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 full exploration sessions, containing illustrative queries that showcase interesting parts 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, flexible 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 comparative 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-invariant 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 [**?**]. Most prominently, query recommendation systems and simplified analysis interfaces like Tableau[1] and Power BI[2]. Recently, a new line of work called Automated Data Exploration (ADE), considers data exploration as a multi-step, AI *control problem* [5, 6, 47**? ?** , 48]. As opposed to the interactive process in which assistive tools help users one step at a time, ADE systems receive an input dataset, and auto-generate a full exploration session comprising multiple, interconnected queries. This output session is often displayed in a scientific notebook interface [46], allowing users to quickly gain substantial knowledge on the data before investigating it further.

However, 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 [29], which contains information about more than 8.8K different titles (See Figure **??**). 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 showcasing 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

[1] https://www.tableau.com/
[2] https://www.microsoft.com/en-us/power-platform/products/power-bi

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 tailored specifically to the dataset and the given goal at hand.

To create a personalized output session, LINX follows two steps: First, it interprets the input analysis goal and derives from it a set of specifications for the desired output exploration session. Second, the dataset along with the specifications are transferred to a novel, flexible ADE engine which can adapt its output accordingly.

Therefore, LINX requires two major components: A flexible ADE framework that can take into consideration custom specifications (as opposed to a fixed objective in existing ADE systems), and a way to generate these specifications from a natural language prompt.

Specification-aware ADE framework.

Since ADE systems need to search within the space of exploration sessions comprising multiple possible queries, they need to employ powerful optimization solutions.

Our flexible ADE engine is based on ATENA [6], a deep reinforcement learning (DRL) framework for goal-agnostic exploration, adapting it to the GDE use case. ATENA uses a predefined objective function that scores the quality of output sessions based on the interestingness of query results, their diversity, and the overall coherency of the entire session. Then, a DRL engine performs an optimization process, in which it learns to produce a maximal-scoring session by employing a multitude of intermediate sessions, then updating its internal policy according to their scores until converging to an optimal one.

Building a flexible ADE requires two significant components lacking in existing ADE systems. First, a means to articulate custom exploration specifications. Second, the ability to provide real-time feedback within the DRL optimization process which effectively takes into account the provided specification requirements. In order to solve this twofold challenge, we introduce LDX, a formal, intermediate language for data exploration coupled with a verification engine which is used to provide the necessary feedback guiding the optimization process.

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, LINX users do not need to compose LDX queries, but instead, these are derived directly from the analysis goal description, as explained below. The LDX verification engine is used in several variations within the optimization process.

A challenge arises because the exploration specifications complicates the ADE optimization problem, as the space of maximal-scoring *and* specifications compliant ones can be significantly smaller.

compliate the problem because now we need to find exploration sessions that are both maximize both the exploration score and are also compliant with the specificaitons. the general This is a known challenge known as Sparse Optimality [**?**] or Reward Sparsity [40].

(2) means to provide specifications-compliance feedback while overcoming the reward sparsity (sparse optimality) problem. Namely, adding more constraints to the desired output session increases the complexity of the search problem, since now a smaller portion of the high-utility DE sessions are compliant with the specifications.

This is a known challenge known as Sparse Optimality [**?**] or Reward Sparsity [40]. To tackle this challenge, we first use 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 further utilize a novel neural network architecture, inspired by prior work in *constrained deep reinforcement learning* (CDRL) [10, 56], where the neural network agent is specifically designed to handle additional requirements, such as safety constraints in autonomous driving frameworks [22]. In such systems, an external mechanism is used to override the agent's action if they are violating the constraints. In our case, rather then overriding actions externally, we encourage it to perform compliant operations: This is done via a novel *specification-aware neural network* that derives its final structure from the LDX specifications, thereby prompting the agent to "explore" the space of potentially-compliant operations with a higher probability.

LLM-Based solution for deriving exploration specifications given description of an analytical goal. When given a description of an analytical goal and a dataset, the primary challenges in generating exploration specifications are (1) determine which elements in the output session can be derived apriori, leaving the remainder to be discovered by the ADE engine, and (2) output a syntactically correct LDX query. 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 Section 2 for a discussion).

To overcome the absence of NL-LDX information in the LLM training data, we use a *few-shot* setting [42, 67], coupled with *intermediate code representation* [9, 39, 70]: Instead of directly instructing the LLM to generate LDX specifications, we adopt a two-stage *chained* prompt **[[[(Tomer) what do you mean by "chained prompt"? I am not familiar with the term.]]]**. In the first prompt, the LLM is tasked with expressing the exploration plan as a non-executable Python Pandas [68] code **[[[(Tomer) why not just use Pandas instead of LDX?]]]**. In the second stage, an additional prompt instructs the LLM to translate the resulting code into formal LDX specifications. **[[[(Tomer) we do not use a chained prompt. we use two instances of the same LLM, each prompted for a separate translation task: (1) NL-to-Pandas; (2) Pandas-to-LDX]]]**

[[[(Amit) OLD:::::::::::]]] In LINX, we overcome the challenge of generating goal-oriented exploratory notebooks by introducing a combined LLM-ADE solution. We first harness the LLM's "analytical mind" to a set of specifications for the desired exploratory process based on the user's goal description. The specifications derived from the analysis goal will be used to *guide* a data-driven search process, performed by a novel ADE system based on *Constrained* Deep Reinforcement Learning (CDRL). Our CDRL agent learns from repeatedly performing a multitude of exploratory sequences on the data, until it finally converges to an exploration process that meets the user's analytical goal.

This approach raises several challenges. First, how to express exploration specifications? Namely, what elements in the desired exploration process can and should be specified in advance, and how can we quickly determine if an output session is compliant? Second, how to design the exploration specification derivation as a suitable task for an LLM? Last, how can these apriori specifications be used to effectively *guide* the ADE agent in generating specifications-compliant, useful exploratory sessions? **[[[(Tomer) maybe list the challenges: (1); (2); (3), just to make it a bit easier to follow, when you get back to them in the following passages?]]]**

Solution Outline. We first design a structured specification language for data exploration processes, called LDX **[[[(Tomer) why not use Pandas? or Tableau?]]]**. LDX allows to effectively define the space of possible exploration sessions, and within it, the space of *desired, relevant* sessions w.r.t. the analytical goal. LDX is designed according to the observation that the output exploratory process **[[[(Tomer) we first used the term exploratory session, then sequence and now process. maybe we should stick to a single term for the ADE output throughout the paper, to minimize confusion?]]]** should form a *narrative* [31, 46], wherein the contextual connection and logical organization allow the observing user to fully comprehend the process and the insights it surfaces. To that end, LDX allows specifying desired fragments of analytical operations **[[[(Tomer) did not fully understand what do you mean by ”desired fragments of analytical operations”]]]**, shaping the process structure, and crucially, defining how to contextually link individual operations. LDX is based on Tregex [35], and uses regular-expression syntax to allow for *partial specification* of operations (to be fully instantiated by the ADE engine) **[[[(Tomer) why not use Tregex, what's novel about our extension?]]]**. Importantly, LDX is coupled with an efficient verification engine that is used to quickly determine if an output exploration process is indeed compliant with the given specifications. **[[[(Tomer) why is this language better than existing specification language for data exploration processes? maybe we should highlight its benefits compared to existing solutions, WDYT?]]]**

We then tackle the second challenge of deriving exploratory specifications (in LDX) using an LLM **[[[(Tomer) is this the challenge? I thought the challenge was to provide users with a natural language interface for ADE. using an LLM is not the challenge, the interface is, no?]]]**. This is a non-trivial task for a language model, given the novelty of the assignment and the absence of such knowledge in its training data. To overcome the absence of NL-LDX information in the LLM training data, we use a *few-shot* setting [42, 67], coupled with *intermediate code representation* [9, 39, 70]: Instead of directly instructing the LLM to generate LDX specifications, we adopt a two-stage *chained* prompt **[[[(Tomer) what do you mean by ”chained prompt”? I am not familiar with the term.]]]**. In the first prompt, the LLM is tasked with expressing the exploration plan as a non-executable Python Pandas [68] code **[[[(Tomer) why not just use Pandas instead of LDX?]]]**. In the second stage, an additional prompt instructs the LLM to translate the resulting code into formal LDX specifications. **[[[(Tomer) we do not use a chained prompt. we use two instances of the same LLM, each prompted for a separate translation task: (1) NL-to-Pandas; (2) Pandas-to-LDX]]]**

Last, to overcome the third challenge **[[[(Tomer) what is the third challenge? I had to scroll up and ctrl F the word 'challenges' to find it. maybe we can restate it here for clarity?]]]** we devise an ADE framework that can effectively support custom instructions. As the search space in ADE is already large (contains all possible sequences of analytical operations), general-purpose systems such as [6, 48] suggest using Deep Reinforcement Learning for the data-driven search of "optimal" (i.e., high-utility) exploration sequences. This is in order to maximize the utilization of previously executed exploration sequences performed by the agent. Unfortunately, introducing custom specifications for the desired output adds a significant burden on the agent, as the number of optimal output sessions decreases due to non-compliance.

Our solution draws inspiration from prior work in *constrained deep reinforcement learning* (CDRL) [10, 56], where the neural network agent is specifically designed to handle additional requirements, such as safety constraints [22] **[[[(Tomer) this sounds as if we do the same thing. maybe state how we extend or improve thes works which inspired us?]]]**. We adapt the general DRL-based ADE framework in [6] to effectively consider custom specifications by (1) a *specification-compliance* reward scheme that integrates multiple feedback signals, encouraging the agent to initially generate notebooks with a correct *structure* and then make further adjustments to operation parameters. (2) An adaptive, *specification-aware neural network* that derives its final structure from the LDX specifications, thereby prompting the agent to "explore" the space of potentially-compliant operations with a higher probability.

Experimental Evaluation & Benchmark Dataset. We make two major contributions: (1) We constructed the first benchmark dataset, to our knowledge, that evaluates the ability to derive exploration specifications given a dataset and analytical goal. Our benchmark contains 182 pairs of analytical goals and corresponding exploratory specifications. (2) We conducted a thorough user study involving 30 participants to evaluate the relevance and overall quality of LINX exploration notebooks. The results are highly positive: Notebooks generated by LINX not only receive higher ratings for usefulness and relevance compared to baselines like ChatGPT and ATENA [6] but also significantly aid users in completing analytical tasks, providing 3-5 times more relevant insights.

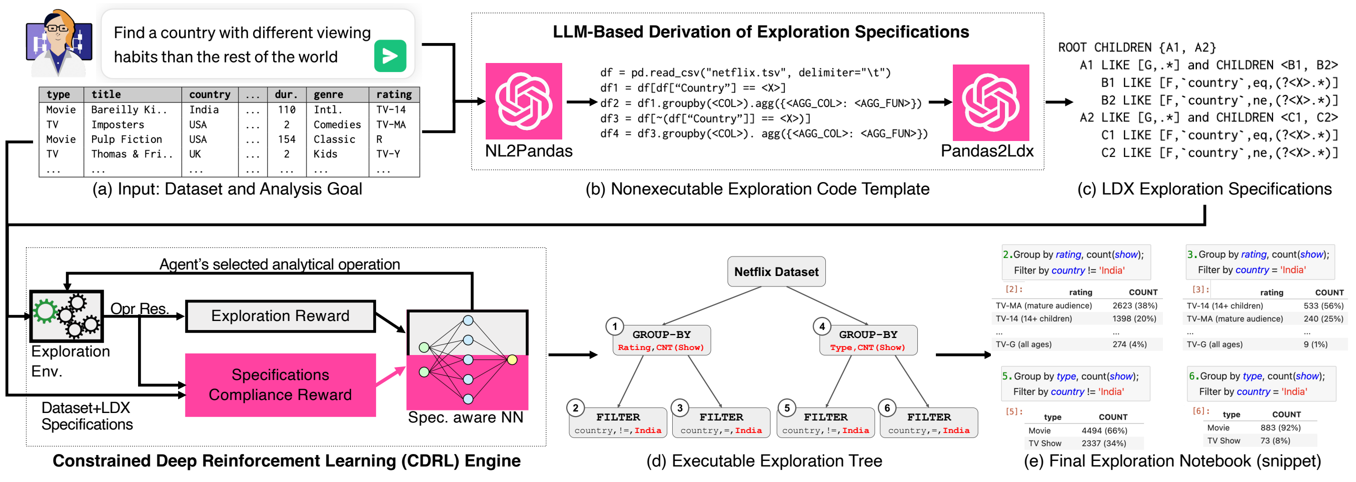
A recent demo paper [38] briefly introduces LDX and the CDRL engine, but with a main focus on a web interface for manual specification composition. Differently, in this work we present an end-to-end solution that only requires the user to describe their analytical goal in natural language.

[[[(Tomer) Is it worth adding a short paragraph to describe the paper's outline? (e.g. In Section 2 we... In Section 3..)]]]

2 RELATED WORK

Assistive Tools for Interactive Exploration. Assisting users in data exploration has been the focus on numerous previous works. Examples include simplified analysis interfaces for non-programmers [33, 60], explanation systems for exploratory steps, [12, 36], insights extraction [62], as well as recommender systems for *single* exploratory steps [13, 14, 17, 18, 26, 41, 54, 75]. While these works facilitate important aspects of the *interactive* analytical work, LINX focuses on generating *complete exploration notebooks*, known to be highly useful for analysts and data scientists [31, 46, 52].

Automated Data Exploration (ADE). More recent systems such as [5, 47**? ?** , 48] **[[[(Tomer) why do some of the citations appear as question marks?]]]** aim to generate an end-to-end exploratory process,



Find a country with different viewing habits than the rest of the world
(a) Input: Dataset and Analysis Goal
LLM-Based Derivation of Exploration Specifications
NL2Pandas
Pandas2Ldx
(b) Nonexecutable Exploration Code Template
(c) LDX Exploration Specifications
Agent's selected analytical operation
Opr Res.
Exploration Reward
Exploration Env.
Specifications Compliance Reward
Dataset+LDX Specifications
Spec. aware NN
Constrained Deep Reinforcement Learning (CDRL) 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
(e) Final Exploration Notebook (snippet)

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
...	...	...	...	...	...	...

```
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>})
```

```
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>.*)]
```

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%)

Figure 1: LINX🙀 Workflow Overview

given an input dataset, with the purpose of highlighting interesting aspects in the data, and providing preliminary (yet compound) insights. While the space of possible single exploratory step is large, the domain of full exploratory sessions is exponentially larger. Therefore, systems such as [5, 6, 47, 48] use sophisticated *deep reinforcement learning* (DRL) architectures, designed to effectively explore this vast space and generate useful, interesting exploratory sessions. However, these systems are *invariant* to the user's task and context. LINX CDRL engine is based on the DRL environment of ATENA [6], augmenting it with a novel specification language, as well as a corresponding reward scheme and a neural network architecture that guide the agent toward interesting sessions that comply with given specifications. The specifications themselves are derived from the user's *analysis goal*, using an LLM-based solution.

Specification Languages for Data Visualization. In this work we devise LDX, a specification language for data exploration. We note that previous work has introduced specification languages to facilitate the programmatic creation of data visualizations. Examples include Matplotlib [24] and Vega [72], and Vega-lite [55], Closer to our work, visualization recommender systems like [34, 71, 73] enable users to provide partial specifications. Similarly to these languages, LINX also uses parametric definitions for query operations – but does so in the context of defining full *sequences* of contextually-connected operations, rather than a single visualization as supported in [34, 71, 72].

Text-to-SQL. A prominent component in our work is an LLM-based solution that derives exploratory specifications from a textual description of an analytical goal.

This task draws some similarities with the task of *text-to-SQL*, where a structured query is to be translated from a natural language (NL) request [1, 32]. Recently, Text-to-SQL via LLMs [37, 49] has shown promising results, nearly comparable to dedicated architectures [81]. Large supervised datasets have played a pivotal role in advancing this research, enabling the training of machine learning models for this task [37, 77**?**] **[[[(Tomer) why is the last citation not shown?]]]**. Nevertheless, in our work we need to overcome several additional challenges: (1) The generative output in ADE contains 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. (2) LLMs are not explicitly trained on vast amounts of exploratory sessions, compared to the prevalence of relevant such information for writing SQL queries (in addition to numerous publicly-available benchmark datasets [77]). (3) Importantly, the task of deriving exploratory specifications also requires the LLM to decide which elements in the desired exploration sessions should be *discovered* in a data-driven manner, unlike in Text-to-SQL, in which the NL request is instantly translated to an executable query.

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 [16], columns annotation [**?**] and even the potential of substituting database query execution engines [43, 53, 63, 64]. These exciting research directions are orthogonal to our work.

3 PROBLEM SETTING, EXAMPLE WORKFLOW

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

Automated, goal-oriented data exploration. Given an input dataset D, and an analytical goal description g, we define the task of automatically generating a full exploration session comprised of N query operations: $Q_{(D,g)} = \langle q_1, q_2, \dots q_N \rangle$.

As in standard ADE settings, we assume a predefined set of query operation types. Following [6] we focus on the following *parametric* filter, group, and aggregate 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. $=, \geq$, *contains*) and `term` is a numerical/textual term to filter by. A *Group-by and Aggregate.* 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, 41], in which each query operation is represented by a corresponding 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 (See Figure 1d for an example exploration tree).

Now, given a *utility* notion for exploratory session, denoted U, a goal-invariant ADE system is tasked to generate a session Q_D such that $U(Q_D)$ is maximal. For instance, [6] defines a utility notion that assesses the *interestingness*, *diversity* and *coherency* of an exploratory session (See Section **??** for more detail).

In LINX, the objective is to generate a session $Q_{(D,g)}$ such that $U(Q_{(D,g)})$ is maximal (we use the definition of [6]), and also, that $Q_{(D,g)}$ is *relevant* w.r.t. analysis goal g. The relevance of a session is determined according to a set of exploration specifications, that is derived w.r.t. the goal g, as explained below.

Example workflow. Given a dataset D and analytical goal g, LINX first derives a set of exploration specifications that form a "skeleton" of the exploratory process that can accommodate a variety of compatible exploration paths. In the second step, our *CDRL ADE engine* generates a full session $Q_{(D,g)}$, which maximizes the exploration utility and is also compliant with the specifications derived from the goal g.

Figure 1 illustrates the detailed architecture and an example workflow of LINX.

Input: Data and analysis goal. First (See Figure 1a), the user provides a dataset and an analytical goal in natural language: *"Find a country with different viewing habits than the rest of the world, using the Netflix Movies and TV Shows Dataset"*. **[[[(Amit) refine this again]]]**

Step 1: Deriving Exploration specifications w.r.t. the goal. As mentioned above, we devise LDX, a specification language for exploratory sessions. LDX can be seen as an "intermediate" interface, allowing to represent goal-related instructions to the ADE agent. Given the dataset D and goal g, the primary challenges lie in (1) determining which properties of the desired exploration session can be immediately derived from g and which should be discovered in a data-driven manner; and (2) formally express the specifications in LDX.

We harness an LLM-based solution for this task. However, as described above, directly generating adequate LDX specifications is difficult for the language model, given the absence of such knowledge in its training data. We therefore employ a two-staged solution, in which LINX first prompts the LLM to generate an *non-executbale exploration Code* (See Figure 1b). Session elements that cannot be determined directly are marked with special placeholders (marked with <>) for query operations (or fragments thereof) that cannot be directly derived from the goal. The non-executable code is then translated to LDX via a subsequent prompt, as illustrated in Fig. 1c.

Following our running example (*"find an atypical country"*) LINX derives that the goal-relevant session should include a comparison of aggregated attributes when filtering in on a country, then out. However, determining *which* country to apply the filter to, and what exact aggregations to use need be discovered by the ADE agent.

Step 2: Generating a maximal-utility exploratory session, in accordance with the goal-driven specifications. We devise a CDRL ADE framework, as detailed in Section 6. Our framework is based on the data exploration DRL environment and exploration reward from [6], but with new components geared towards generating *specifications-compliant* sessions: we introduce a *compliance reward scheme* that employs the verification engine of LDX to gradually guide the agent in performing compliant sessions. The agent itself is built on a novel, specification-aware neural network architecture, which adjusts its structure based on the input specifications. This adaptation allows the selection of compliant operations with higher probability.

After the CDRL process converges, LINX produces an *executable exploration tree* (Fig.1d), consisting of an explicit sequence of analytical operations that adhere to the input specifications. The operation 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*, a popular analytical interface among data scientists [31**?**]. The interface is used for both performing data analysis and also for presenting and sharing analytical products. Users can examine the output session, then continue the exploration process in the same interface. Fig.1e illustrates a snippet of the resulted notebook. 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 viewing habits. For instance, the *rating* comparison (See notebook cells 2 and 3) reveals that *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+)*. Another noteworthy insight from the *show type* comparison (Cells 5 and 6) indicates that *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)*.

Limitations & Assumptions Discussion. We conclude this section with a discussion regarding the scope and focus of LINX:

- *ADE Vs. interactive exploration.* LINX is primarily designed as a goal-oriented ADE system, which generates full exploratory sessions, similarly to systems such as [**?**]. ADE do not substitute manual, interactive exploration, but support it by coherently presenting users with preliminary insights on the data. The gained knowledge give users some head start in completing their analytical assignments – as in the case of examining human-generated exploration notebooks on sites such as Kaggle and Github [31].
- *Supported query operations.* LINX, following [6] currently supports filter, group and aggregate queries. As we show in our experimental evaluation, exploratory sessions composed of these operations are highly useful. However, more operations can be introduced, by extending the underlying DRL environment for data exploration (See [6] for a discussion).
- *Problem complexity.* In comparison to goal-invariant ADE, where a maximum-utility exploration session Q_D is generated, our problem introduces an additional complexity, as we require goal relevance *and* maximum utility from the generated session $Q_{(D,g)}$ (due to a reward-sparsity problem). However, in

some cases the input specifications may also restrict the search space of all exploration sequences Q_N. For instance, in our running example, the specifications restrict the filter operations to the attribute *'country'*. This allows LINX, as shown in Section 7.4, to retain similar convergence times and success rates as [6].

4 LDX: SPECIFICATION LANGUAGE FOR DATA EXPLORATION

We introduce LDX, an intermediate language designed for specifying exploratory sessions. To the best of our knowledge, this is the first language enabling the partial specification of an entire *sequence* of interconnected analytical operations, as opposed to a single query or operation.

LDX and its verification engine, detailed below, play a vital role in generating personalized exploration notebooks. The LDX specifications produced by the LLM component only *partially* define the desired notebook, leaving the remaining elements to be discovered in a data-driven manner. These specifications are then passed to the CDRL engine of LINX, which internally utilizes the LDX verification engine, to ultimately generate an insightful exploration tree that adheres to the input specifications (See Section 6).

4.1 LDX Language Overview

An exploratory notebook, as mentioned above, is comprised of a sequence of analytical operations. Each operation may be employed on the input dataset or on the results of a previous operation. Importantly, the operations in the notebook are semantically connected, thus forming a *narrative* [31, 46] – the contextual connection between the operations, organized in a logical, coherent manner, that gradually leads the user to nontrivial insights on the data.

LDX therefore allows posing specifications on the structure of the notebook and the operations it will contain. Then, to further consider the *contextual* connection between query operations, LDX introduces *continuity variables.* These variables are used to *match* between parameters of different operations, without explicitly stating them. For example, after a group-by operation is performed on *some attribute*, we may want to restrict the next operation to be a filter on *the same attribute* used in the preceding group-by.

The LDX specifications, which, as mentioned above, only partially define the output notebook, are used to guide the CDRL engine of LINX, when generating the final exploratory notebook.

Our specification language LDX extends Tregex [35], a query language for tree-structured data. Tregex natively allows partially specifying structural properties of the tree, as well as the nodes' labels. In our context, this allows specifying the execution order of the notebook's operations and also their type and parameters.

The basic unit in LDX is a *single node specification*, which addresses the structural and operational preferences w.r.t. a single query operation. A full LDX query is then composed by conjuncting multiple single-node specifications, interconnected using the *continuity variables*, as explained below.

We begin with a simple "hello world" example, then describe the LDX constructs in more detail. A full LDX guide with more examples can be found in [50].

Example 4.1. The following LDX query describes a simple exploratory session with two analytical operations: a group-by, followed by a filter query, both employed on the full dataset. We further specify that the filter is to be performed on the same attribute as the group-by. The rest of the parameters are *unspecified*, and will be completed using the LINX CDRL engine.

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

In the query, the `ROOT` node represents the raw dataset, and its two children `A` and `B` – the group-by and filter operations. `A` is a group-by with "free" parameters: unspecified column, aggregation function, and aggregation column. The continuity variable `X` captures the group-by column. `B` is a filter operation (operated on the full dataset) with unspecified operator and term. See that filter *column* parameter is also free, but since it is also captured by the same continuity variable `X`, the CDRL engine will employ both the filter and group by operations, on the same column.

We next describe how operations' syntax, structure and continuity are defined in LDX.

Specifying exploration tree structure. The order of execution, as well as the connections between the operations in the notebook are 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, note 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). Last, since Operation `B` is *named*, it can take its own set of structural/operational specifications, and be connected to other named operations via *continuity variables*, as described next.

Specifying analytical operations. LDX allows for *partially* specifying the operations using *regular expressions* (regex), as they define *match patterns* that can cover multiple instances. Importantly, this syntax enables leaving "free" analytical operations' parameters, which will be later instantiated in a data-driven manner by the LINX CDRL engine (as described in Section 6). 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 $*$). The exact choice of the aggregation function and attribute, is deferred to the CDRL engine of LINX.

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 (by the CDRL engine). 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 1b) 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>.*)]`'

Algorithm 1: LDX Query Compliance Verification

```
VerifyLDX ($T, L_X, A = \langle \phi_V = \{$ROOT:0$\}, \phi_C = \emptyset) \rangle$)  // Inputs:
    Exploration tree $T$, LDX Specifications list $L_X$, assignment $A$
1   if $S = \emptyset$ then return True
2   $s \leftarrow L_X.pop()$                 // get a single LDX specification
3   for $c \in Cont(s)$ do                 // Assign continuity vars in $s$
4       if $c \in \phi_C$ then $s.c \leftarrow \phi_C(c)$
5   $V^s_T \leftarrow$ GetTregexNodeMatches$(s, T, \phi_V)$
6   for $v \in \mathcal{V}^s_T$ do
7       $\phi^s_V \leftarrow \phi_V \cup \{Node(s) : v\}, \phi^s_C \leftarrow \phi_C$
8       for $c \in Cont(s)$ do          // Update continuity mapping
9           $\phi^s_C(c) \leftarrow v.c$
10      if VerifyLDX$(T, L_X, \langle \phi^s_V, \phi^s_C \rangle)$ then
11          return True
12  return False
```

– 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 1b, 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.

4.2 LDX Verification Algorithm

We next describe our LDX verification engine, which takes an exploratory tree (session) T and a LDX query Q_X, and verifies whether T is compliant with Q_X.

For an input LDX query Q_X, we denote the set of its named nodes (operations) 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, an *assignment* $A(Q_X, T) = \langle \phi_V, \phi_C \rangle$, s.t., (1) ϕ_V is a *node mapping function*, assigning each named node $n \in Nodes(Q_X)$ an operation node $v \in V(T)$ 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 $\phi_V(\text{ROOT}) = 0$, i.e., mapping the root node in the LDX query to the root node of T.

LDX Verification Algorithm. We consider an LDX query Q_X as a set of *single* specifications, denoted L_X, s.t. each specification $s \in L_X$ 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 $Cont(s)$. Our verification procedure, as depicted in Algorithm 1, takes as input an LDX specifications list S, an exploration tree T, and the initial assignment A, in which ϕ_V contains the trivial root mapping (as described above) and an empty continuity mapping ϕ_C. Note that since Tregex does not support continuity variables, we only use its node matching function *GetTregexNodeMatch* [61] to obtain an initial matching between the nodes in $Nodes(L_X)$ to the nodes in $V(T)$. We then maintain the continuity mapping ϕ_C, and dynamically update the specifications in L_X with concrete values as follows: In each recursive call, a single specification s is popped from S (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. This function, as described in [61], returns all valid node matches for $Node(s)$, denoted V^s_T, given the current state of the node mapping ϕ_V. Then, for each valid node $v \in V^s_T$, 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 satisfies s, *including the mapping* ϕ_C, therefore only unassigned variables in $Cont(s)$ are updated.) Once both mappings are updated (denoted ϕ^s_V and ϕ^s_C), we make a recursive call to *VerifyLDX* (Line 10), now with the shorter specifications list L_X (after popping out s) and the new mappings (ϕ^s_V, ϕ^s_C). Finally, the recursion stops in case there is no valid assignment (Line 12) or when the specification list L_X is finally empty (Line 1).

In Section 6 we describe how the LDX verification algorithm is used in our CDRL engine of LINX.

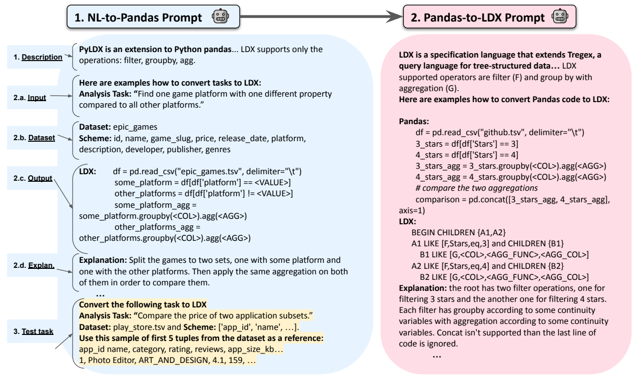
5 LLM-BASED SOLUTION FOR DERIVING EXPLORATION SPECIFICATIONS

As previously mentioned, when given a description of an analytical goal and a dataset, the primary challenges in generating exploration specifications are (1) determining which elements in the output session can be derived apriori, leaving the remainder for data-driven discovery, and (2) expressing the specificaiotns using a syntactically correct LDX query, to support effective verification by the CDRL engine (See Section 6).

To address these challenges, we introduce an LLM-based component using in-context learning [15]. As mentioned above, while a similar approach has recently proven useful in the task of Text-to-SQL [49], we contend that deriving LDX specifications may be even more challenging, as LDX *partially* depict a *sequence* of operations rather than a single, cohesive query (refer to Section 2 for a discussion). Moreover, the success of LLMs in text-to-SQL is significantly influenced by the abundant availability of resources as well as parallel text-to-SQL datasets [23, 69, 77, 80, 81] . Such resources are unavailable for LDX.

Approach Overview. To overcome the absence of NL-LDX information in the LLM training data, we use a *few-shot* setting, renowned for its excellent performance across diverse analytical goals [42, 67]. In this approach, several illustrative examples are provided to the LLM before soliciting task completion.

To further enhance the distillation of exploration specifications, we employ an *intermediate code representation.* Taking inspiration from analogous solutions for tasks such as arithmetic and commonsense reasoning [9, 21, 39, 79], 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 [68] code. 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.* As we empirically show in Section 7.2, our two-stage


1. NL-to-Pandas Prompt
2. Pandas-to-LDX Prompt
1. Description
2.a. Input
2.b. Dataset
2.c. Output
2.d. Explan.
3. Test task

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

Here are examples how to convert tasks to LDX:
Analysis Task: "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

```
LDX:    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.

...

Convert the following task to LDX
Analysis Task: "Compare the price of two application subsets."
Dataset: play_store.tsv and **Scheme:** ['app_id', 'name', …].
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, …

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

approach exhibits superior generalization compared to a direct NL-to-LDX approach. It performs better when the dataset and/or goals are new to the LLM (i.e., absent from the few-shot prompt examples).

Prompt Engineering. Figure 2 depicts a snippet of our chained prompts: NL-to-Pandas and Pandas-to-LDX. We further designed an additional baseline prompt, directly mapping NL-to-LDX (See Section 7.2).

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. 2) ; (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 [7, 65].

Step (d) is influenced by the Chain-of-Thought (CoT) prompting paradigm, which has demonstrated enhanced performance in multi-step tasks [59, 66, 67, 76]. Following the CoT methodology, we incorporate an explanation for each few-shot example. We use *least-to-most prompting* [82], 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.

To evaluate solution, we constructed a new NL-to-LDX dataset, consisting of 182 instances of analysis goals and corresponding LDX specifications. We describe the dataset construction in §7.1 and present the experimental results in §7.2. The full versions of all of our prompts, including the NL-to-LDX baseline, are provided in [50].

6 LINX CDRL ENGINE

Given a dataset and LDX exploration specifications, our goal is to generate a *useful*, interesting sequence of operations that are also *compliant* with the given specifications. As previously shown

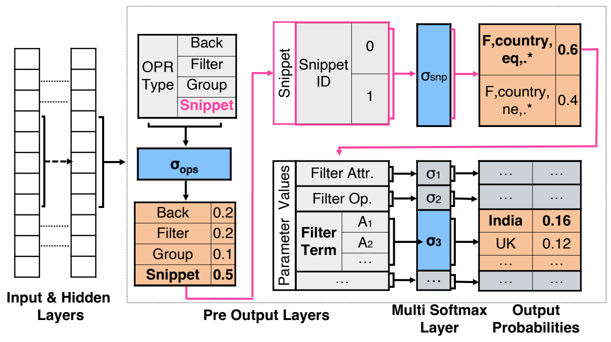


Figure 3: Specification-Aware Network Architecture

in [5, 6, 48], reinforcement learning is a powerful paradigm for generating exploratory sessions, as it can quickly dissect the vast domain of all possible sequences of query operation, focusing on high-utility ones.

LINX is based on the goal-invariant engine of [6] augmenting it with a new reward scheme and network architecture for supporting input custom specifications. We next define the Markov Decision Process (MDP) model and the environment of our CDRL framework, then delve into the LDX-compliance reward scheme and specification-aware network we use.

6.1 MDP Model for Goal-Oriented Exploration

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. The MDP model corresponds with our goal-oriented exploration session, as defined in Section 3: In each episode the agent performs a multi-step exploration session $Q_{(D,g)}$, where in each step i it employs a parametric query operation, as defined in Section 3. The agent is then transferred to state $s_i = \Delta_a(s_{i-1}, a)$. s_i represents the resulted view of query q_i, when employed on the previous results of query q_{i-1} (s_0 is the initial state, representing the raw dataset) , and obtains a reward $R_a(s_i, a) := \alpha \cdot R_{gen}(s_i, a) + \beta \cdot R_{rel}(s_i, a, Q_X)$, which is composed of two parts: (1) $R_{gen}(s_i, a)$ is the general, goal-invariant reward $R_{gen}(s_i, a)$. in our implementation, we use the exploration utility notion from [6], based on interestingness, diversity and coherency (yet other notions, e.g. familiarity/curiosity as defined in [48], can be used instead). (2) $R_{rel}(s_i, a, Q_X)$ is a *goal-relevance* reward, received for action a, based on the input LDX specifications Q_X. This reward component, together with the neural network architecture described below, allow LINX CDRL agent to generate exploratory sessions that comply with the specifications derived from the user's analytical goal.

6.2 LDX-Compliance Reward Scheme

Given an input LDX query Q_X, and an output exploratory session T, we next define our *relevance* reward signal, received at each step i: $R_{rel}(s_i, a, Q_X) := \gamma \cdot EOS(s_i, a, T, Q_X) + \delta \cdot IMM(s_i, a, T_i, Q_X)$, where EOS is an *end-of-session* feedback signal, and IMM is received immediately, as described below.

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 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.* Namely, if the agent learns to generate *correct* operations in an *incorrect* order/structure, the learning process becomes largely futile as the agent needs to relearn from scratch, to employ the desired operations in the correct order.

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 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 is not compliant with Q_X, we check its compliance only with *structural specifications* of Q_X (see Section 4.1) using the Tregex engine. If no valid assignments are found, a fixed negative penalty is applied. (3) If T satisfies structural specifications but not *operational* ones, a non-negative reward is assigned based on the number of satisfied operational 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 $IMM(s_i, a, T_i, Q_X)$ granted individually for each step i. This real-time signal negatively rewards specific operations that violate the structural specifications in Q_X.

To do so, we use a modification of the LDX verification engine (Algorithm 1), that can operate on an *ongoing* session T_i rather than a full session T. Intuitively (See Appendix A.3 for more details), we assess the possibility of a *future* assignment satisfying the *structural* constraints of 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 more details).

6.3 Specification-Aware Neural Network

To further assist the agent in increasing the probability of choosing compliant operations, we draw inspiration from *intervention-based* CDRL solutions [10, 56], suggesting to externally override the agent's probabilities in order to avoid "unsafe" actions. Following this line of research, we devise a *specification-aware neural network*, which infers its pre-output layer from the input LDX query and the dataset. Rather than *changing* the agent's chosen operation, the goal of this network design is to *encourage* the agent to use compatible operations more frequently.

Figure 3 depicts the network architecture (Specifications-aware functionality is highlighted in pink). As in [6], our network architecture uses *pre-output* layers, allowing the agent to first choose an operation type (e.g., filter, group-by, back), and only then the required parameters. this is done via separate *softmax* segments, one for each individual parameter. In LINX CDRL we augment this network design with built-in placeholders for operation "snippets" that are derived from the LDX specifications Q_X. The input layer (Fig. 3, left-hand-side) receives an observation vector, representing the current "state" in the exploration session, and passes it to the dense hidden layers. Then, the agent composes an analysis operation via the pre-output layers, where it first chooses an operation type and subsequently its corresponding parameters. As depicted in Figure 3, Softmax Segment σ_{ops} is connected to the operation types, and Segments $\sigma_1, \sigma_1, \ldots$ are connected to the value domain of each parameter.

To further encourage the agent to choose compliant operations, 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 S_{opr}. 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‘`, will only require the agent to choose a filter term, rather than composing the full query operation.

The architecture derivation process is as follows. (See the pink elements in Figure 3.) First, given an LDX query Q_X, we generate an individual snippet neuron for each remaining specification $s \in S_{opr}$. In case S contains a disjunction, we generate an individual snippet for each option. All snippet neurons, as depicted in Figure 3, are connected to σ_{snp}, the snippet multi-softmax segment. Now, since an operational specification $s \in S_{opr}$ typically restricts only *some* of the operation's parameters, the agent still needs to choose values for the "free" parameters, denoted P_s. We therefore wire the snippet of each operational specification s to the multi-softmax segments of its corresponding free parameters P_s, i.e, $\{\sigma_p \mid \forall p \in P_c\}$. The following example demonstrates the action selection process.

Example 6.1. Figure 3 depicts an example selection process, in which the selection probabilities are already computed by the agent's neural network (colored in light orange). First, the agent samples an operation type using the multi-softmax segment σ_{ops}. As in Figure 3, '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'.

This network design, as we empirically show in Section 7.4, allows LINX to consistently generate compliant exploration sessions in 100% of the datasets and LDX queries in our experiments.

7 EXPERIMENTS

We implemented our framework in Python 3: The LDX verification engine utilizes the Tregex Python implementation in [61], and our CDRL engine is built in ChainerRL [20], based on the generic DRL framework for data exploration in [25]. All our experiments were run on a 24-core CPU server. The full experiments code and data are provided in our Github repository [50].

	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 Exploration Specification Benchmark (182 Instances):

We then conducted extensive experiments along three key facets: (1) Success in deriving correct LDX specifications given an analytical goal and dataset; (2) relevance and usefulness of auto-generated exploratory notebooks; and (3) performance and ablation study of LINX CDRL engine.

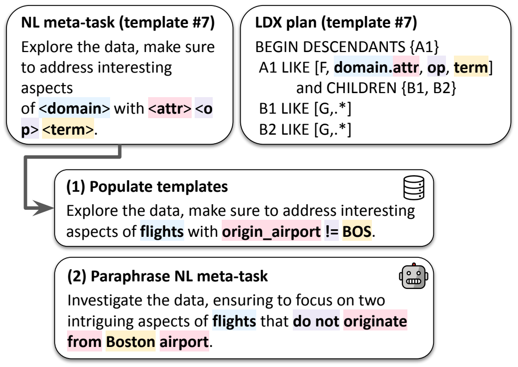
We next describe the construction and properties of our benchmark dataset, then provide the details for each experiments set.

7.1 Benchmark Dataset for Goal-oriented Exploration Specifications

We constructed the first benchmark dataset, to our knowledge, for goal-oriented exploration specifications. Our dataset comprises 182 pairs of analytical goals and their corresponding exploration specifications in LDX using three different tabular datasets: *(1) Netflix Movies and TV Shows Dataset* [29], listing about 9K titles on Netflix. Each title is described using 11 attributes such as the country of production, duration/num. of seasons, etc. *(2) Flight-delays Dataset [27]*, containing 5.8M records describing domestic US flights using 12 attributes such as origin/destination airport, flight duration, issuing airline, departure delay times, delay reasons, etc. *(3) Google Play Store Apps [28]*, which holds 10K mobile apps available on the Google Play Store. Each app is described using 11 attributes: Name, category, price, num. of installs, etc.

To build the benchmark dataset, we first characterized 8 exploration “meta-goals”, as depicted in Table 1. The meta-goals were selected by analyzing 36 real-life exploration notebooks available on Kaggle, for the Netflix, Flights, and Playstore datasets (See [27–29]), and in accordance with [71] and [3], in which the authors identify common analytical tasks, among them several open-ended exploration goals (questions). We then chose an exemplar *concrete* goal for each meta goal (See g_1-g_8 in Table 1), and composed LDX specifications $Q^1_X - Q^8_X$ based on the content of relevant exploration notebooks on [27–29]. Q^1_X is depicted in Figure 1c, and the rest of the queries are available on [50].

Then, to extend our dataset from 8 pairs of analytical goals and corresponding specifications to 182, we adopted the scheme outlined in Figure 4. First, we stripped each exemplar pair (g_i, Q^i_X) from any dataset-related trait 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` [27] data domain, the `origin_airport` attribute, operator $\neq$,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


NL meta-task (template #7)
Explore the data, make sure to address interesting aspects of <domain> with <attr> <op> <term>.

LDX plan (template #7)
```
BEGIN DESCENDANTS {A1}
 A1 LIKE [F, domain.attr, op, term]
        and CHILDREN {B1, B2}
  B1 LIKE [G,.*]
  B2 LIKE [G,.*]
```

(1) Populate templates
Explore the data, make sure to address interesting aspects of flights with origin_airport != BOS.

(2) Paraphrase NL meta-task
Investigate the data, ensuring to focus on two intriguing aspects of flights that do not originate from Boston airport.

Figure 4: NL-to-LDX Benchmark Dataset Generation

(implemented with ChatGPT). This resulted in a more naturally phrased, rich, and diverse set of 200 analytical tasks, out of which we manually discarded 18 nonsensical goals, that did not reflect a realistic user intent. Table 1 lists total number of instances for each meta goal (See [50] 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 direct, single prompt approach.

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 [30, 74, 77]. For example, Text-to-SQL performance assessments often rely on query execution results [51, 57, 74], but this is unsuitable for LDX specifications as they span a multitude of compliant output sessions. Alternative measures include exact string match [19, 77], and graph edit distance, commonly used for graph semantic parsing tasks [8, 30]. Drawing inspiration from these, we introduce two measures for comparing the generated LDX queries against the gold benchmark queries: the *Two-way Levenshtein Distance*, which considers the query strings, and the *Exploration Tree Edit Distance* from [41], focusing on the parsed output of the queries.

(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 distance metric designed for exploration trees, as proposed in [41]. This measure is a standard tree edit distance [78], incorporating a dedicated label distance function to assess the distinction between two parametric analytical operations (See [41] for full detail). To apply this metric, we construct a minimal tree for each compared LDX query while masking the continuity variables, see §B.2 for more details. The score is normalized by dividing it by the maximum tree size minus one, disregarding the root node comparison as it is consistently identical.

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.

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 5, 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. Note, however, that even in the simpler scenarios, (1) and (3), the model never receives a few-shots example that contains the test analytical goal.

To evaluate the efficacy of our NL2PD2LDX chained prompt solution (P2), we contrasted it with a direct NL2LDX prompt, where the LLM is tasked with directly generating LDX specifications based on the provided analytical goal (See Appendix B.1). We assessed the performance for both ChatGPT (gpt-3.5-turbo)[45] and GPT-4 [44].

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) 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, seen meta-goal*, the chained solution yields higher scores for both LLMs, with GPT-4+PD slightly outperforming ChatGPT+PD.

Model\Settings	**Seen Meta-Goal**		**Unseen Meta-Goal**	
	lev^2	$xTed$	lev^2	$xTed$
Seen Dataset				
ChatGPT	0.87	0.87	0.64	0.6
ChatGPT + Pd	0.89	0.89	0.72	0.69
GPT-4	**0.97**	**0.97**	0.71	0.7
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	0.68
GPT-4	**0.95**	**0.95**	0.71	0.7
GPT-4 + Pd	0.94	0.93	**0.73**	**0.71**

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

Summary. Our experimental results show the efficacy of our solution, even when the entire class of analytical goals (i.e., all goals with a similar intent) or the dataset are new to the model. We note that further improvements can be achieved, especially for the latter, most challenging scenario where both the datasets and the meta-goals are unseen. We therefore make our benchmark dataset publicly available [50], to facilitate the evaluation of future solutions.

7.3 Relevance and Quality of Output Exploration Notebooks (User Study)

We next evaluate the overall quality and relevance of exploration notebooks generated by LINX, compared to notebooks generated by alternative baselines. We conducted both a *subjective* study, were users are asked to rate notebooks 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 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 [50] for space constraints), to obtain a total of four different goals for each of our three datasets.

We used LINX CDRL engine to generate an exploration notebook for each goal and dataset, and evaluated them against 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

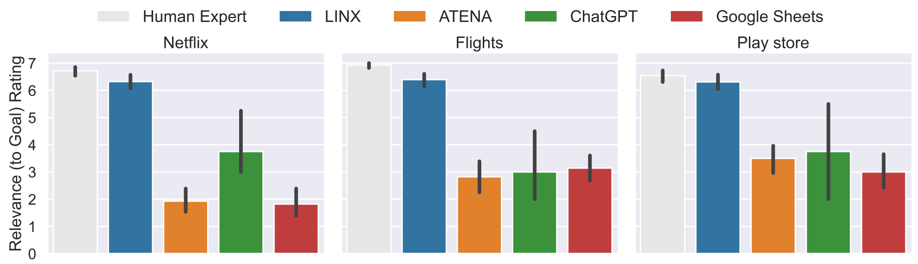


Figure 5: User Study – Relevance Rating of Exploration Notebooks to the Given Goal

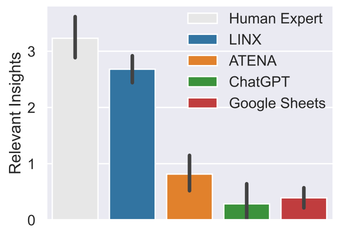


Figure 6: Avg. Num. Insights

Insight
"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

notebook for all four tasks of each dataset. **(2) ChatGPT (gpt-3.5-turbo) [45].** In this baseline we generated notebooks by asking the LLM to compose an entire notebook, containing Pandas code, for a given description of the dataset (attribute names and types) and a analytical task. We then made the code runnable and executed it in a Jupyter notebook. **(3) Google Sheets Explorer [60].** A commercial ML-based exploration tool that accommodates user specifications (yet rather limited ones – columns and data subsets to focus on). The specifications were again composed by the three experts. For example, for task T5 ("properties of summer flights"), the expert chose the columns 'month', 'airline', 'delay-reason' and 'scheduled arrival'/'departure', focused on flights from July and August. **(4) Human Expert.** Last, we used 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 [50].

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, 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 (resp.) for its automatically generated sessions.

Next, see that the relevance ratings of ChatGPT are lower (3 to 3.75). While ChatGPT does support natural language specifications. Its notebooks (recall from Example 1.1) provides descriptive statistics and simple aggregations, without undertaking a more comprehensive exploration of underlying data patterns. The scores of ATENA and Google Sheets are lower, reaching about 2-3 out of 7. Naturally, the fact that ATENA does not support user specifications, and Google Sheets supports only limited specifications – makes their solutions insufficient for generating *relevant* notebooks for the given analytical goals.

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.)

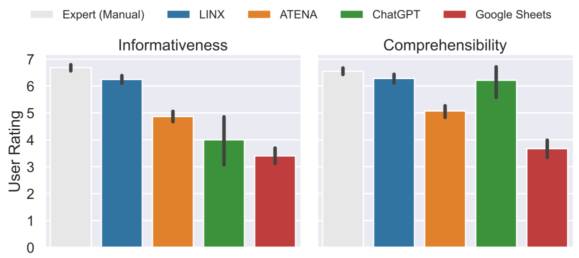


Figure 7: Informativeness & Comprehensibility Rating

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.

Interestingly, LINX still obtains higher scores than ATENA, ChatGPT, and Google Sheets, (6.24 and 6.28 for informativeness and coherency). This particularly shows that LINX does not compromise on informativeness or comprehensibility, when generating *goal-oriented* exploratory sessions.

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

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 exploration 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. This is because for the vast majority of tasks, users could not derive any explicit insight (since, as mentioned above, ChatGPT notebooks contained mostly general descriptive statistics).

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 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, then a convergence 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 (§6.2 and specification-aware network (§6.3). Instead, it uses the basic neural network of [6]. **(2) Binary+Imm. Reward** uses the reward scheme, as described in Section 6.2, 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].

We employed each baseline on the same 12 LDX queries used in the user study, and examined how many of the generated exploration sessions were indeed compliant with the input specifications. 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.

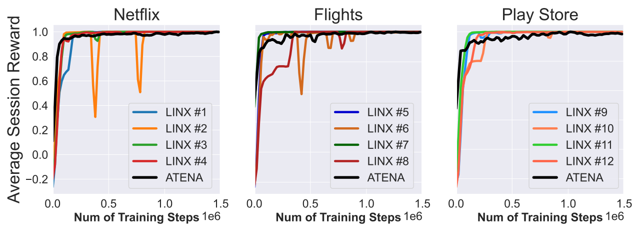


Figure 8: Convergence Comparison to ATENA

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 6.3, is particularly useful in encouraging the agent to perform specification-compliant operations – despite the inherently large size of the action-space.

Convergence & Running Times. Last, we examine the convergence and running times of our CDRL engine, and compare them with ATENA [6], in order to assess whether its more complex architecture affects performance. Figure 8 depicts 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'*, where the black line in each figure depicts the convergence process of ATENA [6] which serves as a baseline. (Recall that ATENA can only produce one, generic exploration process per dataset.) As the maximal reward varies, depending on the LDX query and dataset, we normalize the rewards in the plots s.t. the maximum obtained reward is 100%.

First, see that the convergence process of both ATENA and LINX-CDRL are roughly similar, both fully converging to 100% of the maximal reward after 1M steps at most. LINX-CDRL sometimes shows sudden drops in reward (e.g., for g_2 and g_6). These drops stem from the heavy penalty for violations of structural specifications. As can be seen in the plots, the agent soon "fixes" these violations and stabilizes the reward. Furthermore, see that LINX-CDRL sometimes converges even faster than ATENA (e.g., for the Play Store dataset), in which ATENA takes 0.85M steps and LINX only 0.4M on average. This happens as the input specifications assist the agent in focusing on a narrower space of promising sessions, and eliminate the need for further exploring areas of lesser relevance.

We further examined whether the LDX-compliance reward scheme affects the running times of LINX. In practice, for all LDX queries, the computation times of the full compliance reward were negligibly small compared to the entire learning process – 0.26% on average. On our CPU-based server, this takes an average of about 2.5 minutes out of a total of 98 minutes for an entire learning process

(Recall that similarly to ATENA, LINX is not used for interactive analysis, thus such times are acceptable.)

Particularly, the compliance verification (Algorithm 1) takes only 0.001%, the operational-based reward takes 0.007%, and as expected, the costliest component is the immediate reward, with 0.25% of the total computation. The rest of the time is spent on more expensive components such as performing the query operations on the data and computing the gradients in the learning process.

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 and running times of LINX are on par with ATENA.

8 CONCLUSION & FUTURE WORK

This paper introduces LINX, a generative system for automated, goal-oriented exploration. Given an analytical goal and a dataset, LINX combines an LLM solution for deriving exploratory specifications and a CDRL based ADE engine that takes the custom specifications and generate a personalized exploratory session in accordance with the input goal.

In future work, we will will explore ways in which LLMs can further enhance the analytical process. A promising direction is to utilize LLMs for augmenting LINX notebooks with additional elements like captions, explanations, and visualizations, while also considering auto-visualization solutions such as [34, 73]. Another direction is the adaptation of LINX to interactive analysis and data manipulation code generation.

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] 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

[8] 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

[9] 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

[10] 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).

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

[12] 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.

[13] 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.

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

[15] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. 2023. A Survey on In-context Learning. arXiv:2301.00234 [cs.CL]

[16] 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

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

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

[19] 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

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

[21] 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

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

[23] Moshe Hazoom, Vibhor Malik, and Ben Bogin. 2021. Text-to-SQL in the Wild: A Naturally-Occurring Dataset Based on Stack Exchange Data. *ArXiv* abs/2106.05006 (2021). https://api.semanticscholar.org/CorpusID:235377010

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

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

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

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

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

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

[30] 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

[31] 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*.

[32] 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

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

[34] 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.

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

[36] 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.

[37] 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]

[38] 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.

[39] 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

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

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

[42] 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]

[43] 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

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

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

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

[47] 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.

[48] 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.

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

[50] LINX Github Repository. 2024. https://github.com/analysis-bots/LINX.

[51] 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

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

[53] 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

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

[55] 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.

[56] 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).

[57] 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

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

[59] 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

[60] Google Sheets Explore. 2024. https://www.blog.google/products/g-suite/visualize-data-instantly-machine-learning-google-sheets/.

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

[62] 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.

[63] 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

[64] 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

[65] 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

[66] 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

[67] 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

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

[69] Tomer Wolfson, Daniel Deutch, and Jonathan Berant. 2022. Weakly Supervised Text-to-SQL Parsing through Question Decomposition. In *Findings of the Association for Computational Linguistics: NAACL 2022*, Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (Eds.). Association for Computational Linguistics, Seattle, United States, 2528–2542. https://doi.org/10.18653/v1/2022.findings-naacl.193

[70] 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

[71] 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).

[72] 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).

[73] 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.

[74] 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

[75] 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.

[76] 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

[77] 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

[78] 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.

[79] 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

[80] Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2023. ScienceBenchmark: A Complex Real-World Benchmark for Evaluating Natural Language to SQL Systems. arXiv:2306.04743 [cs.DB]

[81] 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

[82] 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 [35], 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 [58], evaluating an LDX query may require, in the worst case, iterating over all possible node assignments, of size $\frac{|T|!}{(|T|-|Nodes(Q_X)|)!}$. 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 and the query Q_X are rather small[3].

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 performed by the agent and the set of specifications L_X (from the LDX query Q_X), we first check (Line 1) whether the session T is compliant with LDX using

[3]For example, the mean session size in the exploratory sessions collection of [41] is 8.

Algorithm 2: End-of-Session Conditional Reward

```
   Input: Exploration Tree $T$, LDX Specification List $L_X$
   Output: Reward $R$
1  if $\textbf{\textit{VerifyLDX}}(L_X, T) = True$ then
       // $T$ is compliant with $L_X$
2      return $POS\_REWARD$
3  $S_{struct} \leftarrow$ Structural specs of $L_X$
4  $\Phi_V \leftarrow GetTregexNodeAssg(S_{struct}, T)$
5  if $\Phi_V = \emptyset$ then
       // $T$ violates Structural specs
6      return $NEG\_REWARD$
   // Calculate operational-based reward:
7  $S_{opr} \leftarrow$ Operational specs of $L_X$
8  return $\max_{\phi_V \in \Phi_V} \textbf{\textit{GetOprReward}}(\phi_V, S_{opr})$

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

Algorithm 1. In case T is compliant, a high positive reward is given to the agent (Line 2).

Next, in case T is not compliant with Q_X, we now check whether it is compliant with the *structural specifications* in Q_X, denoted S_{struct}. 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 (Line 3). If there are no valid assignments, it means that T is non-compliant with the specified structure, therefore a fixed negative penalty is returned (Line 6).

In case T 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_i, i.e., after i steps, and the remaining 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 S_{struct}. This is simply done by calling the Tregex match function $GetTregexNodeAssg(S_{struct}, T^{\star})$, with each possible *tree completion* (denoted $T^{\star}$) for the ongoing exploration tree. The completion of the ongoing exploration tree T_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 [41]). 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 can show 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.

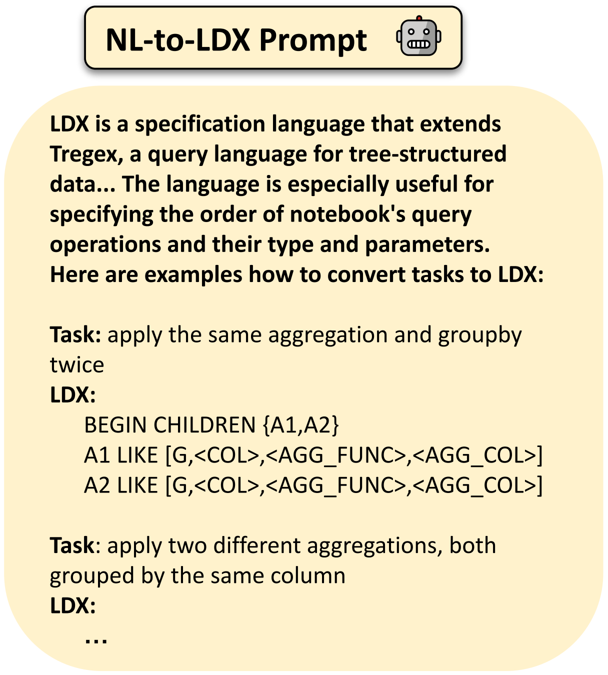
In more details, the immediate reward procedure at step i of an ongoing session, has $N - i$ remaining nodes to complete a full possible session tree. In order to bound the number of possible trees at each iteration, we will examine the iteration with the largest number of completions, which is right 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 [11]. 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.

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 §5: (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.



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

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.

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.