reinforcement Learning (CDRL) [13, 62].

CDRL Framework Overview. To effectively embed the specifications in the optimization process we use a twofold solution: First, we introduce a flexible compliance reward scheme that gradually guides the DRL agent towards fully compliant sessions by encouraging it to first generate structurally compliant sessions (learning the queries type and order of execution), and only then refine individual query parameters. Then, we devise a novel neural network architecture, inspired by intervention-based CDRL [13, 62]. In these CDRL systems an external mechanism is used to override the agent's actions if they are violating the constraints. In our case, we cannot always detect a violation immediately, and verify the compliance only at the end of a session. Thus, rather than overriding actions externally, we internally encourage the agent to perform compliant operations via a novel *specification-aware* network architecture, pushing query parameters that are likely to comply with the specifications with a higher probability. We show in Section 7.4 that only the combination of these two solutions allow LINX to successfully and consistently converge.

We next define the Markov Decision Process (MDP) model used in our CDRL framework, then delve into the LDX-compliance reward scheme and specification-aware network.

5.1 MDP Model

Following [6] we use an episodic MDP model in our CDRL engine, defined as $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \Delta_a, R_a)$, where $\mathcal{S}$ is a state space; $\mathcal{A}$ is an action space; $\Delta_a : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is a transition function that returns the outcome state S' obtained from employing an action a in state S; and $R_a(S, a)$ is the *reward* received for action a in state S.

Our MDP model is defined as follows: Given a dataset D and specifications Q_X, the agent produces an exploration session T_D on D in each episode. At each step i, the agent employs a parametric query operation q_i, as defined in Section 3. After executing an operation, the agent transitions to state $S_i = \Delta_a(S_{i-1}, a)$, where S_i represents the resulting view of q_i. In addition to query operations,

Algorithm 1: LDX Query Compliance Verification

```
VerifyLDX ($T_D, Q_X, A = \langle \phi_V = \{\texttt{ROOT:0}\}, \phi_C = \emptyset \rangle$)
  // Inputs: Exploration tree $T_D$, LDX Specifications $Q_X$,
  assignment $A$
    if $Q_X = \emptyset$ then return True
    $s \leftarrow Q_X.pop()$ // pop a single node specification from $Q_X$
    for $c \in Cont(s)$ do              // Assign continuity vars in $s$
        if $c \in \phi_C$ then $s.c \leftarrow \phi_C(c)$
    $V_T^s \leftarrow$ GetTregexNodeMatches$(s, T, \phi_V)$
    for $v \in \mathcal{V}_T^s$ do
        $\phi_V^s \leftarrow \phi_V \cup \{Node(s) : v\}, \phi_C^s \leftarrow \phi_C$
        for $c \in Cont(s)$ do        // Update continuity mapping
            $\phi_C^s(c) \leftarrow v.c$
        if VerifyLDX$(T_D, Q_X, \langle \phi_V^s, \phi_C^s \rangle)$ then
            return True
    return False
```

the agent can use a *back* operation to return to a previous state and start a new action from there.

For each action, the agent receives a bi-objective reward:

$$R_a(S_i, a) := \alpha \cdot R_{gen}(S_i, a) + \beta \cdot R_{comp}(S_i, a, Q_X)$$

The first reward component R_{gen} is based on the generic exploration reward defined in [6]. It is a weighted sum of the interestingness scores of the session's individual queries and their diversity: $R_{gen}(S_i, a) = \mu \cdot \sum_{j \leq i} \text{Interestingness}(q_j) + \lambda \cdot \text{Diversity}(S_i)$. As described in [6], interestingness scores are calculated using *KL-divergence* for filter operations and *conciseness* [28] for group-by-and-aggregate operations. The diversity of the session S_i is measured by computing the minimal distance between q_i and a previous query $q_j, j \leq i$ (using the query results distance provided in [6]).

The second component, $R_{comp}(S_i, a, Q_X)$, is a compliance reward unique to LINX. This reward is based on the input LDX specifications Q_X and is described in more detail below.

5.2 LDX-Compliance Reward Scheme

Given LDX specifications Q_X, and an exploratory session T_D, we define our *compliance* reward signal, received at each step i:

$$R_{comp}(S_i, a, Q_X) := \gamma \cdot EOS(S_i, a, T, Q_X) + \delta \cdot IMM(S_i, a, T_D^i, Q_X)$$

Here *EOS* is an *end-of-session* feedback, equally divided across all query operations; and *IMM* is received immediately, per operation. These signals provide a fine-grained feedback, allowing our CDRL engine to overcome the *reward sparsity problem* described above.

End-of-Session Compliance Reward. The EOS reward component $EOS(S_i, a, T, Q_X)$ is received at the end of an episode (once the exploration session T_D is fully generated), then equally distributed across all states S_i. We utilize the LDX verification engine (Algorithm 1), but in light of the observation that *structural specifications should be learned first.* Intuitively, if the agent learns to generate *correct* query operations in an *incorrect* order/structure, the learning process becomes largely futile as reordering requires the agent to relearn the session from scratch. We therefore partition the set of individual node specifications in Q_X to *structural* and *operational* subsets, denoted by $struct(Q_X)$ and $opr(Q_X)$, s.t. $struct(Q_X) \cup opr(Q_X) = Q_X$. $struct(Q_X)$ refer to the definitions of the session tree structure and $opr(Q_X)$ to the definition of query operation parameters, as described in Section 4.

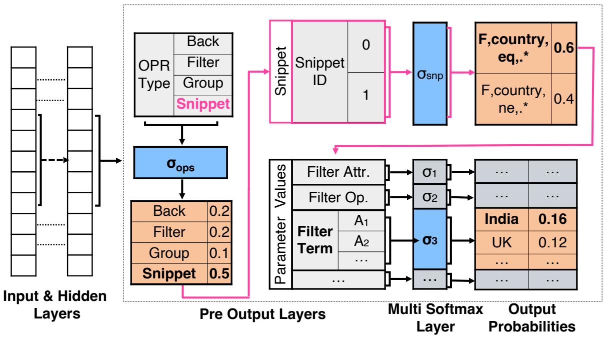


Figure 2: Specification-Aware Network Architecture

Briefly, our end-of-session rewards works as follows (See Appendix A.3 for full details). First, we use Algorithm 1 to check if T_D complies with Q_X. Then, a conditional reward is granted, according to the following three cases: (1) If fully compliant, a high positive reward is given. (2) If T_D is not compliant with Q_X, we check its compliance only with $struct(Q_X)$, the structural specifications in Q_X. If no valid assignments are found, a fixed negative penalty is applied. (3) If T_D satisfies $struct(Q_X)$ but not $opr(Q_X)$, i.e., the operational specifications, a non-negative reward is assigned based on the number of satisfied query parameters (The larger the number of satisfied parameters, the higher the reward). Intuitively, this reward enforces the learning of correct structure by imposing a high penalty for non-compliant sessions. Once the correct structure is learned, the agent receives gradually increasing rewards to encourage satisfaction of operational specifications. Upon generating a fully compliant session, the agent receives a high positive reward.

Immediate (per-operation) Compliance Reward. To reinforce adherence to structural constraints, we introduce an *immediate* reward signal $IMM(S_i, a, T_D^i, Q_X)$ granted individually for each step i. This real-time signal negatively rewards specific operations that violate the structural specifications $struct(Q_X)$. To do so, we use a modification of the LDX verification engine (Algorithm 1), that can operate on an *ongoing* session T_D^i (in step i) rather than a full session T_D. Intuitively, we assess the possibility of a *future* assignment satisfying $struct(Q_X)$ in up to $N - 1$ more steps. This is done by attempting to extend the exploration tree with $N - i$ additional "blank" nodes, respecting the order of query operations execution. In case no valid assignment is found to any of the new trees, a negative reward is granted. The number of possible tree completions throughout an N-size session is bounded by C_N, the Catalan number (See Appendix A.3 for full details). In practice, we show in Section 7.4 that this reward poses a negligible computational overhead on the optimization process.

Balancing the reward components. LINX employs multiple reward components, which may exhibit different convergence rates due to variations in value domains and sparsity levels. To address this, as is common in multi-objective reinforcement learning [**?**], one can increase the weight of a reward signal that converges poorly. However, in our experiments, we used equal weights and observed balanced convergence across all signals.

5.3 Specification-Aware Neural Network

We next describe our specification-aware architecture used to increase the probability of choosing compliant operations. Our neural network modifies its structure according to the input LDX specifications, by creating special segments for operation "snippets" likely to be compatible with LDX. The agent can uses these snippets more frequently, thus advance faster toward a fully compliant session.

Figure 2 depicts the network architecture (Specifications-aware functionality is highlighted in pink). First, rhe input layer receives an observation of the current state S_i in the MDP model, and passes it to the dense hidden layers. Then, the agent composes a query operation via the pre-output layers, where it first chooses an operation type and subsequently its corresponding parameters. As depicted in Figure 2, Softmax Segment σ_{ops} is connected to the operation types, and Segments $\sigma_1, \sigma_2, \ldots$ are connected to the value domain of each parameter.

In our architecture, we add a new high-level action, called "snippet" (σ_{snp}). When choosing this action, the agent is directed to select a particular snippet that is derived from the operational specifications $opr(Q_X)$. The snippets function as operation "shortcuts", which eliminate the need for composing full, compliant operations from scratch. For example, using a snippet of `‘F, Country, eq‘` (See Fig. 2), only requires the agent to choose a filter term, rather than composing the full query operation.

Given an LDX qery Q_X, the network architecture is derived as follows. First, we generate an individual *snippet* neuron for each operational specification $s \in opr(Q_X)$ (In case the regular expression in s contains a disjunction, we generate an individual snippet for each option) All snippet neurons, as depicted in Figure 2, are connected to σ_{snp}, the snippet multi-softmax segment. Now, to choose the "free" parameter, unspecified in s, the snippet neuron is wired to the corresponding parameters in the multi-softmax segments. For instance, the snippet of `‘F, Country, eq‘` is wired to σ_3 for choosing a filter *term* parameter, as depicted in Fig. 2. In Appendix A.4 we provide an example such derivation process.

In combination with the reward scheme presented earlier, our network architecture allows LINX to consistently generate compliant exploration sessions in 100% of the datasets and LDX queries in our experiments, as detailed in Section 7.4.

6 LLM-BASED SOLUTION FOR DERIVING EXPLORATION SPECIFICATIONS

As previously mentioned, LINX users do not need to manually compose LDX specifications, but only provide a description of their analytical goal g. Deriving LDX specifications for the CDRL engine is done via an LLM-based solution: We use a *few-shot* setting, renowned for its excellent performance across diverse analytical goals [48, 77]. In this approach, several illustrative examples are provided to the LLM before soliciting task completion. Then, to further enhance the distillation of exploration specifications, we also use a solution based on *intermediate code representation* [12, 26, 45, 89], where instead of directly instructing the LLM to generate LDX specifications, we adopt a two-stage chained prompt: In the first prompt, the LLM is tasked with expressing the specifications as a non-executable, template Python Pandas [78] code, restricted to the operations supported by LINX. The template code (See Figure 1a) contains special placeholders representing the query operations (or specific parameters) to be discovered in a data-driven manner. In the second stage, an additional prompt instructs the LLM to translate the intermediate Pandas code into formal LDX specifications. We coin our approach *NL2PD2LDX*. We further considered a solution based on an intermediate SQL representation instead of Python; however, it yielded subpar results (see our full version in [56] for more details).

Recall again that the last conversion to LDX is *required* in LINX, due to its efficient verification engine embedded in the CDRL process. As we empirically show in Section 7.2, our two-stage approach exhibits superior generalization compared to a direct NL-to-LDX approach, and when combined with the CDRL engine described above, it produces exploratory sessions that are deemed more useful and insightful than other baselines such as ChatGPT and ATENA [6].

Prompt Engineering. Figure 3 depicts a snippet of our chained prompts: NL-to-Pandas and Pandas-to-LDX.

NL-to-Pandas. The prompt is structured into three main components: (1) NL-to-Pandas task description; (2) a series of few-shot examples; (3) the test analysis goal alongside a small dataset sample. Each few-shot example in (2) comprises several steps: (a) example analytical goal; (b) dataset and schema description (e.g., `epic_games` in Fig. 3) ; (c) the correct Pandas code template for the task; (d) an NL explanation of the output. Including dataset information is motivated by past work in text-to-SQL [8, 75].

Step (d) is influenced by the Chain-of-Thought (CoT) prompting paradigm, which has demonstrated enhanced performance in multi-step tasks [70, 76, 77, 85]. Following the CoT methodology, we incorporate an explanation for each few-shot example. We use *least-to-most prompting* [91], in which we provide the examples at an increasing level of difficulty. Hence, we gradually "teach" the LLM fundamental concepts before progressing to more intricate examples. Finally, in part (3), we describe the analytical goal along with a sample of the first five rows of the input dataset.

Pandas-to-LDX. For the Pandas-to-LDX prompt, its structure mirrors the previous prompt, i.e., first presenting the Pandas-to-LDX translation task, few-shots examples, etc. This time, we omit the dataset information (2.b) as it is redundant for this simpler task.

LDX Syntax Correction Prompt Finally, since the CDRL engine cannot operate if the LLM returns a syntactically incorrect query, we designed a syntax-correction prompt for such cases. This prompt includes a more detailed description of LDX along with a few examples of common syntax errors and their corrections.

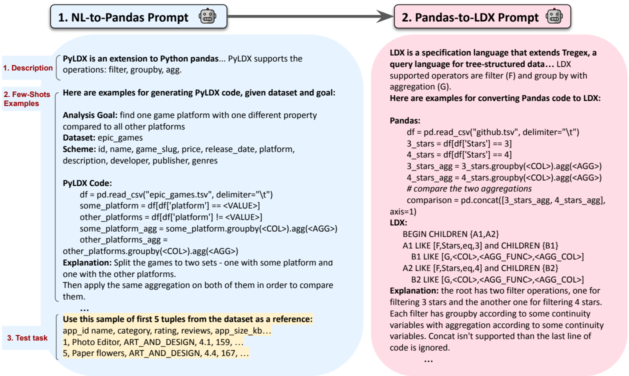
The full versions of all prompts are provided in [56].

7 EXPERIMENTS

We implemented LINX in Python 3: The LDX verification engine utilizes the Tregex Python implementation in [22], and our CDRL engine is built in ChainerRL [25], based on the DRL framework for data exploration, publicly available in [30]. All our experiments were run on an Intel Xeon CPU-based Linux server with 24 cores and 96 GB of RAM. The full experiments code and data are provided in our Github repository [56].

	Exploration Meta Goal	Example (concrete) Goal	# Ex.
1	Identify an uncommon entity	g_1: “Find an atypical country” (NETFLIX)	18
2	Examine a phenomenon (subset)	g_2: “Examine characteristics of successful TV shows” (NETFLIX)	16
3	Discover contrasting subsets	g_3: "Find three actors with contrasting traits” (NETFLIX)	22
4	Survey an attribute	g_4: "Survey apps’ price” (PLAY STORE)	21
5	Describe an unusual subset	g_5: "Highlight distinctive characteristics of summer-month flights” (FIGHTS)	27
6	Investigate various aspects of an attribute	g_6: "Investigate reasons for delay” (FLIGHTS)	22
7	Explore through a subset	g_7: "Analyze the dataset, with a focus on flights affected by weather-related delays” (FLIGHTS)	28
8	Highlight interesting sub-groups	g_8: "Highlight interesting sub-groups of apps with at least 1M installs” (PLAY STORE)	28

Table 1: Overview of the Goal-Oriented ADE Benchmark (182 Instances)



PyLDX is an extension to Python pandas... PyLDX supports the operations: filter, groupby, agg.

Here are examples for generating PyLDX code, given dataset and goal:

Analysis Goal: find one game platform with one different property compared to all other platforms
Dataset: epic_games
Scheme: id, name, game_slug, price, release_date, platform, description, developer, publisher, genres

PyLDX Code:

```
      df = pd.read_csv("epic_games.tsv", delimiter="\t")
      some_platform = df[df['platform'] == <VALUE>]
      other_platforms = df[df['platform'] != <VALUE>]
      some_platform_agg = some_platform.groupby(<COL>).agg(<AGG>)
      other_platforms_agg =
other_platforms.groupby(<COL>).agg(<AGG>)
```

Explanation: Split the games to two sets - one with some platform and one with the other platforms.
Then apply the same aggregation on both of them in order to compare them.

…

Use this sample of first 5 tuples from the dataset as a reference:
app_id name, category, rating, reviews, app_size_kb…
1, Photo Editor, ART_AND_DESIGN, 4.1, 159, …
5, Paper flowers, ART_AND_DESIGN, 4.4, 167, …

LDX is a specification language that extends Tregex, a query language for tree-structured data… LDX supported operators are filter (F) and group by with aggregation (G).
Here are examples for converting Pandas code to LDX:

Pandas:

```
      df = pd.read_csv("github.tsv", delimiter="\t")
      3_stars = df[df['Stars'] == 3]
      4_stars = df[df['Stars'] == 4]
      3_stars_agg = 3_stars.groupby(<COL>).agg(<AGG>)
      4_stars_agg = 4_stars.groupby(<COL>).agg(<AGG>)
      # compare the two aggregations
      comparison = pd.concat([3_stars_agg, 4_stars_agg],
axis=1)
```

LDX:

```
    BEGIN CHILDREN {A1,A2}
    A1 LIKE [F,Stars,eq,3] and CHILDREN {B1}
       B1 LIKE [G,<COL>,<AGG_FUNC>,<AGG_COL>]
    A2 LIKE [F,Stars,eq,4] and CHILDREN {B2}
       B2 LIKE [G,<COL>,<AGG_FUNC>,<AGG_COL>]
```

Explanation: the root has two filter operations, one for filtering 3 stars and the another one for filtering 4 stars. Each filter has groupby according to some continuity variables with aggregation according to some continuity variables. Concat isn't supported than the last line of code is ignored.

…

Figure 3: Examples of the chained prompts: (1) NL to non-executable Pandas code, and (2) Pandas code to LDX

To evaluate LINX, we first developed a benchmark dataset for goal-oriented ADE, containing 182 analytical goals and corresponding LDX queries (see §7.1). We then conducted three experiments sets, evaluating LINX’s success in deriving correct LDX specifications (§7.2); relevance and usefulness of auto-generated exploratory sessions (§7.3); and performance and ablation study of our CDRL-based modular ADE engine (§7.4)

7.1 Benchmark Dataset for Goal-oriented ADE

We constructed, to our knowledge, the first benchmark dataset for goal-oriented exploration specifications. Our dataset comprises 182 instances of the form $\langle D, g, Q_X \rangle$, i.e., a dataset, analytical goal, and corresponding exploration specifications in LDX. Our benchmark dataset was created by analyzing and extrapolating from real-life exploration notebooks across three different tabular datasets: *(1) Netflix Titles Dataset* [34], *(2) Flight-Delays Dataset* [32], and *(3) Google Play Store Apps* [33].

To build the benchmark dataset, we first examined publicly available exploration notebooks for the Netflix, Flights, and Playstore datasets (see [32–34]). We specifically focused on identifying notebooks that are goal-oriented, rather than those performing generic exploratory data analysis (EDA) aimed at showcasing general characteristics of the dataset. The notebooks were then manually filtered to select complete, high-quality notebooks (with at least 10 votes), resulting in a set of 36 different notebooks. Inspired by the data exploration surveys in [80] and [3], which identify common analytical tasks, including several open-ended exploration goals, we clustered the 36 notebooks into 8 distinct “meta-goals”. For example, a notebook with the specific goal of *“discover an unusual country in the Netflix dataset”* was associated with the meta-goal of *“discover an unusual subset of the data”*. Table 1 lists the meta-goals we identified, along with a representative concrete goal used in a real-life Kaggle notebook. While the meta-goals are not strictly orthogonal, they were chosen to provide a broad range of real-life use cases.

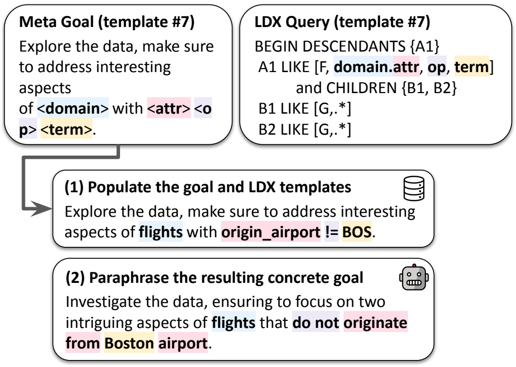
We then composed LDX queries for the goals $g_1 - g_8$ (see Table 1), based on their associated notebooks. This was done manually by refactoring the notebook code to use filter and group-and-aggregate queries, and by abstracting “discovery”-related parameters that cannot be determined solely based on the goal itself (e.g., the specific “country” value to filter on, and the attributes to group by in our running example; see Figure 1c and the description in Section 3). This process resulted in 8 instances of analytical goals and their corresponding exploration specifications.

Then, to extend our dataset from 8 instances to 182, we adopted the scheme outlined in Figure 4. First, for each instance $\langle D, g, Q_X \rangle$, we first stripped g and Q_X from any trait stemming from the dataset D such as attribute names, aggregative operations, and predicates defining data subsets, thus creating “template” goal descriptions and LDX queries. Next, we populated the goal and LDX templates by randomly incorporating values from our three datasets. For instance, the templates in Figure 4, associated with Meta-Goal 7 (see Table 1) are populated using the `Flights` [32] data domain, the `origin_airport` attribute, operator ≠, and the term `‘BOS’` (the populated LDX template is omitted for brevity). Next, since the populated goal description templates may sound unnatural, we utilized an LLM-based paraphrasing approach (implemented with ChatGPT), to obtain goals that are more naturally phrased. Finally, out of 200 generated analytical goals, we discarded 18 nonsensical goals, that did not reflect a realistic user intent. Table 1 lists total number of instances for each meta goal (see [56] for full details).

7.2 Specifications Derivation Performance

We first gauge the effectiveness of our LLM component in deriving correct LDX specifications (Full output sessions are evaluated in Section 7.3). We analyze the LDX derivation performance in four experimental scenarios, varying whether the dataset or meta-goals are seen or unseen in the few-shot prompt examples. We compare the results of our two-stage solution to a single prompt approach that generates LDX directly.

Experimental Settings. We now detail the evaluation measures and provide an overview of the different scenarios and baselines.
Evaluation Metrics. Evaluating text generation quality is a known challenge with various approaches [35, 83, 87]. For example, Text-to-SQL performance assessments often rely on query execution results [57, 63, 83], but this is unsuitable for LDX specifications



Meta Goal (template #7)

Explore the data, make sure to address interesting aspects of <domain> with <attr> <op> <term>.

LDX Query (template #7)

```
BEGIN DESCENDANTS {A1}
 A1 LIKE [F, domain.attr, op, term]
       and CHILDREN {B1, B2}
 B1 LIKE [G,.*]
 B2 LIKE [G,.*]
```

(1) Populate the goal and LDX templates

Explore the data, make sure to address interesting aspects of **flights** with **origin_airport != BOS**.

(2) Paraphrase the resulting concrete goal

Investigate the data, ensuring to focus on two intriguing aspects of **flights** that **do not originate from Boston airport**.

Figure 4: Benchmark Dataset Generation

as they span a multitude of compliant output sessions. Alternative measures include exact string match [24, 87], and graph edit distance, commonly used for graph semantic parsing tasks [9, 35]. Drawing inspiration from these, we introduce two measures for comparing the generated LDX queries against the ground-truth queries: the *Two-way Levenshtein Distance*, which considers the query strings, and the *Exploration Tree Edit Distance* from [47], focusing on the parsed output of the queries. Both measures are designed to be flexible enough to accommodate *equivalent* queries (i.e., meet the analysis goal but not identical to the ground-truth) without imposing a heavy penalty.

(1) Two-way Levenshtein distance (lev^2). Levenshtein distance is commonly used to measure the character overlap between two strings. However, its standard implementation falls short in the context of LDX, as two queries may be conceptually similar but differ, for instance, in the order of operations. To address this limitation, we computed the string distance separately for structural and operational specifications, and then aggregated the two scores. The structure score, denoted as $\overline{lev}(Q_{struct}, Q'_{struct})$, represents the normalized Levenshtein score when omitting operational specifications. The operational distance is defined by $\frac{1}{|Q_{opr}|} * \sum_{o \in Q_{opr}} min_{o' \in Q'_{opr}} \overline{lev}(o, o')$. In this expression, Q_{opr} and Q'_{opr} are sets of operational specifications in the two compared LDX queries. We sum the distance scores, for each operation o in Q_{opr} of the most similar operation in the compared LDX query Q'_{opr}, and then divide the result by the size of Q_{opr}. The final lev^2 is computed as the harmonic mean of the inverses of each score.

(2) Exploration Tree Edit Distance ($xTED$). We employ the exploration tree edit distance, proposed in [47]. This measure augments the tree edit distance [88] function with a dedicated label distance notion to assess the distinction between two query operations (See [47] for full detail). To apply this metric, we construct a minimal tree for each compared LDX query while masking the continuity variables (see Appendix B.2 for more details).

Since both lev^2 and $xTED$ scores represent normalized distance functions, we consider their complements (i.e., $1 - score$), where a higher value is indicative of better performance. If a generated query has incorrect syntax, it receives a score of 0.

Scenarios and Baselines. We conducted four distinct experimental scenarios involving the presence or absence of dataset and meta-goals in the few-shot prompts. In each scenario, the model receives a *test* analytical goal and dataset (selected from the 182 instances in Table 1) and asked to generate appropriate LDX specifications. In the simplest scenario *(1) seen dataset and meta-goal*, the prompts, as described in Section 6, include few-shot examples over the same test dataset and meta-goal associated with the test goal (*excluding* the test goal itself). In the subsequent scenarios *(2) seen dataset, unseen meta-goal* and *(3) unseen dataset, seen meta-goal*, prompts include examples from the same dataset, excluding the associated meta-goal, and vice versa. In the most challenging scenario *(4) unseen dataset and meta-goal*, few-shot examples are provided from different datasets and different meta-goal compared to the test goal. Importantly, in *no scenario* the model obtains an example of the exact same analysis goal used in the test.

To evaluate the efficacy of our NL2PD2LDX solution, we contrasted it with a direct NL2LDX prompt, where the LLM directly generates LDX specifications (See Appendix B.1). We assessed the performance for both ChatGPT (gpt-3.5-turbo)[51] and GPT-4 [50] (for both LLMs we used 0 temperature to obtain consistent results).

Results. Table 2 presents the results for both ChatGPT and GPT-4, with and without our chained prompt solution (denoted +PD in the table), across all four scenarios. First, in the easiest Scenario 1 (*(1) seen dataset and meta-goal*), both LLMs perform well, with GPT-4 achieving optimal results as expected. See that the chained prompt solution exhibits negligible impact, suggesting that the presence of the meta-goal within the prompt allows for easy overfitting, reducing the need for an intermediary solution. In Scenario 2 (*seen dataset, unseen meta-goal*), the performance of both LLMs decreases as the few-shot examples diverge from the test task. Here, a significant improvement (more than 5 points) is achieved by employing our NL2PD2LDX solution for both models, with GPT-4+PD yielding the best results. Moving to Scenario 3 (*unseen dataset, seen meta-goal*), see that the overall performance is better than in Scenario 2, as both LLMs tend to generalize better to unseen datasets than to unseen meta-goals. While our chained solution boosts ChatGPT results by more than 5 points, GPT-4 still achieves the highest score, almost on par with the results in Scenario 1. Lastly, in the most challenging scenario 4 (*unseen dataset, unseen meta-goal*), the chained solution yields higher scores for both LLMs, with GPT-4+PD slightly outperforming ChatGPT+PD.

Summary. Our experimental results demonstrate the effectiveness of our solution for both ChatGPT and GPT-4. Moreover, when using NL2PD2LDX, the performance gap between the two LLMs narrows. Therefore, if cost reduction is a priority, ChatGPT can be a viable option with only a minor compromise on accuracy.

7.3 Relevance and Quality (User Study)

We next evaluate the overall quality and relevance of exploration sessions generated by LINX, compared to sessions generated by alternative baselines. We conducted both a *subjective* study, were users are asked to rate the output sessions according to numerous criteria, and an *objective* study, where we measured users' performance in inferring relevant insights w.r.t. the analytical goal.

Experiment Setup. We recruited a total of 30 participants, by publishing a call for CS students or graduates that are familiar with data analysis yet are not subject matter experts. We then selected 12

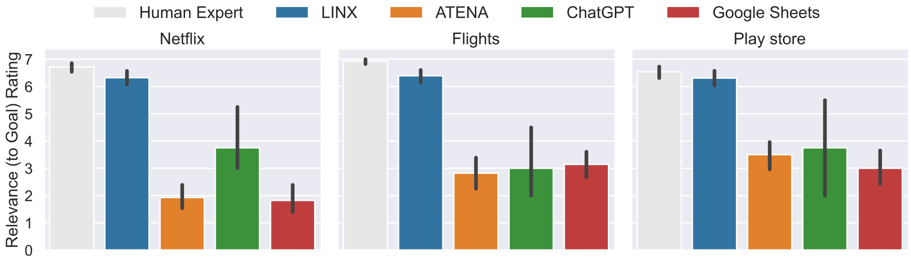


Figure 5: User Study – Relevance Rating of Exploration Notebooks to the Given Goal

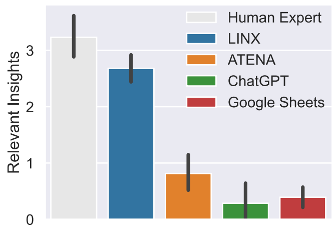


Figure 6: Avg. Num. Insights

Model\Settings	Seen Meta-Goal		Unseen Meta-Goal	
	lev^2	$xTed$	lev^2	$xTed$
Seen Dataset				
ChatGPT	0.87	0.87	0.64	0.62
ChatGPT + Pd	0.89	0.89	0.72	0.7
GPT-4	**0.97**	**0.97**	0.71	0.71
GPT-4 + Pd	0.97	0.96	**0.77**	**0.75**
Unseen Dataset				
ChatGPT	0.79	0.79	0.65	0.65
ChatGPT + Pd	0.86	0.84	0.72	0.69
GPT-4	**0.95**	**0.95**	0.71	0.7
GPT-4 + Pd	0.94	0.93	**0.73**	**0.72**

Table 2: Specification Derivation (NL-to-LDX) Results

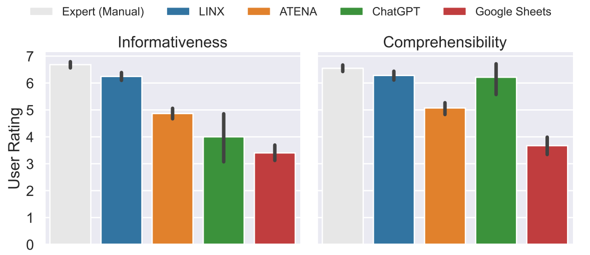


Figure 7: Informativeness & Comprehensibility Rating

analysis goals and LDX specifications from our benchmark dataset. We used g_1-$g-8$, as depicted in Table 1, and four additional pairs (deferred to [56] for space constraints), to obtain a total of four different goals for each of our three datasets.

We used LINX to generate an exploration notebook for each goal and dataset, and presented the output session in a Jupyter notebook (see Figure 1e for a snippet, and the full notebooks in [56]). We evaluated LINX compared to the the following baselines: **(1) ATENA [6].** We ran ATENA on each of the datasets. As it automatically generates an exploration session but does not accommodate user specifications, it produces the same exploration notebook for all four tasks of each dataset. **(2) ChatGPT (gpt-3.5-turbo) [51].** In this baseline we generated notebooks by asking the LLM to directly build an entire exploration notebook, containing real Pandas code, for a given description of the dataset and an analytical task. We executed the code provided by the LLM and presented the results in a Jupyter notebook. **(3) Google Sheets Explorer [23].** A commercial ML-based exploration tool that accommodates limited user specifications, allowing to specify columns and data subsets of interest. The specifications were composed w.r.t. to the LDX queries for each goal. For example, for goal $g5$ ("characteristics of summer flights"), we selected the columns 'month', 'airline', 'delay-reason' and 'scheduled arrival'/'departure', and the data subset containing flights from July and August. **(4) Human Expert.** Last, we used exploration notebooks generated manually by experts data scientists, to provide an "upper bound" for the output quality of the automatic approaches. We asked three experienced data scientists to manually compose a notebook (*without* any assistive tool) of interesting query operations that are relevant for the given goal. The instructions, data, and output of all baselines are provided in our code repository [56].

Subjective Study (User Rating). In this study, the participants were asked to review notebooks, generated by either LINX or the baselines, w.r.t. each notebook's corresponding analytical task. Each participant reviewed one notebook per dataset to neutralize the effect of experience. We then asked the participants to rate each notebook on a scale from 1 (lowest) to 7 (highest) according to the following criteria: (1) *Relevance - To what degree is the exploration notebook relevant for the given analysis goal?* (2) *Informativeness — To what extent does the notebook provide useful information about the data?* (3) *Comprehensibility - To what degree is the notebook comprehensible and easy to follow?*

Figure 5 presents the *relevance score* of LINX and the baselines for each of the three datasets. The results are averaged across all participants and goals for each dataset (The vertical line depicts the .95 confidence interval). As expected, manually composed notebooks from human experts obtained the highest rating - 6.71, 6.92, 6.53 for the Netflix, Flights, and Play Store datasets (resp). However, see that LINX obtains very close scores – 6.32, 6.39, 6.30 (respectively), surpassing ChatGPT, ATENA, and Google Sheets, which all have ratings below 4.

Next, we inspected the *informativeness* and *comprehensiveness* scores. Figure 7 depicts the average scores, over all three datasets. (The black vertical lines represent the .95 confidence interval.)

The human-expert notebooks once again achieve the highest scores. Additionally, both ATENA and Google Sheets now attain higher scores: ATENA scores 4.86 and 5.07, while Google Sheets follows with 3.40 and 3.67 for informativeness and comprehensiveness, respectively. ChatGPT achieves a high comprehensiveness score of 6.21, primarily due to its utilization of very simple analytical operations and straightforward code documentation. However, it falls behind in terms of informativeness, scoring an average of 4/7 (we further explore this gap in our analysis of the objective user study, below).

User Insight (for goal g_i)
"The ratio of movies-series in India is higher than the movies-series ratio anywhere else." (g_1)
"Most multi-season US TV shows are dramas or comedies" (g_2)
"About one-third of flights occur in summer, yet the monthly rate of delays remains consistent throughout the year." (g_5)
"While long flights are not delayed often, if they are, this is mainly for a security reason." (g_6)
"Apps with 1M installs are typically free, highly rated, and compatible with Android 4." (g_8)

Table 3: Examples of Insights Derived by Users Using LINX

LINX outperforms ATENA, ChatGPT, and Google Sheets (scoring 6.24 and 6.28 for informativeness and coherency), demonstrating it maintains both informativeness and comprehensibility in generating *goal-oriented* exploratory sessions.

Finally, we examine whether the ratings of LINX vary across different analytical goals. As shown in Figures 5 and 7, the small vertical error bars for LINX represent lower variance per dataset. The standard deviations, with respect to the goal g_i (in each notebook), are ±0.28 for relevance, ±0.43 for informativeness, and ±0.48 for comprehensibility. These deviations are second only to the human expert baseline, which demonstrates slightly lower variation. These results indicate that LINX's quality scores are relatively stable across different analytical goals and datasets.

Objective Study (Task Completion Success). We also compared LINX with the baselines in an objective manner – by asking users to examine notebook and then extract a list of insights that are *relevant* w.r.t. the given analytical goal. The correctness and relevance of insights were evaluated, by the same experts who constructed the manual *human-expert* notebook (Baseline 4), and is therefore highly familiar with the datasets and respective goals.

Figure 6 shows the average number of goal-relevant insights derived using each baseline. Using LINX, users derived an average of 2.7 relevant insights per goal, which is second only to the human-expert notebooks (3.2 insights). ATENA and Google Sheets are again far behind with an average of 0.8 and 0.4 relevant insights per goal (resp). Interestingly, ChatGPT obtains the lowest score of 0.3 insights. To better understand the unexpectedly low scores of ChatGPT notebooks, we further analyzed the feedback provided by users alongside their descriptions of the mined insights. The LLM-generated notebooks were perceived as well-documented and easy to follow; however, they primarily consisted of a collection of generic, descriptive statistics rather than the more "investigative" sessions produced by LINX. Users also noted negatively that the LLM tended to focus on arbitrary attributes or patterns.

Last, to further examine the quality of the insights derived from LINX-generated notebooks, we provide example insights derived by the participants, depicted in Table 3. See that users were able to extract compound, non-trivial insights that are indeed relevant to the corresponding analytical goals.

Summary. An extensive user study with 30 participants shows that users not only rate the exploration notebooks generated by LINX as highly relevant, informative, and comprehensible, but were also able to derive significantly more relevant insights compared to the non-human baselines.

7.4 CDRL Performance & Ablation Study

Last, we examine the performance of our CDRL Engine, by conducting first an ablation study to examine the necessity of each component, then a convergence speed comparison with the goal-invariant ATENA [6] ADE system.

Ablation Study. To gauge the necessity in the components of LINX we compared it to the following system versions, each missing one or more components: **(1) Binary Reward Only** uses a binary end-of-session reward, based solely on the output of the LDX verification engine, without using our full reward scheme (§5.2) and specification-aware network (§5.3). Instead, it uses the basic neural network of [6]. **(2) Binary+Imm. Reward** uses the reward scheme, as described in Section 5.2, without the immediate reward and the specification-aware network. **(3) W/O Spec.Aware NN** uses the full reward scheme (including the immediate reward), but with the basic neural network of [6].

To gauge the components necessity, We employed each baseline on the same 12 LDX queries used in the user study, and examined in how many cases each baseline is able to generate a compliant session, in up to 10M training steps. The results are depicted in Table 4, reporting the baselines' success in: (1) *structural* compliance, where a generated notebook complies with the structural specifications but not the operational ones, and (2) *full compliance*, where all specifications are met.

First, see that *Binary Reward Only*, which only receives the binary, end-of-session reward, fails to generate compliant sessions for any of the queries. As mentioned above, this is expected due to the sparsity of the reward and the vast size of the action space. *Binary+Imm. Reward*, which uses the more flexible compliance reward at the end of each session. obtains better results – fully complying with 3 queries, and structure-compliant with 7 additional ones. Next, *W/O Spec.Aware NN* obtains a significant improvement – it is able to comply with the structural specifications of all 12 LDX queries. However, it was *fully* compliant only for 5/12 queries.

Finally, see that only the full version of LINX-CDRL, which uses both the full reward-scheme *and* the specification-aware neural network, is able to generate compliant sessions for 100% of the LDX queries. This shows that our adaptive network design, as described in Section 5.3, is particularly useful in encouraging the agent to perform specification-compliant operations – despite the inherently large size of the action-space.

Convergence Speed. Lastly, we examine the convergence speed of our CDRL engine and compare it with the DRL engine of ATENA [6] in order to examine whether its more complex structure causes slower convergence, i.e., more steps are required to reach the output with the maximal reward. Figure 8 shows the convergence plots for the 12 LDX queries. The convergence for each LDX query i (corresponding to goal g_i) is depicted using a line labeled *'LINX #i'*, with the black line in each figure representing the convergence process of ATENA [6], serving as a baseline. (Recall that ATENA can only produce one generic exploration session per dataset.) Since the maximal reward varies depending on the LDX query and dataset, we normalize the rewards so that the maximum is 100%. Observe that the convergence processes of both ATENA and LINX-CDRL are roughly similar. Notably, LINX-CDRL sometimes converges even

LINX Version	Structure Compliance	Full Compliance
Binary Reward Only	0/12 (0%)	0/12 (0%)
Binary+Imm. Reward	10/12 (84%)	3/12 (25%)
W/O Spec. Aware NN	12/12 (100%)	5/12 (42%)
LINX-CDRL (Full)	**12/12 (100%)**	**12/12 (100%)**

Table 4: Ablation Study Results

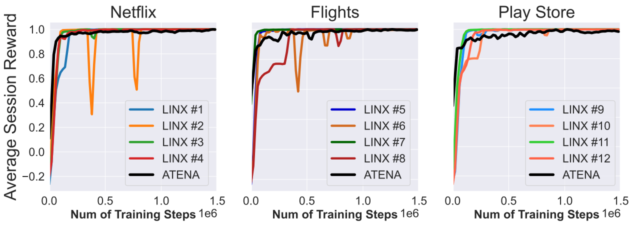


Figure 8: Convergence Comparison to ATENA

faster than ATENA (e.g., for the Play Store dataset), where ATENA takes 0.85M steps and LINX only 0.4M steps on average. In general, the average convergence to 100% reward is 0.36M steps, which takes about 20 minutes on our simple CPU hardware. While this could be significantly improved with GPU hardware, it is important to note that such running times are acceptable as LINX is *not* intended for interactive analysis (See again our discussion in Section 3).

Summary. Our performance study demonstrates two key findings: (1) all elements of LINX-CDRL, are required for consistently generating compliant notebooks. (2) Despite its more complex reward system and neural network, the convergence performance of LINX is on par with ATENA.

8 CONCLUSION & FUTURE WORK

This paper introduces LINX, a generative system for automated, goal-oriented exploration. It uses an LLM-based solution to derive exploratory specifications from the goal, and a modular ADE engine to create personalized exploratory sessions based on the derived specifications. In future work, we will explore ways in which LLMs can further enhance the analytical process. A promising direction is to utilize LLMs for augmenting LINX notebooks with natural language summaries and explanations, as well as auto-visualization and insights-mining solutions such as [39, 44, 71, 82]. Another direction is the adaptation of LINX to interactive analysis and data manipulation code generation. In addition, we will examine more analytical goals and corresponding exploration specifications, in order to extend and improve our benchmark dataset.

REFERENCES

[1] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. 2019. A comparative survey of recent natural language interfaces for databases. *The VLDB Journal* 28 (2019), 793 – 819. https://api.semanticscholar.org/CorpusID:195316636

[2] Alfred V Aho. 1991. Algorithms for finding patterns in strings, Handbook of theoretical computer science (vol. A): algorithms and complexity.

[3] Sara Alspaugh, Nava Zokaei, Andrea Liu, Cindy Jin, and Marti A Hearst. 2018. Futzing and moseying: Interviews with professional data analysts on exploration practices. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 22–31.

[4] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *ArXiv* abs/2304.09433 (2023). https://api.semanticscholar.org/CorpusID:258212828

[5] Ori Bar El, Tova Milo, and Amit Somech. 2019. Atena: An autonomous system for data exploration based on deep reinforcement learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2873–2876.

[6] Ori Bar El, Tova Milo, and Amit Somech. 2020. Automatically generating data exploration sessions using deep reinforcement learning. In *SIGMOD*.

[7] Microsoft Power BI. 2024. https://www.microsoft.com/en-us/power-platform/products/power-bi.

[8] Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 4560–4565. https://doi.org/10.18653/v1/P19-1448

[9] Shu Cai and Kevin Knight. 2013. Smatch: an Evaluation Metric for Semantic Feature Structures. In *Annual Meeting of the Association for Computational Linguistics*. https://api.semanticscholar.org/CorpusID:11345321

[10] Alexandre Chanson, Ben Crulis, Nicolas Labroche, Patrick Marcel, Verónika Peralta, Stefano Rizzi, and Panos Vassiliadis. 2020. The traveling analyst problem: definition and preliminary study. In *Design, Optimization, Languages and Analytical Processing of Big Data*.

[11] Alexandre Chanson, Nicolas Labroche, Patrick Marcel, Stefano Rizzi, and Vincent t'Kindt. 2022. Automatic generation of comparison notebooks for interactive data exploration.. In *EDBT*. 2–274.

[12] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2022. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *ArXiv* abs/2211.12588 (2022). https://api.semanticscholar.org/CorpusID:253801709

[13] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. 2018. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757* (2018).

[14] Nachum Dershowitz and Shmuel Zaks. 1980. Enumerations of ordered trees. *Discret. Math.* 31, 1 (1980), 9–28.

[15] Daniel Deutch, Amir Gilad, Tova Milo, and Amit Somech. 2020. ExplainED: explanations for EDA notebooks. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2917–2920.

[16] Luciano Di Palma, Yanlei Diao, and Anna Liu. 2019. A Factorized Version Space Algorithm for" Human-In-the-Loop" Data Exploration. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1018–1023.

[17] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. 2016. AIDE: An Active Learning-based Approach for Interactive Data Exploration. *TKDE* (2016).

[18] Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. 2019. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In *SIGMOD*.

[19] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and M. Oyamada. 2022. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. *ArXiv* abs/2212.07588 (2022). https://api.semanticscholar.org/CorpusID:254685724

[20] Marina Drosou and Evaggelia Pitoura. 2013. YmalDB: exploring relational databases via result-driven recommendations. *VLDBJ* 22, 6 (2013).

[21] Magdalini Eirinaki, Suju Abraham, Neoklis Polyzotis, and Naushin Shaikh. 2014. Querie: Collaborative database exploration. *TKDE* (2014).

[22] Tregex engine implementation. 2022. https://github.com/yandex/dep_tregex.

[23] Google Sheets Explore. 2022. https://www.blog.google/products/g-suite/visualize-data-instantly-machine-learning-google-sheets/.

[24] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving Text-to-SQL Evaluation Methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Victoria, Australia, 351–360. https://doi.org/10.18653/v1/P18-1033

[25] Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. 2019. Chainerrl: A deep reinforcement learning library. *arXiv preprint arXiv:1912.03905* (2019).

[26] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. PAL: Program-aided Language Models. *ArXiv* abs/2211.10435 (2022). https://api.semanticscholar.org/CorpusID:253708270

[27] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16, 1 (2015), 1437–1480.

[28] Liqiang Geng and Howard J Hamilton. 2006. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)* 38, 3 (2006), 9–es.

[29] John D Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in science & engineering* 9, 03 (2007), 90–95.

[30] ATENA Basic Implementation. 2024. https://github.com/TAU-DB/ATENA-A-EDA/tree/master/atena-basic.

[31] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. 2014. Smart Drill-Down. *Target* 6000 (2014), 0.

[32] Flights Dataset (Kaggle). 2023. https://www.kaggle.com/usdot/flight-delays.

[33] Google Play Store Dataset (Kaggle). 2023. https://www.kaggle.com/lava18/google-play-store-apps.

[34] Netflix Dataset (Kaggle). 2023. https://www.kaggle.com/shivamb/netflix-shows.

[35] Pavan Kapanipathi, I. Abdelaziz, Srinivas Ravishankar, Salim Roukos, Alexander G. Gray, Ramón Fernández Astudillo, Maria Chang, Cristina Cornelio, Saswati Dana, Achille Fokoue, Dinesh Garg, A. Gliozzo, Sairam Gurajada, Hima P. Karanam, Naweed Khan, Dinesh Khandelwal, Young suk Lee, Yunyao Li, Francois P. S. Luus, Ndivhuwo Makondo, Nandana Mihindukulasooriya, Tahira Naseem, Sumit Neelam, Lucian Popa, Revanth Reddy Gangi Reddy, Ryan Riegel, Gaetano Rossiello, Udit Sharma, G. P. Shrivatsa Bhargav, and Mo Yu. 2020. Leveraging Abstract Meaning Representation for Knowledge Base Question Answering. In *Findings*. https://api.semanticscholar.org/CorpusID:235303644

[36] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E John, and Brad A Myers. 2018. The story in the notebook: Exploratory data science using a literate programming tool. In *CHI*.

[37] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *Proc. VLDB Endow.* 13 (2020), 1737–1750. https://api.semanticscholar.org/CorpusID:220528413

[38] Tim Kraska. 2018. Northstar: An interactive data science system. *PVLDB* 11, 12 (2018).

[39] Doris Jung-Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A Hearst, et al. 2021. Lux: always-on visualization recommendations for exploratory dataframe workflows. *PVLDB* 15, 3 (2021), 727–738.

[40] Roger Levy and Galen Andrew. 2006. Tregex and Tsurgeon: Tools for querying and manipulating tree data structures.. In *LREC*. Citeseer, 2231–2234.

[41] Chenjie Li, Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. 2021. Putting things into context: Rich explanations for query answers using join graphs. In *Proceedings of the 2021 International Conference on Management of Data*. 1051–1063.

[42] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. arXiv:2305.03111 [cs.CL]

[43] Tavor Lipman, Tova Milo, and Amit Somech. 2023. ATENA-PRO: Generating Personalized Exploration Notebooks with Constrained Reinforcement Learning. In *Companion of the 2023 International Conference on Management of Data*. 167–170.

[44] Pingchuan Ma, Rui Ding, Shi Han, and Dongmei Zhang. 2021. Metainsight: Automatic discovery of structured knowledge for exploratory data analysis. In *Proceedings of the 2021 International Conference on Management of Data*. 1262–1274.

[45] Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. Language Models of Code are Few-Shot Commonsense Learners. *ArXiv* abs/2210.07128 (2022). https://api.semanticscholar.org/CorpusID:252873120

[46] Maja J Mataric. 1994. Reward functions for accelerated learning. In *Machine learning proceedings 1994*. Elsevier, 181–189.

[47] Tova Milo and Amit Somech. 2018. Next-Step Suggestions for Modern Interactive Data Analysis Platforms. In *KDD*.

[48] Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing Few-shot Text-to-SQL Capabilities of Large Language Models: A Study on Prompt Design Strategies. arXiv:2305.12586 [cs.CL]

[49] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher R'e. 2022. Can Foundation Models Wrangle Your Data? *Proc. VLDB Endow.* 16 (2022), 738–746. https://api.semanticscholar.org/CorpusID:248965029

[50] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[51] GPT 3.5 (OpenAI). 2023. https://platform.openai.com/docs/models/gpt-3-5.

[52] Jeffrey M Perkel. 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature* 563, 7732 (2018), 145–147.

[53] Aurélien Personnaz, Sihem Amer-Yahia, Laure Berti-Equille, Maximilian Fabricius, and Srividya Subramanian. 2021. Balancing Familiarity and Curiosity in Data Exploration with Deep Reinforcement Learning. In *Fourth Workshop in Exploiting AI Techniques for Data Management*. 16–23.

[54] Aurélien Personnaz, Sihem Amer-Yahia, Laure Berti-Equille, Maximilian Fabricius, and Srividya Subramanian. 2021. DORA THE EXPLORER: Exploring Very Large Data With Interactive Deep Reinforcement Learning. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4769–4773.

[55] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. arXiv:2304.11015 [cs.CL]

[56] LINX Github Repository. 2022. https://github.com/analysis-bots/LINX.

[57] Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, Online, 311–324. https://doi.org/10.18653/v1/2021.naacl-main.29

[58] Adam Rule, Aurélien Tabard, and James D Hollan. 2018. Exploration and explanation in computational notebooks. In *CHI*.

[59] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. 2023. Querying Large Language Models with SQL. *ArXiv* abs/2304.00472 (2023). https://api.semanticscholar.org/CorpusID:257913347

[60] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. 1998. Discovery-driven exploration of OLAP data cubes. In *EDBT*.

[61] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.

[62] William Saunders, Girish Sastry, Andreas Stuhlmueller, and Owain Evans. 2017. Trial without error: Towards safe reinforcement learning via human intervention. *arXiv preprint arXiv:1707.05173* (2017).

[63] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 9895–9901. https://doi.org/10.18653/v1/2021.emnlp-main.779

[64] Edward Segel and Jeffrey Heer. 2010. Narrative visualization: Telling stories with data. *IEEE transactions on visualization and computer graphics* 16, 6 (2010), 1139–1148.

[65] Danqing Shi, Xinyue Xu, Fuling Sun, Yang Shi, and Nan Cao. 2020. Calliope: Automatic visual data story generation from a spreadsheet. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 453–463.

[66] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. 2016. Effortless Data Exploration with Zenvisage: An Expressive and Interactive Visual Analytics System. *Proc. VLDB Endow.* 10, 4 (nov 2016), 457–468.

[67] Matthew Skala. 2014. A structural query system for Han characters. *arXiv preprint arXiv:1404.5585* (2014).

[68] Tableau Software. 2024. https://www.tableau.com/.

[69] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*. 1493–1503.

[70] Mirac Suzgun, Nathan Scales, Nathanael Scharli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed Huai hsin Chi, Denny Zhou, and Jason Wei. 2022. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them. In *Annual Meeting of the Association for Computational Linguistics*. https://api.semanticscholar.org/CorpusID:252917648

[71] Bo Tang, Shi Han, Man Lung Yiu, Rui Ding, and Dongmei Zhang. 2017. Extracting top-k insights from multi-dimensional data. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1509–1524.

[72] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Y. Halevy. 2021. From Natural Language Processing to Neural Databases. *Proc. VLDB Endow.* 14 (2021), 1033–1039. https://api.semanticscholar.org/CorpusID:232328446

[73] Immanuel Trummer. 2023. Demonstrating GPT-DB: Generating Query-Specific and Customizable Code for SQL Processing with GPT-4. *Proc. VLDB Endow.* 16 (2023), 4098–4101. https://api.semanticscholar.org/CorpusID:261382153

[74] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. 2015. SeeDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB* 8, 13 (2015).

[75] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677

[76] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Huai hsin Chi, and Denny Zhou. 2022. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *ArXiv* abs/2203.11171 (2022). https://api.semanticscholar.org/CorpusID:247595263

[77] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Huai hsin Chi, F. Xia, Quoc Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *ArXiv* abs/2201.11903 (2022). https://api.semanticscholar.org/CorpusID:246411621

[78] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *SciPy*, Stéfan van der Walt and Jarrod Millman (Eds.).

[79] Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break It Down: A Question Understanding Benchmark. *Transactions of the Association for Computational Linguistics* 8 (2020), 183–198. https://doi.org/10.1162/tacl_a_00309

[80] Kanit Wongsuphasawat, Yang Liu, and Jeffrey Heer. 2019. Goals, Process, and Challenges of Exploratory Data Analysis: An Interview Study. *arXiv preprint arXiv:1911.00568* (2019).

[81] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2016. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *TVCG* (2016).

[82] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 chi conference on human factors in computing systems*. 2648–2659.

[83] Navid Yaghmazadeh, Yuepeng Wang, Işıl Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages* 1 (2017), 1 – 26. https://api.semanticscholar.org/CorpusID:8210357

[84] Cong Yan and Yeye He. 2020. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1539–1554.

[85] Ori Yoran, Tomer Wolfson, Ben Bogin, Uri Katz, Daniel Deutch, and Jonathan Berant. 2023. Answering Questions by Meta-Reasoning over Multiple Chains of Thought. *ArXiv* abs/2304.13007 (2023). https://api.semanticscholar.org/CorpusID:258309779

[86] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 3911–3921.

[87] Tao Yu, Rui Zhang, Kai-Chou Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Z Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. *ArXiv* abs/1809.08887 (2018). https://api.semanticscholar.org/CorpusID:52815560

[88] Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing* 18, 6 (1989), 1245–1262.

[89] Li Zhang, Liam Dugan, Hai Xu, and Chris Callison-Burch. 2023. Exploring the Curious Case of Code Prompts. *ArXiv* abs/2304.13250 (2023). https://api.semanticscholar.org/CorpusID:258332119

[90] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *ArXiv* abs/1709.00103 (2017). https://api.semanticscholar.org/CorpusID:25156106

[91] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. arXiv:2205.10625 [cs.AI]

APPENDIX

A ADDITIONAL MATERIAL & DISCUSSIONS

A.1 LDX Comparison to Tregex

As mentioned above, LDX is based on Tregex [40], a popular tree query language among the NLP community. Tregex is typically used to query syntax trees, grammars and annotated sentences. LDX adopts similar syntax for specifying the exploratory tree structure and labels (exploratory operations), and extends it to the context of data exploration using the *continuity variables*, which semantically connect the desired operations as described above.

A.2 LDX Verification Engine Computation Times Discussion

As is the case for Tregex [67], evaluating an LDX query may require, in the worst case, iterating over all possible node assignments, of size $\frac{|T_D|!}{(|T_D|-|Nodes(Q_X)|)!}$. However, we show in Section 7.4 that in practice, computing the LDX-compliance reward scheme takes a negligible amount of time, compared to the overall session generation time. This is mainly because both the exploration tree T_D and the query Q_X are rather small[2].

A.3 LDX-Compliance Reward Scheme

The generic exploration reward, as described above, encourages the agent to employ interesting, useful analytical operations on the given dataset. However, the session is inadequate if it is interesting yet irrelevant to the goal at hand. Our goal is therefore to enforce that the agent also produces a sequence of operations that is fully compliant with the LDX specifications derived from the analytical goal description.

Given the input dataset and Q_X specifications, our compliance reward scheme gradually "teaches" the agent to converge to a Q_X-compliant exploratory session. This is based on the observation that *structural specifications should be learned first.* Namely, if the agent had learned to generate *correct* operations in an *incorrect* order/structure – then the learning process is largely futile (because the agent had learned to perform an incorrect exploration path, and now needs to relearn, from scratch, to employ the desired operations in the correct order). LINX therefore encourages the agent to first generate exploratory sessions with the correct structure, using a high penalty for non-compliant sessions. Once the agent has learned the correct structure, it will now obtain a gradual reward that encourages it to satisfy the operational specifications.

Finally, when the agent manages to generate a fully compliant session it obtains a high positive reward, and can now further increase it by optimizing on the exploration reward signal (as described above).

We next describe our reward scheme, comprising both an *end-of-session* and an *immediate* reward signals.

End-of-Session Compliance Reward. Our End-of-Session reward scheme is depicted in Algorithm 2. Given a session T_D performed by the agent and the set of specifications Q_X (from the LDX query Q_X), we first check (Line 1) whether the session T_D is compliant with *LDX* using Algorithm 1. In case T_D is compliant, a high positive reward is given to the agent (Line 2).

Algorithm 2: End-of-Session Conditional Reward

```
    Input: Exploration Tree $T_D$, LDX Specifications $Q_X$
    Output: Reward $R$
1   if VerifyLDX($Q_X, T_D$) = True then
        // $T_D$ is compliant with $Q_X$
2       return POS_REWARD
3   $S_{struct} \leftarrow struct(Q_X)$ // Structural specs of $Q_X$
4   $\Phi_V \leftarrow GetTregexNodeAssg(S_{struct}, T_D)$
5   if $\Phi_V = \emptyset$ then
        // $T_D$ violates Structural specs
6       return NEG_REWARD
    // Calculate operational-based reward:
7   $S_{opr} \leftarrow opr(Q_X)$ // Operational specs of $Q_X$
8   return $\max_{\phi_V \in \Phi_V}$ GetOprReward($\phi_V, S_{opr}$)

    GetOprReward ($\phi_V, S_{opr}$)
9       $reward \leftarrow 0$
10      for $s \in S_{opr}$ do
11          $v_s = \phi_V(Node(s))$
            $reward$ += $\frac{\text{\# of matching opr. params in } v_s}{\text{\# of specified params in } s}$
12      return $reward$
```

Next, in case T_D is not compliant with Q_X, we now check whether it is compliant with $struct(Q_X)$, the *structural specifications* in Q_X (denote S_{struct} here for brevity) This is done by calling the Tregex engine (since there is no need to verify the continuity variables), which returns all valid assignments Φ_V for S_{struct} over T_D (Line 3). If there are no valid assignments, it means that T_D is non-compliant with the specified structure, therefore a fixed negative penalty is returned (Line 6).

In case T_D satisfies the structural specifications (but not the operational/continuity) – we wish to provide a non-negative reward that is proportional to the number of satisfied *operational* specifications (and parts thereof). Namely, the more specified operational parameters (e.g., attribute name, aggregation function, etc.) are satisfied – the higher the reward. To do so, we compute the operational reward for each node assignment $\phi_v \in \Phi_V$, returning the maximal one. The operational reward is calculated in *GetOprReward)* (Lines 9-12). We initialize the reward with 0, then iterate over each operational specification $s \in S_{opr}$ (Line 9), and first, retrieve its assigned node v_s (according to ϕ_V). We then compute the ratio of matched individual operational parameters in v_s, out of all operational parameters specified in s (Line 11). This ratio is accumulated for all specifications in S_{opr}, s.t. the higher the number of matched operational parameters, the higher the reward.

Immediate (per-operation) Compliance Reward. To further encourage the agent to comply with the structural constraints, we develop an additional, *immediate* reward signal, granted after each operation. The goal of the immediate, per-operation reward is to detect, in real-time, an operation performed by the DRL agent that violates the structural specifications.

The procedure is intuitively similar to the LDX verification routine (Algorithm 1), yet rather than taking a full session as input it takes an *ongoing* session T_D^i, i.e., after i steps, and the remaining

[2] For example, the mean session size in the exploratory sessions collection of [47] is 8.

number of steps $N - i$. It then assesses whether there is a *future* assignment, in up to n more steps, that can satisfy the structural constraints $struct(Q_X)$. This is simply done by calling the Tregex match function $GetTregexNodeAssg(struct(Q_X), T_D^\star)$, with each possible *tree completion* (denoted $T_D^\star$) for the ongoing exploration tree. The completion of the ongoing exploration tree T_D^i simply extends it with $N - i$ additional "blank" nodes. Starting from the current node v_i, blank nodes can be added only in a manner that respects the order of query operations execution in the session (captured by the nodes' pre-order traversal order [47]). Namely, each added node v_j, s.t. $i < j \leq N$ can be added as an immediate child of v_{j-1} or one of its ancestors). We explain below that the number of possible tree completions throughout a session of size N (including the root) is bounded by C_N, where C_N is the Catalan number.

In more details, the immediate reward procedure at step i of an ongoing session, has $N - i$ remaining nodes to complete a full possible session tree. In order to bound the number of possible trees at each iteration, we will examine the iteration with the largest number of completions, which is right after the first step of the agent, namely when $i = 1$ and T_i is a tree with a root and one child v_1 which is the current node. In this scenario, after adding an additional node v_2, we got two possible trees: one where v_2 is a child of v_1, and another one where v_2 is the right sibling of v_1. When adding one more node, v_3, we have total of 5 possible trees: If v_2 is child of v_1, then v_3 can be a child of v_2, right sibling of v_2 or right sibling of v_1. Otherwise it can be a child of v_2 or right sibling of v_2. The number of possible trees continue to grow in each iteration. Even though the procedure is iterative, each possible final tree of size N can only be generated once, due to the pre-order traversal manner. Thus, the number of possible trees is bounded by the number of ordered trees of size N, which is bounded by $C_N = \frac{1}{n+1}\binom{2n}{n}$, where C_N is the Catalan number [14]. To further improve the procedure performance, we employ the immediate reward only after $i \geq 3$ steps. This way still encourage the agent to comply with the structural constraints, but avoid the large number of tree-completions in the first steps. For example, when $10 \leq N \leq 20$, the number of completions is reduced by 4-5X.

A.4 Specification-Aware Network - An Example

Recall that Figure 2 depicts our specification-aware neural network architecture, deriving its structure from the LDX specifications Q_X. We next give an example for an action selection process in accordance with our example workflow depicted in Figure 1.

Figure 2 displays an example state in the network, in which the action probabilities are already computed (colored in light orange). First, the agent samples an operation type using the multi-softmax segment σ_{ops}. As in Figure 2, 'Snippet' obtained the highest probability of 0.5. If indeed selected, the specific snippet is chosen using Segment σ_{snp}. See that the Filter snippet `'F, Country, eq, *'` obtains the highest probability (0.6). Now, as the set of free parameters only contains the filter 'term' parameter, the agent now chooses a term using Segment σ_3. The chosen term is 'India'.

A.5 Templated SQL-based Prompt Baseline (NL2SQL2LDX)

We examined the performance of an SQL transformation instead of Python (Pandas). To achieve this, we implemented an NL2SQL2LDX chained prompt solution, where we replaced the intermediate transformation with SQL. The first prompt generates templated SQL based on the NL description of the analysis goal, and the second prompt translates the templated SQL into LDX (see our code repository [56] for the full prompts).

We then compared the performance of NL2SQL2LDX to our Python (Pandas)-based solution, NL2PD2LDX. Table 5 shows the scores for 50 random samples in the "Unseen dataset" scenario for both ChatGPT and GPT-4. The NL2SQL2LDX solution is labeled "+ SQL," and the NL2PD2LDX is labeled "+ PD" . As shown, the results of NL2SQL2LDX are inferior to those of NL2PD2LDX. Analyzing these results, we observed that the main issue causing the lower performance is the suboptimal generation of templated SQL code by the LLMs, particularly in supporting the *continuity variables* (recall from Section 4.1).

Model\Settings	Seen Meta-Goal		Unseen Meta-Goal	
	lev2	xTed	lev2	xTed
Unseen Dataset				
ChatGPT	0.81	0.82	0.69	0.65
ChatGPT + PD	0.86	0.85	0.73	0.66
ChatGPT + SQL	0.78	0.72	0.61	0.5
GPT-4	0.93	0.94	0.73	0.68
GPT-4 + PD	0.92	0.93	0.76	0.71
GPT-4 + SQL	0.88	0.9	0.71	0.67

Table 5: Comparison of intermediate code representation methods: SQL vs Python (Pandas).

A.6 Metrics Flexibility to Multiple Ground Truths

In some cases, more than one LDX query may fit the same goal. For example, given the analysis goal *"Display two characteristics of apps with size at least 1000 kilobytes, when compared to all apps"* (Play Store dataset) the ground truth LDX query is:

```
ROOT CHILDREN <A1,A2,A3>
    A1 LIKE [G,(?<COL1>.*),(?<AGG_FUNC1>.*),(?<AGG_COL1>.*)]
    A2 LIKE [G,(?<COL2>.*),(?<AGG_FUNC2>.*),(?<AGG_COL2>.*)]
    A3 LIKE [F,app_size_kb,ge,1000] CHILDREN <B1,B2>
        B1 LIKE [G,(?<COL1>.*),(?<AGG_FUNC1>.*),(?<AGG_COL1>.*)]
        B2 LIKE [G,(?<COL2>.*),(?<AGG_FUNC2>.*),(?<AGG_COL2>.*)]
```

For this goal, the LLM-generated query using GPT-4 chained-prompt (GPT-4 + Pd) is:

```
ROOT CHILDREN <A1,A2,A3>
    A1 LIKE [F,app_size_kb,ge,1000] CHILDREN <B1,B2>
        B1 LIKE [G,(?<COL1>.*),(?<AGG_FUNC1>.*),(?<AGG_COL1>.*)]
        B2 LIKE [G,(?<COL2>.*),(?<AGG_FUNC2>.*),(?<AGG_COL2>.*)]
    A2 LIKE [G,(?<COL1>.*),(?<AGG_FUNC1>.*),(?<AGG_COL1>.*)]
    A3 LIKE [G,(?<COL2>.*),(?<AGG_FUNC2>.*),(?<AGG_COL2>.*)]
```

The queries are highly similar – only differ in the order of the queries (the filter query and its subsequent group-by operations appear first, rather than last).

To gauge the suitability of our metrics for this case, we manually analyzed 50 random pairs of ground-truth queries and LLM-generated queries, categorizing each as either (1) "Identical"—the exact same query (allowing for equivalent constructs, such as "Descendants" and "Children"); (2) "Equivalent"—where the generated LDX query perfectly matches the input goal but differs from the ground-truth query; or (3) "Different"—where the generated query does not adequately fulfill the input goal. We then compared the scores of queries in each category to see if there were significant differences.

Table 6 shows the average lev^2 and $xTED$ scores for each type of generated query. As expected, identical queries achieve scores close to 1. Notably, equivalent queries also receive relatively high scores—0.78 ($xTED$) and 0.75 (lev^2)—which are significantly higher than those of different queries, which score 0.62 and 0.66 for $xTED$ and lev^2, respectively. This experiment demonstrates that our metrics are flexible and effectively capture LDX queries that are equivalent, even if not identical.

Category	Count	$xTed$	lev^2
Identical	13	0.99	0.93
Equivalent	16	0.78	0.75
Different	21	0.62	0.66

Table 6: LDX Query Evaluation Breakdown (50 random cases)

B ADDITIONAL IMPLEMENTATION DETAILS

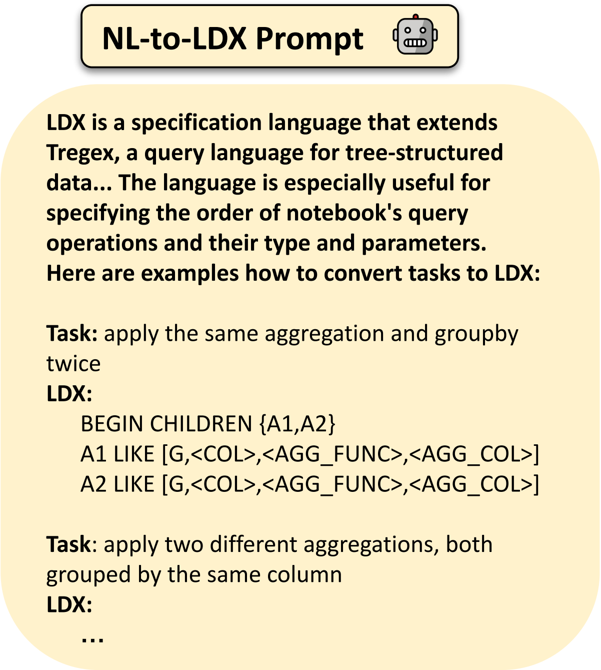
B.1 Prompts Description

NL-to-LDX. The NL-to-LDX prompt is structured the same as the NL-to-Pandas prompt as discussed in section §6: (1) NL-to-LDX goal description; (2) a series of few-shot examples; (3) the test analysis goal alongside a small dataset sample. Each few-shot example in (2) comprises several steps: (a) example goal description; (b) description of the example's corresponding dataset and schema; (c) the correct LDX query for the goal; (d) an NL explanation of the output. See Fig. 9.

Prompts Number of Examples. The NL-to-Pandas and Pandas-to-LDX prompts have 14 and 10 examples respectively, while the NL-to-LDX prompt has 14 examples. All prompts contain 8 examples which correspond to our 8 analytical meta-goals (Table 1). For NL-to-Pandas and NL-to-LDX we added 6 initial examples of mapping basic constructs before moving on to the 8 template examples, and for Pandas-to-LDX we added 2 additional general examples.

B.2 Evaluation Implementation Details

Minimal Tree. In section §7.2,it was previously mentioned that we construct a minimal tree for the compared LDX queries in order to apply them a tree distance metric. The conversion of LDX query to tree is generally straightforward, except for the 'DESCENDANTS' structural specification operator, since the distinction between DESCENDANTS and CHILDREN can't be expressed out-of-the-box. Our ad hoc approach for addressing that is setting the descendants as direct children in the converted tree (meaning the minimal specification-compliant tree) and adding 'children type' as an additional property of the action label. Additionally, we slightly modified the action distance function by penalizing variations in the 'children type' of the compared actions.



NL-to-LDX Prompt

LDX is a specification language that extends Tregex, a query language for tree-structured data... The language is especially useful for specifying the order of notebook's query operations and their type and parameters. Here are examples how to convert tasks to LDX:

Task: apply the same aggregation and groupby twice
LDX:

```
BEGIN CHILDREN {A1,A2}
A1 LIKE [G,<COL>,<AGG_FUNC>,<AGG_COL>]
A2 LIKE [G,<COL>,<AGG_FUNC>,<AGG_COL>]
```

Task: apply two different aggregations, both grouped by the same column
LDX:
...

Figure 9: Example of the prompt used for NL2LDX

Masking Continuity Variables. Furthermore, we mask the continuity variable names of the derived minimal trees to eliminate prevent scoring reduction due to naming differences. We do so by separately masking each category of continuity variables. For instance, continuity variables that define attributes of filter operations are substituted with identifiers such as att1, att2, att3, and so forth. Similarly, those that define aggregation function are substituted with identifiers such as aggfunc1, aggfunc2, aggfunc3, etc. This masking systematic approach aids in avoiding false penalty, while on the other hand ensuring scores are not inappropriately inflated.