Generating Personalized Data Exploration Notebooks with LLMs and Constrained Reinforcement Learning

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

In this work, we present LINX, a language-driven exploration system, designed to generate personalized data exploration notebooks based on a given dataset and analytical task in natural language. LINX integrates a Large Language Model (LLM) in the generative process, while overcoming LLMs' inherent incapability to perform comprehensive interactions with the data. This is done using a two-stage solution: We first use an LLM-based component to construct an exploration *plan*, serving as the foundational structure for the desired output notebook. The plan differentiate between elements that can be directly inferred from the task description, and those which should be discovered in a data-driven manner. Subsequently, we employ a *constrained* Deep Reinforcement Learning (CDRL) engine, taking the dataset and LLM-generated plan as input and using them to generate the complete notebook. Our CDRL engine is based on ATENA, a task-agnostic DRL architecture for data exploration, augmenting it with a new *compliance reward scheme*, and a *specification-aware neural network architecture*. Both enhancements are crucial for generating useful, *personalized* exploratory notebooks. Our experimental evaluation demonstrates that notebooks generated by LINX are notably more relevant and useful compared to ones generated directly by ChatGPT, as well as other baseline solutions.

PVLDB Reference Format:
Tavor Lipman, Tova Milo, Amit Somech, Tomer Wolfson, and Oz Zafar. Generating Personalized Data Exploration Notebooks with LLMs and Constrained Reinforcement Learning. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/analysis-bots/LINX.

1 INTRODUCTION

Data exploration constitutes a crucial yet formidable process for data scientists and analysts. When confronted with an analytical task or inquiry concerning a dataset, users sequentially execute exploratory queries, such as filtering and grouping, to extract pertinent insights for their task. One of the most effective methods to kickstart this intricate process is to examine existing *data exploration notebooks* [28, 46, 53]—curated, illustrative exploratory sessions crafted by other data scientists on the same dataset, often shared on collaborative online platforms like Kaggle, Google Colab, or Github. However, in many cases, such notebooks are unavailable or nonexistent. In a previous work, we introduced ATENA [5]—a system designed to automatically generate general-purpose exploration notebooks based on an input dataset. Nonetheless, general-purpose notebooks may be inadequate, since users may have distinct information needs stemming from diverse analysis tasks. To address this, autogenerated notebooks must be far more personalized.

To this end, we introduce LINX, a *Language-driven generative data exploration system*. Given a dataset and an analytical task description in natural language, LINX generates a *personalized* exploratory notebook tailored to the user's specified task.

In the notebook composition process, LINX utilizes Large Models (LLMs), renowned for their exceptional generative capabilities, such as writing code snippets and even processing small data amounts [42–44]. However, we make an important observation: LLMs cannot generate optimal exploration notebooks *directly*, because *they lack the ability to comprehensively interact with the data itself*. Correspondingly, as we assert through our experimental evaluation, notebooks composed directly by ChatGPT yield notably fewer relevant insights compared to those generated by LINX. This deficiency arises because meaningful insights are often discovered through continuous refinement of the analytical process, and examination of numerous exploratory paths until discoveries are made.

LINX overcomes this limitation of LLMs by segregating the generative process into *planning* and *execution*. We first harness the LLM's "analytical mind" to deduce a set of specifications for the exploratory process based on the user's definition of their analytical task. The exploratory specifications and dataset are then given to a data-driven, Constrained Deep Reinforcement Learning (CDRL) engine, that can actually interact with the input dataset and observe the output of its own analytical operations. In an intelligent trial-and-error manner, the CDRL engine discovers a useful exploration notebook that complies with the user's analytical task.

Correspondingly, we face two major research challenges:

C1. Deriving an exploration plan using an LLM. An ideal exploration notebook exhibits a sequence of semantically connected analytical operations, while maintaining a coherent narrative and

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

structure [28]. Whether crafted by a human analyst or an AI agent, the process involves iterative interactions with the data, uncovering interesting subsets or aggregations through trial and error, and subsequently exploring them in successive operations.

The primary challenge lies in determining which elements in the desired exploration notebook can be immediately derived from the user's task description and which should be discovered in a data-driven manner.

We address this challenge by, first, introducing a specification language for data exploration processes (LDX). LDX allows specifying desired fragments in the output sequence of analytical operations, shaping its structure, and crucially, defining how to contextually link individual operations. However, directly deriving LDX specifications presents an additional challenge for an LLM, primarily due to the absence of pertinent information in its training data. We tackle this issue by employing a two-stage chained prompt approach: In the first stage, we generate template Python Pandas [43] code, a format highly prevalent in LLM training sources such as Stackoverflow and Github[43]. The resulting code template is then translated to LDX through a subsequent prompt.

C2. Guiding a DRL Agent to Produce *Personalized* Exploratory Notebooks Using the Exploration Plan. In our prior work [5], we demonstrated that generating useful analytical sessions (although task-agnostic) is a challenging computational task but feasible with a dedicated deep reinforcement learning scheme. In this approach, a neural-network-based agent iteratively interacts with the input dataset to uncover valuable exploration paths. However, generating useful notebooks that are compliant with the input exploration plan is highly non-trivial, given that only a small fraction of the vast space of possible exploratory paths adheres to the specifications. This poses a reward-sparsity problem, a well-known challenge in reinforcement learning [37].

To address this challenge, we draw inspiration from prior work in *constrained deep reinforcement learning* (CDRL) [10, 57], where the neural network agent is specifically designed to handle additional requirements, such as safety constraints [20]. In LINX, both the dataset and the LDX exploration specifications are provided as input to the CDRL engine. Our CDRL engine is built on top of the generic exploration framework of [5], augmenting it with two main novel components: (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 compliant operations with a higher probability.

To empirically assess LINX, we conducted an extensive set of experiments. Initially, we present results using a new benchmark dataset comprising 182 analytical tasks and corresponding LDX plans. Our LLM-based plan generator demonstrates high performance, successfully generating correct exploration plans even when both the analytical task and dataset are unseen in the prompts.

Next, we carried out a comprehensive user study involving 30 participants to evaluate the relevance and overall quality of LINX 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 but also significantly aid users in completing analytical tasks, providing 3-5 times more relevant insights. Finally, we examined the effectiveness of LINX's CDRL engine, demonstrating that the convergence and performance of LINX are comparable to ATENA [5]. This is noteworthy, considering LINX's more complex reward scheme and neural network architecture, used for handling the custom exploration specifications.

A recent demo paper [35] briefly introduces LDX and the CDRL engine, but with a focus on a web interface for manual specification composition (rather than a description of the task, as in LINX).

Paper Outline. We begin with a brief overview and illustration of our system workflow (§2). We then introduce the LDX specification language and its corresponding verification engine (§3). Next, we describe the exploration plan generator (§4) and our CDRL engine (§5). Subsequently, we detail our experimental results (§6), review related work (§7), and conclude with final remarks (§8).

2 LINX WORKFLOW ILLUSTRATION

In essence, LINX takes a dataset and task description as input, forwarding them first to the *Exploration Plan Generator*. As highlighted in the introduction, we leverage the LLM's capability to "plan" an exploration scheme and overcome its inherent limitation in performing data-driven exploration using our *CDRL engine*. The CDRL agent is hooked to the dataset, executing thousands of interim analytical operations until it converges to a comprehensive and effective exploration process, which is then presented to the user in a notebook format.

Figure 1 illustrates the detailed architecture and an example workflow of LINX. On the left-hand side of the figure, the user provides a dataset and an analytical task in natural language: *"Find a country with different viewing habits than the rest of the world, using the Netflix Movies and TV Shows Dataset"*. This is a compound analytical task that cannot be answered via a single analytical operation or an SQL query. The task requires both to *discover* such a country that deviates with respect to several attributes in the data, then to build an adequate exploratory path that showcases the insights—such as aggregating on the deviating attributes and comparing between the atypical country and the rest of the world.

We next exemplify how LINX initially employs the *exploration plan generator* to establish the "skeleton" of the exploratory process that can accommodate a variety of compatible exploration paths. In the second step, our *CDRL engine* takes both the dataset and the plan as input, navigating through the available paths to discover the most effective one.

(1) From dataset and task description to an LDX exploration plan. The LINX *exploration plan generator* derives that composing an exploratory notebook for discovering an atypical country in the data should include a comparison of aggregated attributes (via group-by operations) when filtering in on a country, then out. However, determining *which* country to apply the filter to, and what exact aggregations to use, cannot be derived directly from the task description, but must be discovered in a data-driven process.

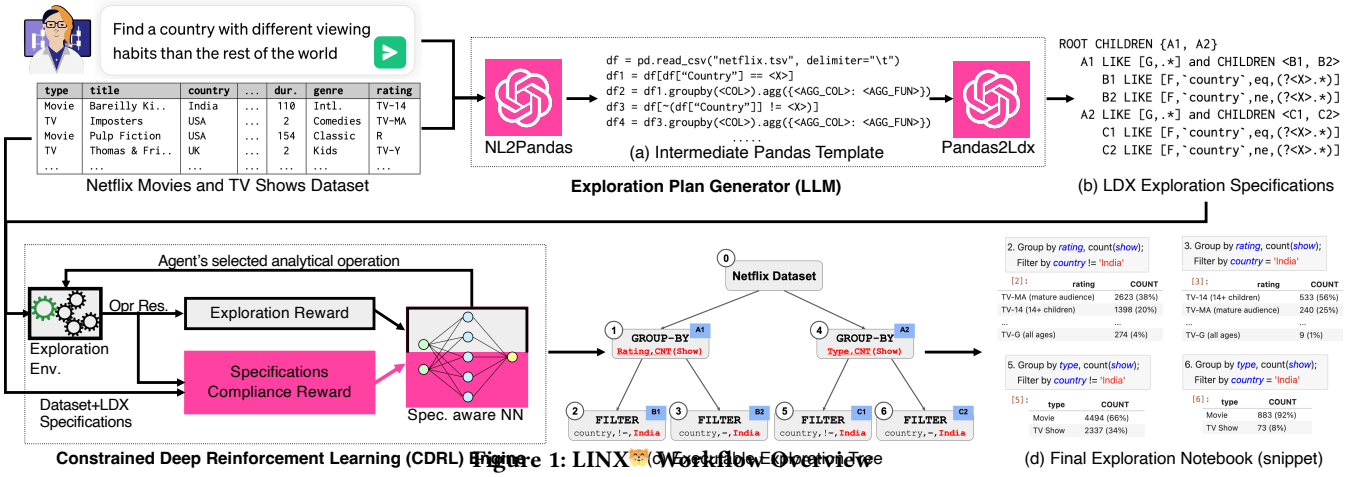

Figure 1: LINX Workflow Overview

As deriving specifications directly in LDX poses a challenge for an LLM, given the absence of such knowledge in its training data, we employ a two-stage prompt solution. Firstly, we prompt the LLM to generate an *Intermediate Pandas Template Code*, as depicted in Fig. 1a. The resulting code template is deliberately non-executable and includes special placeholders (marked with <>) representing elements in the notebooks to be discovered in a data-driven manner, such as the country value and the group-by aggregation fields and functions. Subsequently, the intermediate Pandas template is translated into formal LDX specifications, also by the LLM, as illustrated in Fig. 1b.

The output LDX specifications are used to guide the LINX CDRL engine, which uses the LDX verification engine in its reward scheme, to effectively assess the generated output notebook compliance.

(2) From dataset and LDX specifications to a full-fledged exploration notebook. LINX CDRL engine is illustrated in the bottom left side of Fig. 1. It takes the dataset and the LLM-generated LDX plan, and intelligently employs sequences of filter, group-by, and aggregate operations on the data to fulfill the input requirements. Our CDRL framework, detailed in Section 5, utilizes the basic data exploration DRL environment and exploration reward from [5]. It further augments the basic architecture with a *compliance reward scheme* that employs the verification engine of LDX to gradually guide the agent in performing exploratory sessions that adhere to the input specifications. 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.

Finally, the CDRL engine converges and produces an *executable exploration tree* (Fig.1c), consisting of an explicit sequence of analytical operations that adhere to the input specifications. Each node in the tree corresponds to an individual analytical operation specified in the LDX plan (the blue labels in the figure correspond to the named specifications in the LDX query), with each operation being executed on the result set of its parent node. The numbers indicate the order of execution. 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 tree is then presented to the user in a notebook interface. Fig.1d illustrates a snippet of the notebook, displaying the results of operations 2, 3, 5, and 6. The notebook snippet demonstrates that these exploratory steps indeed reveal 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 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).*

3 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 5).

3.1 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* [28, 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.

Composing LDX queries. Following [5, 39] we assume a tree-based exploration model, s.t. each query operation is a node in the exploration tree, and is applied on the results of a parent operation. The "root" is the original dataset before any operation is applied. An example exploration tree is illustrated in Figure 1c. LINX currently supports filter group-by, and aggregate operations using the following syntax: A filter operation is generally specified by `[F, attr, op, term]` and a group-by via `[G, g_attr, agg_func, agg_attr]`. These operations are sufficient for "slicing and dicing" of a dataset, leading to useful exploratory notebooks as we show in our experimental evaluation 6.3. We thus leave the support of more complex operations (such as join, union, pivot), to future work.

Our specification language LDX extends Tregex [32], 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 [51].

Example 3.1. 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 represent 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 5). 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>.*)]`' – 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.

3.2 LDX Compliance Verification

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 further assume that T is formatted as a tree, in which the root node is the full dataset before any query operation is employed. We first define an LDX assignment, then describe our verification procedure that searches for valid assignments.

Definition 3.2 (LDX Assignment). Given an LDX query Q_X and an exploration session 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

Algorithm 1: LDX Query Compliance Verification

```
VerifyLDX (T, S, A = ⟨φ_V = {ROOT:0}, φ_C = ∅⟩)   // Inputs: Exploration tree T, LDX Specifications list S, assignment A
1   if S = ∅ then return True
2   s ← S.pop()                          // get a single LDX specification
3   for c ∈ Cont(s) do                   // Assign continuity vars in s
4       if c ∈ φ_C then s.c ← φ_C(c)
5   V_T^s ← GetTregexNodeMatches(s, T, φ_V)
6   for v ∈ V_T^s do
7       φ_V^s ← φ_V ∪ {Node(s) : v}, φ_C^s ← φ_C
8       for c ∈ Cont(s) do               // Update continuity mapping
9           φ_C^s(c) ← v.c
10      if VerifyLDX(T, S, ⟨φ_V^s, φ_C^s⟩) then
11          return True
12  return False
```

$n \in Nodes(Q_X)$ 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 $\phi_V(\texttt{ROOT}) = 0$, 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 S, s.t. each specification $s \in S$ 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* [63] to obtain an initial matching between the nodes in $Nodes(S)$ to the nodes in $V(T)$. We then maintain the continuity mapping ϕ_C, and dynamically update the specifications in S 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 [63], returns all valid node matches for $Node(s)$, denoted V_T^s, given the current state of the node mapping ϕ_V. Then, for each valid node $v \in V_T^s$, we first update the node mapping ϕ_V (Line 7) and the continuity mapping ϕ_C (Lines 7-9): we assign each continuity variable c the concrete value of c from v, denoted $v.c$. (Recall that v satisfies s, *including the mapping* ϕ_C, therefore only unassigned variables in $Cont(s)$ are updated.) Once both mappings are updated (denoted ϕ_V^s and ϕ_C^s), we make a recursive call to *VerifyLDX* (Line 10), now with the shorter specifications list S (after popping out s) and the new mappings (ϕ_V^s, ϕ_C^s). Finally, the recursion stops in case there is no valid assignment (Line 12) or when the specification list S is finally empty (Line 1).

In Section 5 we describe how the LDX verification algorithm is used in our CDRL engine of LINX.

4 EXPLORATION PLAN GENERATION

As previously mentioned, when given a task description and dataset, the primary challenges in generating an exploration plan are (1) determining which elements in the output notebook can be derived apriori, leaving the remainder for data-driven discovery, and (2) expressing the plan using a syntactically correct LDX query.

To address these challenges, we introduce an LLM-based component using in-context learning [12]. 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 7 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 [21, 74, 81, 84, 86] . 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 tasks [40, 72]. In this approach, several illustrative examples are provided to the LLM before soliciting task completion.

To further enhance the distillation of the exploration plan, we employ an *intermediate code representation.* Taking inspiration from analogous solutions for tasks such as arithmetic and commonsense reasoning [9, 19, 36, 83], 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 exploration plan as a non-executable, template Python Pandas [73] code. The template code (See Figure 1a) contains special placeholders representing the notebook elements to be discovered in a data-driven manner. In the second stage, an additional prompt instructs the LLM to translate the intermediate Pandas template into formal LDX specifications. We coin our approach *NL2Pd2LDX*.

Intuitively, as Pandas (Python) code is highly prevalent in LLM training sources, formulating the plan as a Pandas template is easier than in LDX. As we empirically show in Section 6.2, our two-stage approach exhibits superior generalization compared to a direct NL-to-LDX approach. It performs better when the dataset and/or tasks 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 6.2).

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 task alongside a small dataset sample. Each few-shot example in (2) comprises several steps: (a) example user task; (b) description of the example's corresponding dataset (e.g., `epic_games` in Fig. 2) and schema; (c) the correct Pandas template for the task; (d) an NL explanation of the output. Including dataset information is motivated by past work in text-to-SQL [7, 70].

Step (d) is influenced by the Chain-of-Thought (CoT) prompting paradigm, which has demonstrated enhanced performance in multi-step tasks [61, 71, 72, 80]. Following the CoT methodology, we incorporate an explanation for each few-shot example. We use

least-to-most prompting [87], 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 specify the user input (target task) 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 our exploration plan generator, we constructed a new NL-to-LDX dataset, consisting of 182 instances of analysis tasks and corresponding LDX specifications. We describe the dataset construction in §6.1 and present the experimental results in §6.2. The full versions of all of our prompts, including the NL-to-LDX baseline, are provided in [51].

5 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 in [4, 5, 48], reinforcement learning is a powerful paradigm for generating exploratory sessions. In a dedicated DRL *environment* for data exploration, a neural-network-based agent *interacts* with an input dataset, by repeatedly employing sequences of analytical operations – until it detects a high-*utility* sequence that highlights interesting aspects of the data (The exact utility definition of a data exploration session is described below).

At each step in a session, the agent intelligently chooses a query operation, obtains information about the results, then receives a positive or negative *reward*, derived from the utility notion for exploratory sessions. The goal of the agent is to learn how to obtain a *maximal cumulative reward*.

As mentioned above, extending this DRL framework to support custom user specifications, is highly nontrivial, as the space of high-utility *and* compliant notebooks is significantly narrower. Also, adding further requirements on an already complex DRL task, may cause the neural-network based agent to fail to converge at all. To overcome this challenge, we devise a novel CDRL engine, that includes both a flexible compliance reward scheme, that gradually guides the agent towards generating compliant sessions (§5.1), and a specification-aware neural network architecture, designed to encourage the agent to "explore" the space of compliant operations with a higher probability (§5.2).

Before introducing the new CDRL architecture of LINX, we first briefly survey the basic *DRL environment for data exploration*, and the *generic exploration reward* as described in [5].

Background. **DRL environment for data exploration.** The environment allows the agent to explore a dataset D by employing the following analytical operations:, *Filter*, *Group-by and Aggregate* operations, as well as an additional (3) *Back* command, allowing the agent to traverse back to a previous operation, and employ a new operation on its results. After employing an operation, the agent obtains an *observation vector* from the environment, which describes the operations' results using statistical meta data, e.g., distinct counts, number of nulls, etc (See [5] for more details).

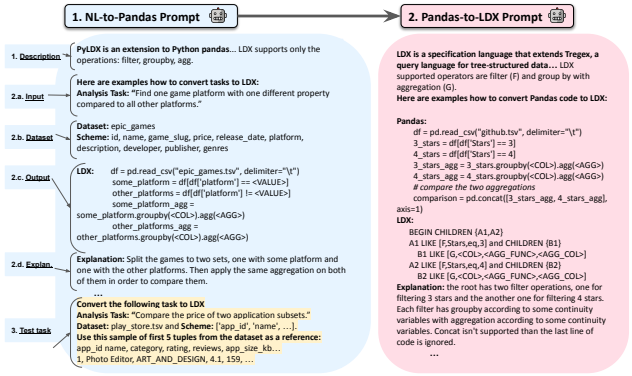

Figure 2: Examples of the two prompts used for NLPd2LDX: (1) NL to Pandas, and (2) Pandas to LDX

Generic Exploration Reward Signal. The exploration reward encourages the agent to perform a sequence of exploratory operations that are: (i) interesting - by employing data-driven notions of *interestingness*. For example, the interestingness of a filter query is measured using an *exceptionality* [68, 69] measure which favors filter operations that cause large deviation in value distributions; (ii) diverse from one another - by computing the Euclidean distance of the vector representation of resulted views; and (iii) coherent - by utilizing a weak-supervision-based classifier [50] that decides if an operation makes sense. (For example, a group-by on a numerical/textual attribute is often incoherent.)

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

End-of-Session Compliance Reward. Given the input dataset and *LDX* specifications, our reward scheme guides the agent towards a compliant exploratory session. Recognizing that *structural specifications should be learned first*, we prioritize correct session structure learning. 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.

Our end-of-session rewards works as follows. First, we use Algorithm 1 to check if T complies with LDX. 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 with *structural specifications* S_{struct} 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 gradual rewards to encourage satisfaction of operational specifications. Upon generating a fully compliant session, the agent receives a high positive reward, and can focus on optimizing the *generic exploration reward*.

Immediate (per-operation) Compliance Reward. To reinforce adherence to structural constraints, we introduce an *immediate* reward signal granted after each operation. This real-time signal negatively rewards operations that violate the structural specifications.

We use a modification of the LDX verification engine (Algorithm 1), that can operate on an *ongoing* session T_i (i.e., after i steps, with $N - i$ remaining steps) rather than a completed session. Intuitively, we assess the possibility of a *future* assignment satisfying structural constraints S_{struct} 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 (The number of possible completions throughout an N-size session is bounded by C_N, the Catalan number, as we show in our technical report [51]). In case no valid assignment is found to any of the new trees, a negative reward is granted.

5.2 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, 57], 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 and thus internally learn to perform compliant sessions.

This is achieved by extending ATENA's general-purpose architecture [5] with built-in placeholders for operation "snippets" that are derived from the LDX constraints. Figure 3 depicts the network architecture (The new, specifications-aware functionality is highlighted in pink). The input layer (left-hand-side) receives the observation vector, which describes 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. These layers allow to first choose an operation type (e.g., filter, group-by, back), and only then the required parameters, via "multi-softmax" segments. As in Figure 3, 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" (σsnp). 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'`, derived from our example LDX query in Figure 1b, 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 $s \in S_{opr}$.

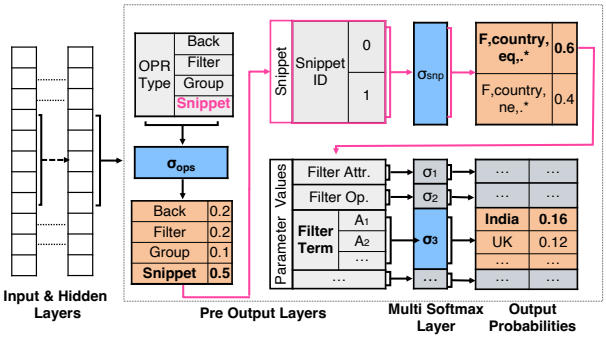

Figure 3: Specification-Aware Network Architecture

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 5.1. Figure 3 depicts an example selection process, in which the selection probabilities are already computed by the agent's neural network (see the right-hand side of the Output Probabilities segment, 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 6.4, allows LINX to consistently generate compliant exploration sessions in 100% of the datasets and LDX queries in our experiments.

6 EXPERIMENTS

We begin by presenting our experiments setup (§6.1) then describe our experiments, conducted along three key facets:

- **LLM-generated exploration plan evaluation (§6.2).** This dimension aims to gauge the performance of our LLM module in generating a correct set of exploration specifications, given an analytical task described in natural language.
- **Exploration notebook quality evaluation (§6.3).** In this experiment we examine the relevance and usefulness of exploratory notebooks generated by LINX, both via subjective assessment (rating questionnaire) and an objective one, where users use LINX to complete an analytical task. We compare the results of LINX against other baselines such as ChatGPT (direct) and ATENA [5], and Google Sheets Explorer.
- **CDRL Performance & ablation study (§6.4).** We study the efficacy of LINX-CDRL in generating specification-compliant notebooks, as well as its convergence and running times.

The full experiments code and data are provided in our Github repository [51].

	Analysis Meta Task	Example (concrete) Task	# Ex.
1	Identify an uncommon entity	T1: “Find an atypical country” (NETFLIX)	18
2	Examine a phenomenon (subset)	T2: “Examine characteristics of successful TV shows” (NETFLIX)	16
3	Discover contrasting subsets	T3: "Find three actors with contrasting traits” (NETFLIX)	22
4	Survey an attribute	T4: "Survey apps’ price” (PLAY STORE)	21
5	Describe an unusual subset	T5: "Highlight distinctive characteristics of summer-month flights” (FIGHTS)	27
6	Investigate various aspects of an attribute	T6: "Investigate reasons for delay” (FLIGHTS)	22
7	Explore through a subset	T8: "Analyze the dataset, with a focus on flights affected by weather-related delays” (FLIGHTS)	28
8	Highlight interesting sub-groups	T9: "Highlight interesting sub-groups of apps with at least 1M installs” (PLAY STORE)	28

Table 1: Experimented Analytical Templates for Multiple Datasets (Total 182 Instances)

6.1 Experimental Setup

Implementation. We implemented LINX in Python 3. The LDX verification engine partially uses the Tregex Python implementation [63], as discussed in Section 3.2). As for the CDRL Architecture, we use Pandas [38] for query operations, and the custom modifications to the neural network (§5.2) were built in ChainerRL [18]. All our experiments were run on a 24-core CPU server.

Datasets. We used three real-world datasets, each with a different schema and application domain: **(1) Netflix Movies and TV Shows Dataset [26]**, listing about 9K Netflix titles. Each title is described using 11 attributes such as the country of production, duration/num. of seasons, etc. **(2) Flight-delays Dataset [24]**, 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 [25]**, 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.

NL-to-LDX Exploration Plan Benchmark Dataset. We constructed a benchmark dataset comprising 182 pairs of analytical tasks and their corresponding LDX exploration plans. To do so, we examined the Kaggle notebooks collection associated with the datasets mentioned above, extracting a diverse set of analytical tasks. For each task, we then formulated an LDX exploration plan based on the content derived from the corresponding real-life exploration notebooks.

To increase the benchmark dataset size, we adopted the scheme outlined in Figure 4. Initially, we organized individual analytical tasks into eight distinct "meta-tasks," each characterized by a unique intent. Table 1 provides a list of meta-tasks along with an example of a concrete task. Then, we generated NL and LDX templates by excluding any elements tied to the data, such as attribute names, aggregative operations, and predicates defining data subsets. We populated both the NL and LDX templates by randomly incorporating values from our three datasets. For instance, the templates in Figure 4, associated with Meta-Task 7 (See Table 1) are populated using the `flight` domain, the `origin_airport` attribute, operator ≠,and the term `'BOS'` (the populated LDX template is omitted for brevity). Next, since the populated NL template may sound unnatural, we utilized an LLM-based paraphrasing approach (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 queries, that did not reflect a realistic user intent. Table 1 lists total number of instances for each meta task (See [51] for full details).

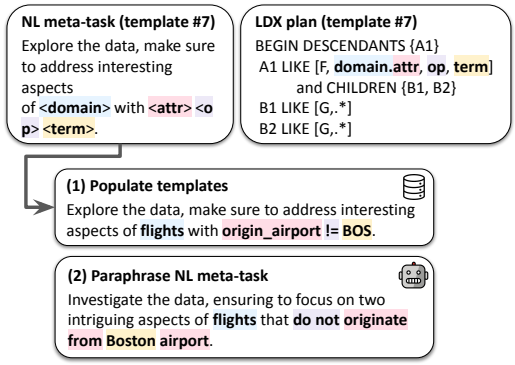

Figure 4: NL-to-LDX Benchmark Dataset Generation

6.2 Exploration Plan Performance Evaluation

We initiate the evaluation by assessing our exploration plan generator, designed to derive LDX exploration specifications based on a natural language description of an analytical task. Two key questions guide this evaluation:

- P1: How effectively does our LLM-based plan generator generalize to unseen tasks and datasets?
- P2: Is our NL2PD2LDX chained prompt solution boosts performance for both GPT-3.5 and GPT-4?

To address P1, we analyze the LDX generation performance in four experimental scenarios, varying whether the dataset or meta-tasks are seen or unseen in the few-shot prompt examples. For P2, we evaluate the performance of both LLMs using a direct NL2LDX approach with a single prompt, comparing it to our two-stage solution that involves pandas templates first.

Experimental Settings. We now detail the evaluation measures for the NL-to-LDX task and provide an overview of the different scenarios and baselines.

Evaluation Metrics. Evaluating text generation quality is a known challenge with various approaches [27, 78, 81]. For example, Text-to-SQL performance assessments often rely on query execution results [52, 58, 78], but this is unsuitable for LDX queries as they are non-executable. Alternative measures include exact string match [17, 81], and graph edit distance, commonly used for graph semantic parsing tasks [8, 27]. 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 [39], 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 [39]. This measure is a standard tree edit distance [82], incorporating a dedicated label distance function to assess the distinction between two parametric analytical operations (See [39] for full detail). To apply this metric, we construct a minimal tree for each compared LDX query, masking all unassigned parameters (including continuity variables). 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. As previously mentioned, to assess the generalization of our LLM-based exploration plan generator (Question P1), we conducted four distinct experimental scenarios involving the presence or absence of dataset and meta-tasks in the few-shot prompts. In each scenario, the generator receives a *test* analytical task 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-task*, the prompts, as described in Section 4, include few-shot examples over the same test dataset and meta-task associated with the test task (excluding the explicit test task itself). In the subsequent scenarios *(2) seen dataset, unseen meta-task* and *(3) unseen dataset, seen meta-task*, prompts include examples from the same dataset, excluding the associated meta-task, and vice versa. In the most challenging scenario *(4) unseen dataset and meta-tasks*, few-shot examples are provided from different datasets and different meta-tasks compared to the test task.

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 task (See [51]). We assessed the performance for both ChatGPT (gpt-3.5-turbo)[44] and GPT-4 [43], the current state-of-the-art LLMs.

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-task*, 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-task within the prompt allows for easy overfitting, reducing the need for an intermediary solution. In Scenario *(2) seen dataset, unseen meta-task*, 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-task*, 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-tasks. 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-task*, the chained solution yields higher scores for both LLMs, with GPT-4+PD slightly outperforming ChatGPT+PD.

Model\Settings	Seen Meta-Task		Unseen Meta-Task	
	lev^2	$xTed$	lev^2	$xTed$
Seen Database				
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 Database				
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: NL-to-LDX Performance Results

Summary. Our experimental results addressing questions P1 and P2 reveal that while superior performance is attained when both the dataset and meta-task are included in the few-shot examples set. Recall that even in these simpler settings, the LLM has never encountered the exact same task on the same or a different dataset. Remarkably, our chained-prompt solution improves results for both ChatGPT and GPT-4 in the more challenging, yet realistic scenarios where the dataset and meta-task are unknown in advance. This latter condition is particularly demanding, as the meta-task encompasses all tasks with a similar intent. Omitting these tasks from the prompt requires the LLM to genuinely generalize.

6.3 Output Notebook Quality Evaluation

We next evaluate the overall quality and relevance of the notebooks generated by LINX, compared to notebooks generated by the baselines. We answer the following two questions:

- U1: Do users find the notebooks generated by LINX better and more relevant compared to baselines such as ChatGPT [44] and ATENA [5]?
- U2: How well do LINX notebooks assist users in accomplishing their analytical tasks?

We performed a *subjective* evaluation to answer S1, in which users rated the notebook according to numerous criteria, and an *objective* evaluation, to answer S2, where we measured users' performance in completing analytical tasks via the notebooks.

Experiment Setup. We recruited a total of 30 users, all of whom are CS students or graduates that are familiar, to some degree, with

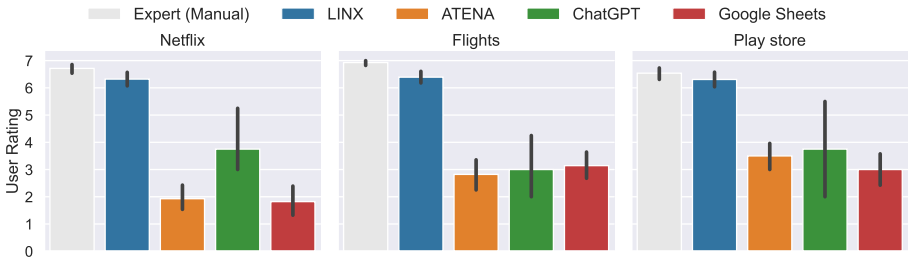

Figure 5: User Study – Rating of Relevance of the Analytical Session to the Given Tasks

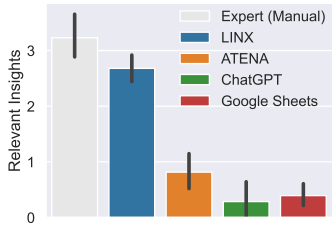

Figure 6: Avg. Num. Insights

Insight
"The ratio of movies-series in India is higher than the movies-series ratio anywhere else." (T1)
"Most multi-season US TV shows are dramas or comedies" (T2)
"About one-third of flights occur in summer, yet the monthly rate of delays remains consistent throughout the year." (T5)
"While long flights are not delayed often, if they are, this is mainly for a security reason." (T6)
"Apps with 1M installs are typically free, highly rated, and compatible with Android 4." (T9)

Table 3: Examples of Insights Derived by Users Using LINX

data analysis and science. We then extracted from our benchmark dataset 12 different analysis tasks and LDX specifications (4 different tasks for each dataset). See Table 1 for Tasks T1-T9 (T10-12 appear in [51] for space constraints).

We used LINX CDRL engine to generate an exploration notebook for each task and dataset, and evaluated them against the following baselines: **(1) Experts-Manual.** The baseline is designed to provide an "upper bound" for the output quality of the automatic approaches. We asked three experienced data scientists to freely explore the dataset then build a notebook of interesting query operations that are *relevant for the given task*. To ensure a fair comparison, we limited their usage to the same query operations currently supported in LINX (filter, group-by, and aggregate). **(2) ATENA [5].** We ran ATENA on each of the datasets. As it automatically generates an exploration session but does not accommodate user specifications, it produces the same exploration notebook for all four tasks of each dataset. **(3) ChatGPT (gpt-3.5-turbo) [44].** 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. **(4) Google Sheets Explorer [62].** 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.

The instructions, data, and output of all baselines are provided in our code repository [51].

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 for each 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 task?* (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 the tasks for each dataset (The vertical line depicts the .95 confidence interval.) Indeed, the notebooks manually prepared by experts obtained the highest average relevance scores of 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, it obtains inferior relevance scores as it lacks the ability to properly interact with the data. It therefore 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 tasks.

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.)

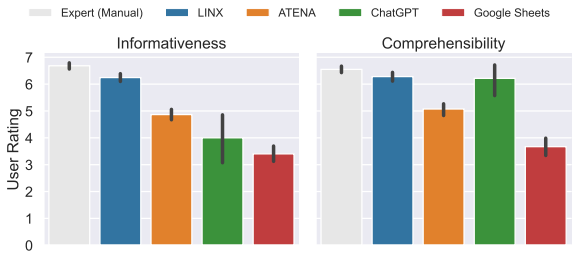

Figure 7: Informativeness & Comprehensibility Rating

The manually-prepared notebooks once again achieve the highest scores. Additionally, both ATENA and Google Sheets now attain

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

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 4.0.

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 there is no apparent trade-off between relevance to the task and the informativeness or comprehensibility of the output notebook, as *LINX generates exploration notebooks that are both highly relevant to the task, as well as informative and coherent.*

Objective Study (Task Completion Success). We also compared LINX with the baselines in an objective manner – by examining how well the exploratory notebooks assist users in completing their analysis tasks. To do so, we asked each participant who reviewed a notebook to further list all insights they were able to derive from it that are *relevant* to the corresponding analytical task. The correctness and relevance of insights were evaluated, for each task, by the same expert who constructed the manual notebook (Baseline 1), and is therefore highly familiar with the dataset and task.

Figure 6 shows the average number of task-relevant insights derived using each baseline. Using LINX, users derived an average of 2.7 relevant insights per task, which is second only to the manually-built notebooks by experts (3.2 insights). ATENA and Google Sheets are again far behind with an average of 0.8 and 0.4 relevant insights per task (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 (as 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 tasks.

Summary. Regarding questions U1 and U2, 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.

6.4 CDRL Performance & Ablation Study

Last, we examine the performance of our CDRL Engine. We answer the following two questions:

- PA1: Are all components of LINX CDRL engine necessary for consistently generating compliant notebooks?
- PA2: Does LINX-CDRL exhibit slower convergence than ATENA [5] due to its more complex reward signals and architecture?

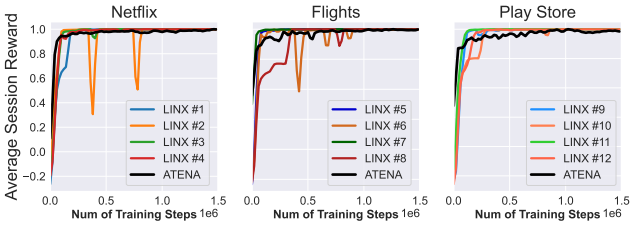

Figure 8: Convergence Comparison to ATENA

Ablation Study. To gauge the necessity in the components of LINX we compared it to the following system versions, each missing one or more components: **(1) Binary Reward Only** uses a binary end-of-session reward, based solely on the output of the LDX verification engine, without using our full reward scheme (§5.1 and specification-aware network (§5.2). Instead, it uses the basic neural network of [5]. **(2) Binary+Imm. Reward** uses the reward scheme, as described in Section 5.1, 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 [5].

We employed each baseline on the same 12 LDX queries used in the user study, and examined how many of the generated notebooks 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.

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

Finally, see that only the full version of LINX-CDRL, which uses both the full reward-scheme *and* the specification-aware neural network, is able to generate compliant sessions for 100% of the LDX queries. This shows that our adaptive network design, as described in Section 5.2, 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 [5], in order to assess whether its more complex architecture affects performance. Figure 8 depicts the convergence plots for each of the three experimental datasets and tasks T1-T12 (used in the user study). The convergence for each LDX query i (corresponding to Task Ti) is depicted using a line labeled *'LINX #i'*, where the black line in each figure depicts the convergence process of ATENA [5] 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 T2 and T6). 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) only the complete version of LINX-CDRL, incorporating the full compliance reward scheme and utilizing the neural network architecture, consistently generates compliant notebooks. (2) Despite its more complex reward system and neural network, the convergence and running times of LINX are on par with ATENA.

7 RELATED WORK

We survey several areas of related work:

Automated/assisted data analysis. Numerous previous works have been developed to aid manual analytical tasks in various ways. Examples include simplified analysis interfaces for non-programmers [30, 62], data preprocessing [6], visualization recommendations [59, 69], explanations [11, 33], insights extraction [64], and safe exploration [85]. While these works facilitate important aspects of the *interactive* analytical work, LINX focuses on generating complete exploration notebooks, known to be highly beneficial for analysts to use before exploring new datasets or tasks [28, 46, 53].

Suggesting analytical next steps [14, 16, 23, 39, 55, 79] has been explored in previous work, primarily focusing on a *single* step rather than a full sequence or notebook, as is done in LINX.

Closer to our work, Deep Reinforcement Learning (DRL)-based systems for automatic exploration have been proposed in the literature [5, 47, 48]. These systems can generate entire exploratory sessions without requiring training data. However, they lack support for user preferences as input, limiting their ability to generate personalized, task-driven sessions like LINX.

Visualization Specification Languages. Previous work has introduced both full and partial specification languages to facilitate the programmatic creation of data visualizations. Examples include Matplotlib [22] and Vega [76], supporting full specifications, and the declarative Vega-lite [56], which offers higher-level, simpler constructs. Closer to our work, visualization recommender systems like [31, 75, 77] enable users to provide partial specifications (using the same syntax as, e.g., Vega-lite) and receive recommended visualizations that meet the specified constraints.

While LINX differs significantly in nature and goals, our specification language LDX is also the first, to our knowledge, designed for specifying a full *sequence* of interconnected operations rather than a single desired object.

Text-to-SQL. Our exploration plan generator, which derives exploratory specifications in LDX from a description of an analytical task, aligns with research on mapping natural language (NL) requests to SQL queries (see [1, 29] for comprehensive Text-to-SQL surveys). Large supervised datasets have played a pivotal role in advancing this research, enabling the training of machine learning models for this task [34, 81, 86]. Recently, Text-to-SQL via LLMs [49] has shown promising results, nearly comparable to fine-tuned dedicated models. However, generating exploration plans poses a substantially different and arguably more challenging task. This is due to (1) the exploration plan comprising multiple analytical operations rather than just one SQL query; (2) the need to decide which exploratory elements can be explicitly derived from the task description and which should be mined in a data-driven manner, a novel task arising from our specific problem settings; and (3) since deriving exploration plans is a novel task, compared to writing SQL queries or data retrieval code, there is significantly less available training data for this task.

LLM Applications in Data Management The promising results demonstrated by LLMs such as GPT-3 [45] and LLaMA [66], paved the path to integrating LLMs in data management systems. Applications (beyond mapping text-to-SQL) include data integration [3], table discovery [13] and even the potential of substituting database query execution engines [41, 54, 65, 67].

8 CONCLUSION & FUTURE WORK

This paper introduces LINX, a language-driven exploration system designed for generating personalized and task-relevant exploration notebooks. Our findings highlight the limitations of LLMs in directly crafting exploration notebooks due to their inability to engage comprehensively with the data. LINX overcomes this challenge by utilizing LLMs for exploration plan generation and employing a novel CDRL engine for execution.

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 [31, 77]. Additionally, integrating LLMs within the reinforcement learning process, as proposed in [15], may greatly improve the quality of LINX notebooks by steering the agent towards more human-meaningful and plausible analytical operations.

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] 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

[4] 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.

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

[6] Sebastian Baunsgaard, Matthias Boehm, Ankit Chaudhary, Behrouz Derakhshan, Stefan Geißelsöder, Philipp M Grulich, Michael Hildebrand, Kevin Innerebner, Volker Markl, Claus Neubauer, et al. 2021. Exdra: Exploratory data science on federated raw data. In *Proceedings of the 2021 International Conference on Management of Data*. 2450–2463.

[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] 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.

[12] 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]

[13] 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

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

[15] Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. 2023. Guiding pretraining in reinforcement learning with large language models. *arXiv preprint arXiv:2302.06692* (2023).

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

[17] 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

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

[19] 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

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

[21] 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

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

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

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

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

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

[27] 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

[28] 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*.

[29] 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

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

[31] 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.

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

[33] 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.

[34] 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]

[35] 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.

[36] 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

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

[38] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, 51–56.

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

[40] 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]

[41] 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

[42] OpenAI. 2023. *Advanced Data Analysis with ChatGPT (Enterprise Version)*. https://help.openai.com/en/articles/8437071-advanced-data-analysis-chatgpt-enterprise-version

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

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

[45] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL]

[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] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *PVLDB* 11, 3 (2017).

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

[52] 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

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

[54] 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

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

[56] 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.

[57] 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).

[58] 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

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

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

[61] 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

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

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

[64] 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.

[65] 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

[66] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aur'elien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *ArXiv* abs/2302.13971 (2023).

[67] 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

[68] Matthijs van Leeuwen. 2010. Maximal exceptions with minimal descriptions. *Data Mining and Knowledge Discovery* 21, 2 (2010), 259–276.

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

[70] 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

[71] 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

[72] 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

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

[74] 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

[75] 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).

[76] 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).

[77] 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.

[78] 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

[79] 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.

[80] 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

[81] 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

[82] 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.

[83] 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

[84] 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]

[85] Zheguang Zhao, Lorenzo De Stefani, Emanuel Zgraggen, Carsten Binnig, Eli Upfal, and Tim Kraska. 2017. Controlling false discoveries during interactive data exploration. In *Proceedings of the 2017 acm international conference on management of data*. 527–540.

[86] 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

[87] 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 EXPERIMENTS

A.1 Comparison to Tregex

As mentioned above, LDX is based on Tregex [32], 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 Computation Times Discussion

As is the case for Tregex [60], 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 6.4 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[1].

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 task 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 task 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 S_X (from the LDX query Q_X), we first check (Line 1) whether the session T is compliant with *LDX* using Algorithm 1. In case T is compliant, a high positive reward is given to the agent (Line 2).

[1]For example, the mean session size in the exploratory sessions collection of [39] is 8.

Algorithm 2: End-of-Session Conditional Reward

Input: Exploration Tree T, LDX Specification List S_X
Output: Reward R

1 **if** ***VerifyLDX***(S_X, T) = *True* **then**
  // T is compliant with S_X
2 **return** *POS_REWARD*
3 $S_{struct} \leftarrow$ Structural specs of S
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 S
8 **return** $\max_{\phi_V \in \Phi_V}$ ***GetOprReward***(ϕ_V, S_{opr})

GetOprReward (ϕ_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** $reward$

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 [39]). 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 [**?**]. To further improve the procedure performances, in our implementation, we have started the immediate reward only after 3 steps of the agent. This way we could still encourage the agent to comply with the structural constraints, and we decreased the number of possible trees when $10 \leq N \leq 20$ by factor of 4-5.

B ADDITIONAL IMPLEMENTATION DETAILS

B.1 Prompt Implementation Details

NL-to-LDX. The NL-to-LDX prompt is structured the same as the NL-to-Pandas prompt as discussed in section §4: (1) NL-to-LDX task description; (2) a series of few-shot examples; (3) the test analysis task alongside a small dataset sample. Each few-shot example in (2) comprises several steps: (a) example user task; (b) description of the example's corresponding dataset and schema; (c) the correct LDX template for the task; (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-tasks (Table 1). For NL-to-Pandas and NL-to-LDX we added 6 initial examples of mapping basic constructs before moving on to the 8 template examples, and for Pandas-to-LDX we added 2 additional general examples.

B.2 xTed Evaluation Explanation

In section §6.2,it was previously mentioned that we construct a minimal tree for the compared LDX queries in order to apply them a tree distance metric. The conversion of LDX query to tree is generally straightforward, except for the 'DESCENDANTS' structural specification operator, since the distinction between DESCENDANTS and CHILDREN can't be expressed out-of-the-box. Our ad hoc approach for addressing that is setting the descendants as direct children in the converted tree (meaning the minimal specification-compliant tree) and adding 'children type' as an additional property of the action label. Additionally, we slightly modified the action distance function by penalizing variations in the 'children type' of the compared actions.

NL-to-LDX Prompt

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

Task: apply the same aggregation and groupby twice
LDX:

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

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

```
...
```

Figure 9: Example of the prompt used for NL2LDX