A Language and Architecture for On-Demand, Personalized Data Exploration Notebooks - Technical Report

Anonymous Author(s)

Abstract

One of the most effective methods for facilitating the process of exploring a dataset is to examine existing data exploration notebooks prepared by other data analysts or scientists. These notebooks contain curated sessions of contextually-related query operations that all together demonstrate interesting hypotheses and conjectures on the data. Unfortunately, *relevant* such notebooks, that had been prepared on the same dataset, and in light of the *same analysis task* – are often nonexistent or unavailable.

In this work we describe ANON-SYS, a framework for auto-generating personalized exploratory sessions. Using a novel specification language, users first describe their desired output notebook. Our language contains dedicated constructs for semantically connecting the future output queries. These specifications are then used as input for a Deep Reinforcement Learning (DRL) engine, which auto-generates the personalized notebook. Our DRL engine relies on an existing, general-purpose, DRL framework for data exploration. However, augmenting the generic framework with user specifications requires to overcome a difficult sparsity challenge, as only a small portion of the possible sessions may be compliant with the specifications. Inspired by solutions for constrained reinforcement learning, we devise a compound, flexible reward scheme as well as specification-aware neural network architecture. Our experimental evaluation shows that the combination of these components allow ANON-SYS to consistently generate interesting, personalized exploration sessions for various analysis tasks and datasets.

Table of Contents

List of Figures

I Introduction

Data exploration is an essential, yet highly challenging process performed by data scientists and analysts in order to examine a dataset and extract insights from it. When exploring a dataset, users perform a multitude of sequential analytical operations on the dataset at hand (e.g., filter, group-by, aggregations), searching for interesting data subsets and aggregative patterns that are relevant to their task/goal at hand.

One of the most effective and popular methods to gain a head start on this difficult process is to examine existing *data exploration notebooks* [32, 18, 28] – curated, illustrative exploratory sessions prepared by other data scientists, on the same dataset, uploaded to a collaborative online platform such as Kaggle, Google Colab, or Github. These notebooks illustrate a data exploration "narrative" [18] via a sequence sequence of selected, contextually related queries and their results, hereby demonstrating (and assessing) hypotheses and conjectures on the data [28].

Naturally, in many cases relevant exploration notebooks (in terms of both the dataset *and* the analysis task) are unavailable.

To account for the case of nonexistent notebooks, previous work suggest ATENA [3, 2], a system for auto-generating general-purpose exploration notebook for a given input dataset, using deep reinforcement learning (DRL). ATENA is shown to successfully generate notebooks that shows generally interesting and diverse aspects of the input dataset, in a coherent, easy to follow manner.

However, general-purpose exploration notebooks (who do not consider the specific task at hand) are suboptimal, since in practice, even when working on the same dataset users may have completely different information needs, derived from different analysis tasks. Therefore, each analysis task for a given dataset calls for a custom, personalized exploration notebook. Consider the following example:

Example 1.0.1. *Data Scientists Clarice and Dolly both work for a large media production company, and assigned to analyze the Netflix Movies and TV Shows dataset [41], which contains information about more than 8.8K different titles (See Figure 1.1). Clarice's assignment is to discover countries with different, atypical viewing habits, compared to the rest of the world (In order to gain ideas for expanding the company's international reach). Dolly, however, is assigned on a different task – investigate the properties of "successful" TV shows that have more than one season.*

Even if familiar with the dataset schema, these two analysis tasks still pose different challenges: Clarice needs to discover dataset attributes in which some countries are

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

Figure 1.1: Netflix Dataset Sample

significantly different from the rest of the world; whereas Dolly needs to find specific attributes and aggregations which largely characterize the subset of "successfull" TV shows. As the dataset is publicly available on the Kaggle [41] platform, it is accompanied by dozens of exploration notebooks (uploaded by other community members). Unfortunately, most notebooks are generic, showing nonspecific insights such as "most Netflix titles are originated in the US", or "the best-rated film on Netflix is Pulp Fiction". As shown in our experimental evaluation for numerous datasets and tasks, such general-purpose notebooks are not as helpful and useful as task-relevant ones. □

To this end, we present ANON-SYS, a language and architecture for auto-generating personalized, task-relevant exploration notebooks. With ANON-SYS, task-related preferences of the desired notebook are first described in our Specification Language for Data E*X*ploration (LDX). LDX allows for specifying desired properties of the session structure, operations, and more importantly – its *continuity*, which is used to tie together subsequent operations.

The dataset, together with the LDX specifications are passed to the DRL engine of ANON-SYS, built on top of [3], but uses a novel, dedicated reward scheme and neural network architecture. The ANON-SYS engine then generates an interesting exploration notebook – that complies with the input specifications.

Example 1.0.2. *Given her assignment of exploring atypical countries (as describe in Example 1.0.1), Clarice is interested in an exploration notebook that features some country (unknown to her at the moment), and compares that country to the rest of the world w.r.t. some attributes (also unknown). These information needs are expressed in LDX (as explained in the sequel) and passed to the ANON-SYS engine, along with the Netflix dataset.*

Figure 1.2e depict a part of the personalized exploration notebook generated for Clarice (Ignore for now the rest of Figure 1.2, which illustrates the full sessions generation process). The system chose to focus its output exploration on India, and "compare" it to the rest of the world using a count aggregation over the attributes rating and show type. These query operation clearly show interesting and relevant insights, demonstrating how India is different than the rest of the world: The rating comparison (Notebook Cells 2 and 3), shows that whereas in the rest of the world, the majority of titles are rated

TV-MA (i.e., 17+), most titles in India are rated TV-14 (14+). An additional relevant insight stems from the show type comparison (Cells 5 and 6) shows that in India, the vast majority of titles are movies (93%), whereas in the rest of the world, movies comprise only 66% of the titles (the rest being TV shows).

Dolly, as described in Example 1.0.1, is interested in successful TV shows, therefore needs (and obtains) a completely different exploration notebook, as depicted in Figure 1.2f. Her output notebook, generated by ANON-SYS, explores common interesting properties of TV shows with two seasons or more: It first examines the country attribute of the successful shows, demonstrating (see Result 3) that most successful TV shows are made in the US. It then focuses on US-based successful shows, and examines their genre attribute (Operation 5), revealing that (successful) American TV shows are largely comedies, which are, in turn (using an additional filter on the last subcategory, and group-by rating) mostly rated as 'mature audience' (Results 6). □

Challenges, Contribution & Outline We begin by a preliminary overview of the basic DRL environment for data exploration and model, then outline the architecture of ANON-SYS (§2). We then describe our solution in detail, in which we focus on the following challenges:

C1. Design an effective specification language for data exploration notebooks (§3). Exploratory sessions comprise a sequence of data operations (e.g., filter, group and aggregate). While trivial specifications may be used to restrict the session to a particular data subset, or focus on some of the attributes, the key challenge is to facilitate compound, *semantic* specifications for the desired session. Such specification should control the *continuity* of the exploration process, i.e., interconnection between its query operation. For example, we may want to specify that a *filter* operation should be employed on *the same attribute* as the previous *group-by* (as is the case for Cells 3 and 5 in the notebook of Figure 1.2f). Such specifications ensure that the exploration will have a *clear, continuous narrative and structure.*

To this end, we devise LDX, a novel specification language for data exploration sessions. LDX is based on TREGEX [21] – a standard query language for tree-structured data, but significantly improves it to support *continuity* definitions that semantically connect between query operations.

C2. Devise a specification-compliance reward function (§4). ANON-SYS is built on the basic DRL scheme for data exploration, first introduced in [3] (See Section 2 for an overview). In such scheme, a neural-network based agent interacts with a dataset (by employing filter/group-by operations), using a dedicated *data exploration environment*, which then *rewards* the agent based on the quality of the session. In our setting, however, the challenge is to *generate useful exploratory session that are compliant with the input specifications.*

The key difficulty stems from the fact that the space of all possible exploratory session is immense (i.e., *all* possible sequences of *all* query operations on the input dataset), yet only a small portion of this entire space contains specification-compliant sessions.

Hence, guiding the DRL model towards specification-compliant sessions introduces a reward-sparsity problem, a well known challenge in reinforcement learning [24]. To overcome this issue, we devise a novel reward scheme, which combines a binary (i.e., satisfying/not satisfying) signal with a flexible, fine-grain feedback. The reward scheme gradually encourages the agent first to generate sessions with the correct *structure* (i.e., order of operations), then to further adjust the query operations, via a dedicated, operations-based reward signal until all specifications are met.

C3. Develop a specifications-oriented neural network architecture (§5) As shown in our experiments, the compliance reward scheme alone is often insufficient for successfully generating fully compliant sessions. This is since at each exploratory step, only a small portion out of about 100K query operations, may comply with the specifications. Therefore the next challenge is to provide an additional, effective guidance for the DRL agent, towards compliant query operations.

To that end, inspired by previous work in *constrained reinforcement learning* [35, 4] we develop an adaptive neural-network architecture that derives its final structure from the LDX specifications. The output layer of the network, (used by the agent to select query operations) has built-in placeholders for operations "snippets" that are extracted from the LDX specifications. This network structure encourages the agent to "expolore" the space of compliant operations with a higher probability.

Simplified User Interface (§6). We implemented a template-based interface allowing users to construct LDX specifications in a simplified, graphical interface, and then to generate a corresponding analysis session.

Experiments (§7). To evaluate ANON-SYS, we performed an elaborate experimental study, using 12 different analytical tasks on three real-world datasets. We show that the specification-driven sessions generated by our system are considerably more useful and relevant to the task at hand, compared to ATENA as well as a commercial, ML-based exploration tools from Google Sheets [40].

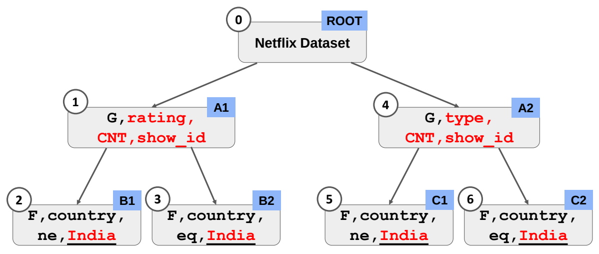
Related Work (§8). We provide an in-depth comparison of works in the field of automated/guided exploration [11, 27, 50, 17, 10, 33] to explicitly position our work. We also mention works in visualization specifications [48, 20] and adjacent DRL-based systems in the data research community [44, 23, 14].

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

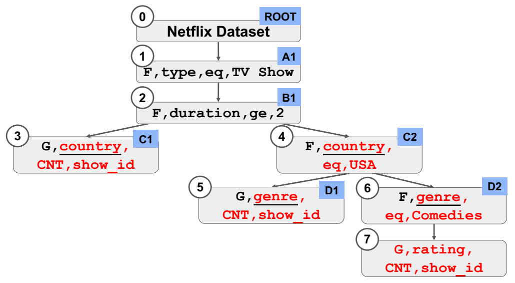
(a) Q_1: “Atypical Country” LDX Specifications

```
ROOT CHILDREN <A1>
  A1 LIKE [F,‘type‘,eq,"TV Show"] and CHILDREN <B1>
    B1 LIKE [F,‘duration‘,ge,2] and CHILDREN <C1, C2>
      C1 LIKE [G,(?<col1>.*),.*]
      C2 LIKE [F,(?<col1>.*),.*] and CHILDREN <D1, D2>
        D1 LIKE [G,(?<col2>.*),.*]
        D2 LIKE [F,(?<col2>.*),.*] and CHILDREN {*}
```

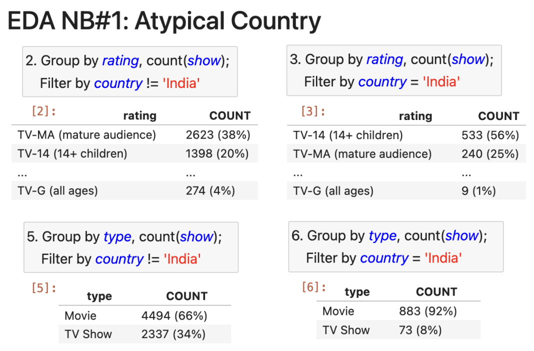
(b) Q_2: “Successful TV Shows” LDX Specifications



(c) “Atypical Country” Analysis Session



(d) “Successful TV Shows” Analysis Session



EDA NB#1: Atypical Country

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

[2]:

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

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

[3]:

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

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

[5]:

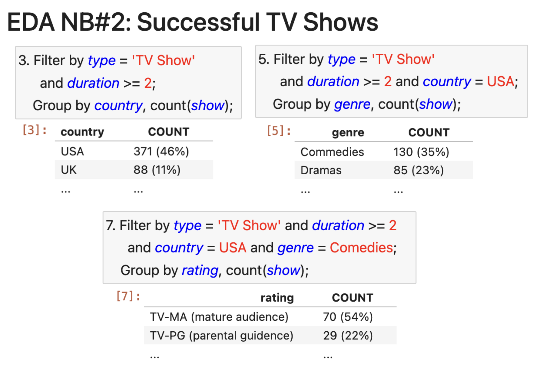
type	COUNT
Movie	4494 (66%)
TV Show	2337 (34%)

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

[6]:

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

(e) “Atypical Country” Notebook Snippet



EDA NB#2: Successful TV Shows

3. Filter by *type* = 'TV Show'
and *duration* >= 2;
Group by *country*, count(*show*);

[3]:

country	COUNT
USA	371 (46%)
UK	88 (11%)
...	...

5. Filter by *type* = 'TV Show'
and *duration* >= 2 and *country* = USA;
Group by *genre*, count(*show*);

[5]:

genre	COUNT
Commedies	130 (35%)
Dramas	85 (23%)
...	...

7. Filter by *type* = 'TV Show' and *duration* >= 2
and *country* = USA and *genre* = Comedies;
Group by *rating*, count(*show*);

[7]:

rating	COUNT
TV-MA (mature audience)	70 (54%)
TV-PG (parental guidence)	29 (22%)
...	...

(f) “Successful TV Shows” Notebook Snippet

Figure 1.2: **Examples of exploratory notebooks generation by ANON-SYS.** Each generation process fits a different analysis task on the netflix dataset. The “atypical country” task is first specified via LDX query (Figure 1.2a) that requests a comparison of “some” (unknown) country to the rest of the world w.r.t. two different attributes. The LDX query and dataset are passed to the ANON-SYS DRL engine, which then generates the full, compliant exploration tree (Figure 1.2c). The session is presented to the user in an exploration notebook (Figure 1.2e), s.t. each cell depicts a query operation and its results.The system chose to filter by *‘India’*, and focus on the attributes *‘Rating’* and *‘Type’*, showing, e.g., that. *In India, most titles are rated TV-14 while TV-MA (17+) is more common in the rest of the world* (See Notebook Cells 2 and 3). A similar process is done for the task of “successful TV shows” task of characterizing shows with more than one season (Figures 1.2b,1.2d, and 1.2f), revealing that *the majority of successful TV shows are Comedies created in the US* (See Cells 3 and 5 in the noteobok in Figure 1.2f).

II Preliminaries & Overview

We first briefly present the generic DRL framework for data exploration [3, 2] and the data model for exploratory sessions, and finally overview ANON-SYS.

2.1 DRL for Data Exploration

As suggested in [3, 2], in a DRL environment for data exploration, a neural-network based agent *interacts* with an input dataset, with the goal of finding a *good* sequence of analysis query operations. At each step in an exploratory session the agent chooses a query operation, obtained information about its result, as well as a positive or negative *reward*. The goal of the agent is learning how to obtain a maximal cumulative reward, by leveraging the experience gained from repeatedly interacting with the environment (without requiring any labeled training data).

We next describe the DRL environment, which controls the interaction of the agent with the data, the general-purpose reward signal (i.e., the notion of how "good" is the exploratory session) and the neural-network agent.

DRL environment for data exploration. The environment allows the agent to query a dataset D by employing commonly-used programmatic operations stemming from notebook-based data exploration [51]: (1) *Filter.* a filter operation is defined by `[F,attr,op,term]`, where `attr` is an attribute in the dataset, `op` is a comparison operator (e.g. $=, \geq, contains$) and `term` is a numerical/textual term to filter by. (2) *Group-by and Aggregate.* This operation is defined by `[G,g_attr,agg_func,agg_attr]`, i.e., grouping on attribute `g_attr`, aggregating on column `agg_attr` with aggregation function `agg_func`. Last, an additional (3) *Back* command is supported, allowing the agent to go back to previous results in order to start a new exploration path.

In ANON-SYS, as explained below, the agent is further given specifications in our dedicated language LDX, in order to generate personalized, task-relevant exploratory sessions.

General-purpose reward signal for exploratory sessions. The exploration reward encourages the agent to perform a session of exploratory operations that are: (i) interesting - by employing data-driven notions of *interestingness*. For example, the interestingness of filter is measured using an *exceptionality* [45, 46] measure which favors filter operations

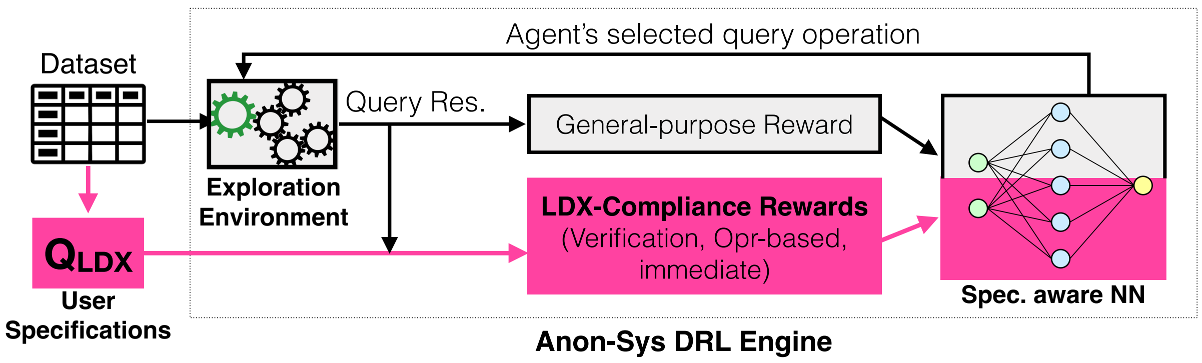


Figure 2.1: ANON-SYS Architecture

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 [31] that decides if an operation makes sense (for example, group-by on a numerical/textual attribute is often incoherent).

In ANON-SYS, we augment the general-purpose reward of ATENA with a novel specification-compliance reward scheme (See Section 4). By combining these signals, ANON-SYS is able to generate interesting yet *relevant* sessions to the task at hand.

Neural Network architecture To account for the large action-space (i.e., all possible exploratory operation), [3] describes the following architecture: Rather than using a single softmax output layer, it uses a two-fold architecture, allowing the agent to first choose an operation type (e.g., filter, group or back) and only then a value for each of the operation's parameters. This is done by a special "multi-softmax" layer, that first generates separate probability distributions for the high-level operation types, then additional probabilities for the corresponding parameters' values.

Unfortunately, this architecture alone is insufficient for reliably generating compliant exploratory sessions. Therefore, as explained below, ANON-SYS uses a novel, specification-aware architecture that derives its final structure from the LDX query.

2.2 Tree Model for Data Exploration Sessions

As mentioned above, using the *back* operation, the agent can start a new exploration path from previous, intermediate query results – inducing a tree-like exploration session. Following previous work [27, 26] we represent an exploratory session of N query operations $q_1, q_2, \ldots, q_N$ with a tree-based model as follows:

Definition 2.2.1 (Exploration Tree). *An exploration tree* $T = (r, V, E, \prec)$ *represents an exploration session* $q_1, q_2, \ldots q_N$ *as follows:*

1. *The root r is the raw dataset, before any operation is employed.*

2. *Each other node $v_i \in V$ represents a query operation q_i, operated on the results of its parent node.*

3. *Pre-order $\prec$ represents the order of execution, i.e. if $v_i \prec v_j$ then the corresponding operation q_i was executed before q_j.*

Two example exploration trees are depicted in Figures 1.2c and 1.2d.

2.3 System Overview

Figure 2.1 depicts the architecture of ANON-SYS. As ANON-SYS is partially based on ATENA, the pink elements in the figure mark the novel components of ANON-SYS.

The input to the system is a dataset, and a set of specifications expressed in LDX, our dedicated language for exploration session specifications. LDX, as depicted in Section 3, can express syntactic as well as semantic connections between the operations of the desired session. Then, the dataset and LDX specifications are passed to ANON-SYS, which employs a DRL solution in order to generate a personalized session. ANON-SYS uses the DRL environment for data exploration suggested in [3], but augments it with a new set of reward signals, which are used to particularly enforce that the agent generates sessions that comply with the input specifications. As mentioned in the introduction, ANON-SYS uses a flexible compliance reward scheme which is necessary to overcome the induced reward-sparsity problem. The interaction with the DRL exploration environment as well as the learning process is performed by a dedicated neural network. We extend the basic architecture suggested in [3] and design a specification-aware architecture, which modifies its structure w.r.t. the input specifications. This allows the selection of compliant operation with higher probability than the rest of the actions, as described in details in Section 5. Finally, the personalized exploration sessions are presented to the users in a notebook interface, as illustrated in Figure 1.2e and 1.2f.

III LDX: Specification Language for Data Exploration

Recall that a data exploration notebook is composed of a sequence of query operations. While the operations themselves can be specified in structured languages (e.g., SQL, Pandas), a trivial concatenation of such queries is inadequate for specifying the desired properties of the session. This is because, as mentioned above, *a key element in a useful exploratory notebook is its narrative [18, 28] – the contextual connection between the queries, organized in a logical, coherent manner.*

To this end, we devise LDX, a specification language for data exploration. Considering the tree-based model of an exploration session (Definition 2.2.1), LDX is based on Tregex [21], a query language for tree-structured data. Tregex natively allows partially specifying structural properties of the tree, which in our context correspond to the structural connection between query operations, as well as properties of the nodes' labels (query operations, in our context).

Then, to further consider the *semantic* connection between query operations, LDX extends Tregex by featuring *continuity variables.* These variables are used to enforce a match between different operations' parameters, without explicitly stating them. For example, after a group-by operation, 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.

Composing LDX queries The basic unit in LDX is a *single node specification*, which addresses the structural and operational preferences w.r.t. a single query operation q. Then, a full LDX query is composed by conjuncting multiple single-node specifications, interconnected using the LDX-proprietary *continuity variables.* See Table 3.1 for example expressions.

Single node specifications. Operational and structural specification are given for a single *named* node. Operational specifications are defined using a regex-like syntax via the `LIKE` operator. For example, the specification `'A LIKE [G,'country',SUM|AVG, *]'` indicates that the named-node '`A`' should be a group-by on the attribute *country*, showing either *sum* or *average* of *some* attribute (marked with $*$).

Structural specifications connects the named node to other nodes in the tree. This is done by combining regular expressions together with tree-structure primitives such as `CHILDREN`, `DESCENDANTS`, and `SIBLINGS`. For instance, `'A CHILDREN <B,+>'` states that

Expression	Type	Description
A CHILDREN <B,C>	Structure	B and C are the only (ordered) children of A
A SIBLINGS {B,C,*}	Structure	B and C are siblings of A (unordered), and there may be more, unnamed ones
A DESCENDANTS {B,C *}	Structure	B and C are two of the (unordered) descendants of A
A LIKE [G,.*,AVERAGE,.*]	Operational	A is a group-by operation on *some* column employing *average* on *some* column
A LIKE [F,category,eq|ne,.*comedy.*]	Operational	A is an equality/inequality filter on 'category', where the filter term includes the string 'comedy'
A LIKE [F,(?<col>.*),.*] B LIKE [G,(?<col>.*),.*]	Continuity	A is a filter operation on *some* column, and B is group-by on the *same* column
A LIKE [G,.*,(?<func>.*),(?<col>.*)] B LIKE [G,.*,(?<func>.*),(?<col>.*)]	Continuity	A and B are group-by operations with the same aggregation function and column
A LIKE [F,(?<col>.*delay.*),ge,(?<term>[0-9]{3,4}) B LIKE [F,(?<col>.*),le,(?<term>.*)]	Continuity	A is a filter operation on *some* column that includes the word 'delay' greater equal *some* value between 100 to 1000 and B is the same action but with lower equal

Table 3.1: LDX Expression Examples

the named-node `A` has a (named) child `B` and at least one more unnamed children. Recall from Definition 2.2.1 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 `B` is a named-node, it can take its own set of structural/operational specifications, and be connected to other named-node via the continuity variables, as described next.

Continuity Variables. Structural and operational specifications allow to *explicitly* constrain the operations' parameters, input data and order of execution. We next introduce the continuity variables in LDX, which allows constructing more complex specifications that *semantically* connect between operations' *unspecified* parameters.

LDX allows this using named-groups [1] 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,.*]`' specifies that the operation 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,(?<CNTRY>.*)]`' – in which the free filter term (`.*`) is captured into the variable `CNTRY`. Using this variable in subsequent operation specifications will restrict them to the same filter term (despite the fact that the term is not explicitly specified). For instance, as shown in Figure 1.2a, the subsequent specification is '`B2 LIKE [F,'country',neq,(?<CNTRY>.*)]`', indicating that the next filter should focus on all *other* countries than the one specified in the previous query operation. Refer to Table 3.1 for additional use cases of continuity variables.

We next describe two example full LDX queries, the ones used by ANON-SYS to generate the two notebooks in Figure 1.2e and 1.2f on the Netflix dataset.

Example LDX queries and output exploration notebooks We can now describe the LDX queries, composed to fit the exploration needs of "atypical country" and "successful TV shows", as described in our running example.

Q_1**: "Atypical country"**. The information needs from the "atypical country" notebook is to point out a specific country that shows different trends or patterns compared to the rest of the world. The corresponding LDX query Q_1 is depicted in Figure 1.2a. Q_1 is composed of two sub-trees, `A1,B1,B2` (Lines 2-4) and `A2,C1,C2` (Lines 5-7) with identical specifications. Each sub-tree forms a "comparison" of a country to the rest of the world, w.r.t. a dataset attribute. The first sub-tree is specified as follows: `A1` is a a group-by operation (with unspecified parameters) with two immediate children `B1` and `B2` (Line 2). `B1` and `B2` specify two subsequent filter operations on the attribute *'country'*, where the unspecified filter term is captured by the continuity variable `CNTRY` (Lines 3-4). This enforces that both `B1` and `B2` operate on the same term, while `B1` filters *in* and `B2` filters *out*. The rest of the query (Lines 5-7) specifies the sub-tree `A2,C1,C2` which have the same specifications as `A1,B1,B2`.

The ANON-SYS engine then auto-generates an exploration session that complies to Q_1 but also optimizes the general-purpose exploration reward (recall from Section 2.1), ensuring the final query operations selected by ANON-SYS are interesting, diverse and coherent. The resulted exploration tree generated by ANON-SYS is depicted in Figure 1.2c. The black parts of each operation in the tree are derived from the specifications of Q_1, whereas the red parts are inferred by the ANON-SYS engine. The square in the upper-right corner points the matching named-node from the corresponding LDX query. See that the system chose to focus its output exploration on *India* (via the continuity variable `CNTRY`) and "compare" it to the rest of the world using a *count* aggregation over the attributes *rating* and *show type*. The corresponding exploration notebook (which illustrates the results of each operation), as depicted in Figure 1.2e, highlights several interesting and relevant insights (as discussed in Example 1.0.2).

Q_2**: "Successful TV-Shows".** Recall that the requirement from the second exploratory session is to explore interesting properties of TV shows with more than one season. This reflects in the corresponding LDX query Q_2 (Figure 1.2b), which first filters the Netflix data on TV-shows with at least two season (Lines 2-3), then specifies two pairs of sibling group-by and filter operations (`C1`,`C2`, and `D1`, `D2`. Each such pair is semantically connected using a continuity variable: `COL1` enforces that the group-by attribute in `C1` is the same as in the subsequent filter `C2` (`COL2` enforces the same for `D1` and `D2`).

The resulted session is depicted in Figure 1.2d. The system first groups the successful TV shows by *country* (Operation 3), then filters on 'country=*US*' (Operation 4). The results (Successful TV shows from the US) are then group-by 'genre' (Operation 5), and further filtered by '*genre=Comedies*' (Operation 6). Last, the results of operation 6 (depicting now successful US comedies) are grouped by *'rating'*. The exploration notebook illustrates the results of Operations 3,5 and 7, as discussed in Example 1.0.2.

Discussion We conclude this section with two remarks.

Comparison to Tregex. As mentioned above, LDX is based on Tregex [21], 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.

Simplified query interfaces on top of LDX. Like Tregex, LDX is not intended for direct usage by novice users, as it requires some basic skills in structured language (e.g., SQL, Pandas). However, reaching a broader audience can be done by developing high-level, simplified interfaces on top of LDX. For example, instead of composing LDX queries from scratch, a simplified interface can utilize LDX templates (e.g., for *comparative* sessions, *investigative* sessions), and let the users fill the parameters of their choosing. Similar high-level interfaces have been developed for data visualizations, as detailed in Section 8. We have built a preliminary, template-based interface on top of LDX, see Section 6 for more details. We leave the design of declarative languages and Natural Language interfaces for future work (as discussed in the conclusion section).

IV LDX-Compliance Reward Scheme

Our goal is to encourage the DRL agent to generate exploratory sessions that not only adhere to the general-purpose reward (as explained in Section 2.1), but are also fully compliant with the input LDX query.

A naive solution to this problem would be to add, on top of the general-purpose reward, an additional binary *compliance* reward – providing a positive value in case the agent's exploration session satisfies the LDX specifications, and 0 otherwise.

However, similarly to board games environments like Chess and Go, in which a binary reward is only granted at the end of the "game", such a binary, delayed reward creates a well-known *sparsity* problem [24]: The proportion of the "winning" states compared to all possible states is very low, hereby the (mostly negative) feedback is insufficient to learn an optimal policy. Indeed, in our experiments (See Section 7.2), we show that when the agent is only provided with a binary LDX-compliance reward – *it does not converge on any of the LDX queries and datasets.*

To that end, we devise a compound, flexible reward scheme which gradually provides the agent a different signal, depending on its "progress" in satisfying the LDX specifications. At the end of the session, a different reward method is used, when the session is: (a) fully compliant with *all* specifications, (b) compliant with the *structural* specifications but not all operational specifications, and (c) not compliant with the structural specifications. Then, we further extend the end-of-session reward with an immediate, per-operation reward, that penalizes *specific* operations that violate the structural specifications, in real time. We show in Section 7.2 that all components of our reward scheme are essential for ANON-SYS to consistently generate LDX-compliant sessions.

In what comes next, we first describe the basic LDX compliance-verification algorithm (§4.1) then explain how it is augmented in the overall compliance-reward scheme (§4.2).

4.1 LDX Compliance Verification

Given an LDX query Q_X and an exploratory session tree T, our reward procedure, as depicted in Algorithm 2, first checks whether exists a valid *assignment* between Q_X and T. If such assignment exists, than T is compliant with Q_X. Unfortunately, the Tregex query evaluation engine [42] cannot be used directly to verify LDX compliance, since it is

Algorithm 1: LDX Query Compliance Verification

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

not compatible with its continuity variables. We therefore utilize Tregex's internal node-matching function, as described below, and develop a recursive procedure that verifies that an exploration tree satisfies both the LDX structure and operational specifications, as well as respects the continuity definitions.

In what comes next, we first define an LDX assignment (containing mappings for the query nodes as well as the continuity variables) then describe our verification procedure that finds full valid assignments.

For an input LDX query Q_X, we denote the set of its named nodes by $Nodes(Q_X)$, and the set of its continuity variables by $Cont(Q_X)$. An LDX assignment is then defined as follows:

Definition 4.1.1 (LDX Assignment)**.** *Given a 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 $n \in Nodes(Q_X)$ a node $v \in V(T)$ in the exploration tree 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 specifications (which stands for the dataset before any query operation is employed, as defined in Section 2.2) to the root node of the exploration tree T.

Example 4.1.2. *A possible assignment $A = \langle \phi_V, \phi_C, \rangle$ for Q_1 (the LDX query in Figure 1.2a) and the explorations tree from Figure 1.2c (nodes are numbered from 0 to 6), is given by: $\phi_V = \{\texttt{ROOT}: 0, \texttt{A1}: 1, \texttt{A2}: 4, \texttt{B1}: 2, \texttt{B2}: 3, \texttt{C1}: 5, \texttt{C2}: 6\}$ and $\phi_C = \{\texttt{Country}: \text{'India'}\}$.*

See, intuitively, that this assignment is valid, i.e., the exploration tree satisfies all specifications in Q_1, and consistently follows the continuity mapping.

We next explain how we find valid assignments for a given LDX query. We consider a 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 $Cont(s)$.

In a nutshell, our verification procedure, as depicted in Algorithm 1, gradually builds valid assignments. Recall that since LDX contains continuity variables, the Tregex engine cannot be used directly for query verification. Therefore, In addition to matching the nodes in $Nodes(S)$ to the nodes in $V(T)$ (which is done via the Tregex engine), our LDX verification procedure maintains the continuity mapping ϕ_C. ϕ_C is used to dynamically update the specifications in S with concrete values for their continuity variables. Then, in turn, the mapping ϕ_C is updated according to the valid node matches in ϕ_V (which contain concrete operations).

In more detail, the verification procedure *VerifyLDX*, as depicted in Algorithm 1, takes as input a LDX specifications list S, an exploration tree T, and the initial assignment A, where ϕ_V contains the trivial root mapping (as described above) and an empty continuity mapping ϕ_C. 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 , when all available continuity variables are updated in s, we use the Tregex internal method *GetTregexNodeMatch* (as in [42]). The method returns all valid node matches for $Node(s)$, denoted V_T^s, given the current state of the node mapping ϕ_V (Line 5). Then, for each valid node $v \in V_T^s$, we first update the node mapping ϕ_V (Line 7, and update 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 (Line 11), 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).

As mentioned above, the full LDX verification itself is not a sufficient negative/positive feedback for the DRL agent. We therefore next describe how the verification routine integrates in the LDX compliance reward scheme.

4.2 LDX-based Reward Algorithms

Given the input dataset and Q_X specifications, the end-of-session conditional reward 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). ANON-SYS 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 general-purpose exploration reward signal (as detailed in Section 2.1).

Our End-of-Session reward scheme is depicted in Algortihm 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).

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 here to verify the continuity variables), which returns all valid assignments Φ_V for S_{struct} over T (Line 3). If there are no valid assignment, 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 is the number of matched operational parameters, the higher the reward.

4.2.1 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 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 pre-order traversal (recall from Definition 2.2.1), which symbolizes the order of query operations execution in the session.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 [5]. 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.

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($\phi_V, S_{opr}$)

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

Computation Times As is the case for Tregex [37], evaluating a LDX query may require, in the worst case, iterating over all possible node assignments, of size $\frac{|T|!}{(|T|-|Nodes(Q_X)|)!}$. However, we show in Section 7.3 that in practice, computing the LDX-compliance reward scheme (including both end-of-session and immediate) takes a negligible amount of time, compared to the overall session generation time (approx. 1.02ms per single session and 15.3 seconds in total, for the entire learning process). This is mainly because both the exploration tree T and the query Q_X are rather small[1].

[1] For example, the mean session size in the exploratory sessions collection of [27] is 8.

V Specification-Aware Network

The LDX-compliance reward scheme indeed mitigates the reward sparsity problem (see our experimental evaluation in Section 7.2), yet additional means are required in order to consistently generate compliant sessions, particularly due to the immensity of the action space in the data exploration DRL environment (in which a small portion out of about 100K query operations, may comply with the specifications).

Existing DRL solutions for *constrained* environment mainly focus on *safety constraints* (See [13] for a survey). Frameworks such as [35, 4] suggest *intervention*-based solutions, i.e., to externally modify the agent's chosen action in case they are "unsafe".

Inspired by this line of research, we devise a *specification-aware neural network*, which infer 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 via built-in placeholders for operations "snippets" that are derived from the LDX constraints.

Figure 5.1 depicts the network architecture (The new, specifications-aware functionality is highlighted in pink).

Recall from Section 2.1 that the input layer receives the observation vector (describing the current "state" in the exploration session) and passes it to the dense hidden layers. Then, the agent composes an analysis operation as follows (As in ATENA's network design) via a pre-output layer. This layer allows it 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 5.1, Segment σ_{ops} is connected to the operation types, and the segments $\sigma_1 \dots$ 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". When choosing this operation, 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 in composing full, compliant operations from scratch. For example, using a snippet of `'F, Country, eq'`, derived from our example LDX query Q_1 (Figure 1.2a), will only require the agent to choose a filter term, rather than composing the full query operation.

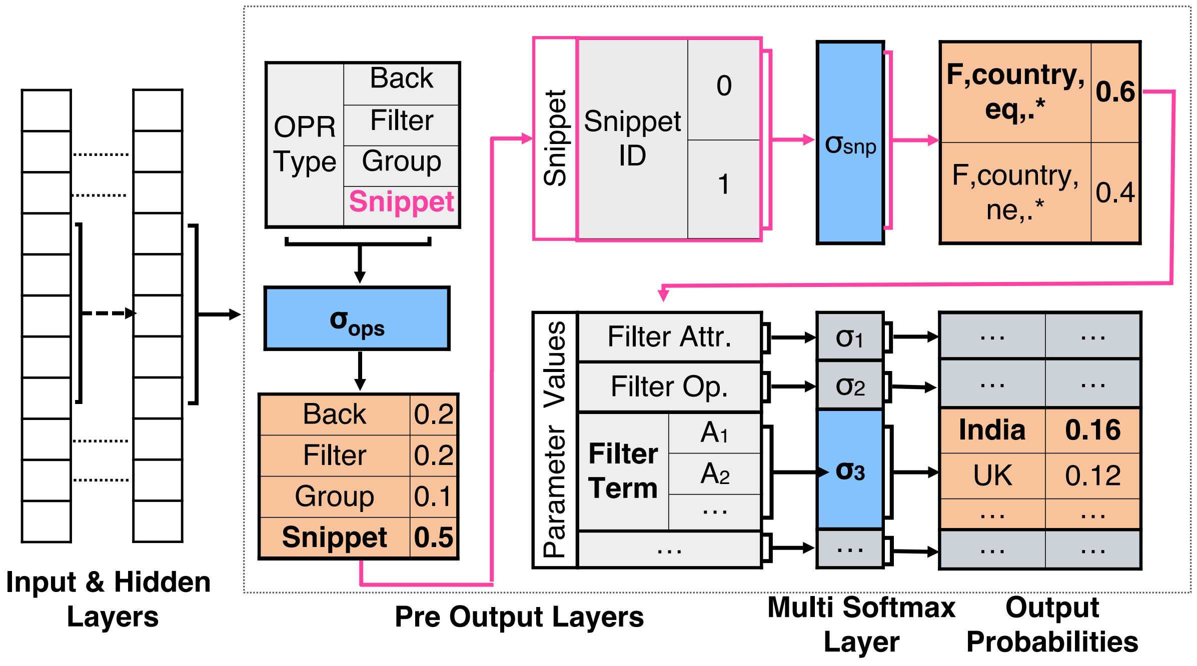


Figure 5.1: Specification-Aware Network Architecture

The architecture derivation process is as follows (See the pink elements in Figure 5.1). First, given a LDX query Q_X, we generate an individual snippet neuron for each remaining specification $s \in S_{opr}$ (In case S contains a disjunction, we generate an individual snippet for each option). All snippet neurons, as depicted in Figure 5.1, are connected to a dedicated multi-softmax segment σ_{snp}. Now, since an operational specification $s \in S_{opr}$ typically restrict 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.

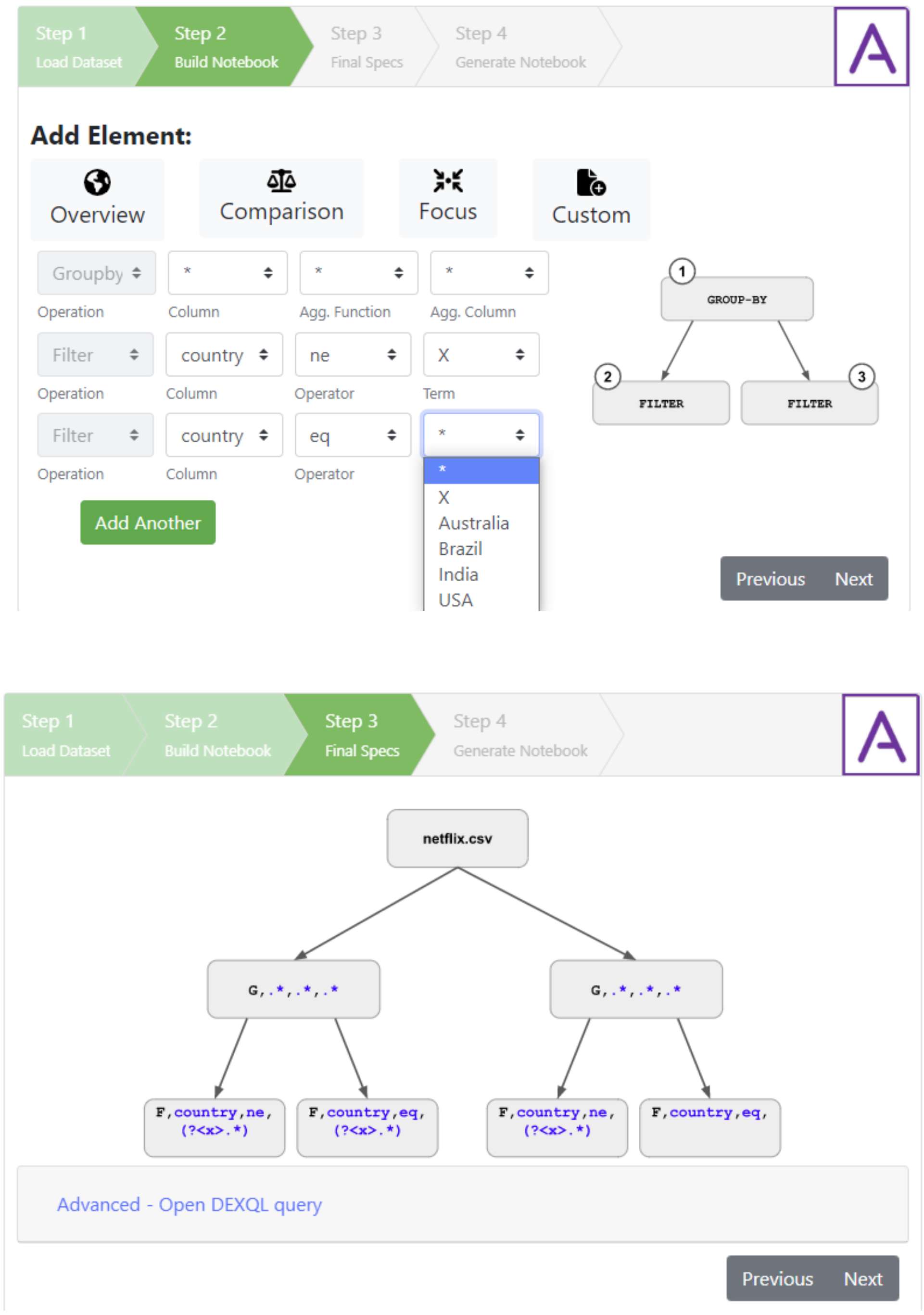
Example 5.0.1. *Figure 5.1 depicts an example selection process, in which the selection probabilities are already computed by the agent's neural network (as can be seen in the right-hand side of the Output Probabilities segment (colored in light orange). First, the agent samples an operation type using the multi-softmax segment σ_{ops}. As in Figure 5.1, 'Snippet' obtained the highest probability of 0.5. If indeed selected, the specific snippet is chosen using Segment σ_{snp}. See that the 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 the segment σ_3. The chosen term is 'India'.*

This network design, as shown in Section 7.2, allows ANON-SYS to consistently generate compliant exploration sessions in 100% of the datasets and LDX queries in our experiments.

VI Simplified User Interface

To demonstrate the potential of building easy-to-use interfaces over LDX, we further implemented a template-based interface – allowing users to easily compose LDX specifications in a simplified, graphical interface. The specifications UI, as illustrated in Figure 6.1, contains several ready-made templates that can be filled by the user and stacked together to form a final set of specifications. For example, a *comparison-element* template, is a group-by followed by two subsequent filter operations. As shown in Figure 1.2a, this allows comparing two segments of the data. The user can then add further constraints on the analysis operations in the element – such as choosing attribute(s), aggregation functions, and aggregation columns. Each such parameter, whether specified by the user or not, can be then marked as a *continuity parameter*, and can be used in subsequent operations. When done, the user adds the current element to the previous ones, choosing at which point in the session it should be executed, and can then keep adding more elements. our implementation currently supports two additional elements – the *Overview Element*, which is a filter operation followed by 2 group-by ones, hence showing two different aggregation views for a data subset; and the *Focus Element*, which is a sequence of filter and group-by operations, such that each filter focuses on a particular group obtained in the previous operation (See Operations 4-7 in Figure 1.2d). In a similar manner, as shown in Figure 6.1, the user can add a custom element.

Finally, when the user clicks on *"Generate Explortaion Notebook"*, the specifications are translated into a LDX specifications file, which is then provided, together with the input dataset, to our DRL framework. The framework then adjusts the neural-network architecture w.r.t. the specifications, and generates a corresponding analysis session. The session is returned to the user as a scientific notebook, allowing her to continue developing the data exploration, by adding explanations and visualizations.


Step 1 Load Dataset | Step 2 Build Notebook | Step 3 Final Specs | Step 4 Generate Notebook

Add Element:

Overview | Comparison | Focus | Custom

Groupby	*	*	*
Operation	Column	Agg. Function	Agg. Column
Filter	country	ne	X
Operation	Column	Operator	Term
Filter	country	eq	*
Operation	Column	Operator	

Term options: *, X, Australia, Brazil, India, USA

Add Another

1 GROUP-BY
2 FILTER
3 FILTER

Previous Next

Step 1 Load Dataset | Step 2 Build Notebook | Step 3 Final Specs | Step 4 Generate Notebook

```
netflix.csv
G,.*,.*,.*                          G,.*,.*,.*
F,country,ne,(?<x>.*)   F,country,eq,(?<x>.*)   F,country,ne,(?<x>.*)   F,country,eq,
```

Advanced - Open DEXQL query

Previous Next

Figure 6.1: Screenshots from the ANON-SYS specification construction interface

VII Experiments

We evaluated ANON-SYS along the following dimensions:

1. Can ANON-SYS consistently generate LDX-compliant exploratory sessions? We used 12 analytical tasks on three datasets. For each task, we composed a different LDX query, and assessed whether ANON-SYS, as well as several degraded versions (missing one or more of the components), can generate a compliant session.

2. Does ANON-SYS converge slower due to its complex reward signals? We analyzed the convergence process of ANON-SYS for the 12 tasks, comparing it to the convergence of ATENA [3] and measured the computation times of the LDX-compliance reward.

3. Are the exploration notebooks generated by ANON-SYS useful and relevant to the analytical tasks? We performed an elaborate user study with 20 participants, who were asked to review the exploration notebooks generated by ANON-SYS, human experts, and two additional baselines. We specifically requested the participants to rate the *relevance* of the exploration notebook to the analytical tasks, in addition to its informativeness and comprehensibility.

4. What type and how many insights do users gain from viewing the exploration notebooks? We further requested users to list the insights they derived from each exploration session. We then examined whether users were indeed able to derive compound, meaningful insights, compared to other baselines.

We next describe our experimental setup and results.

7.1 Setup & Datasets

Implementation. We implemented ANON-SYS in Python 3. The LDX engine uses the internal function (as discussed in Section 4.2) from the TREGEX Python implementation [42]. As for the DRL Architecture, we use Pandas [25] for query operations, and the custom modifications to the neural network (§5) were built in ChainerRL [12]. All our experiments were run on a 24 cores CPU server.

Datasets & Analytical Tasks. We use three commonly used, real-world datasets, each with a different schema and application domain. For each dataset, we devised four different analysis tasks, as listed in both Tables 7.1 and 7.2.

Task Num.	Dataset	Description	Query

T1	NETFLIX	Atypical country	BEGIN CHILDREN {A1, A2}
			A1 LIKE [G,.*] and CHILDREN <B1, B2>
			    B1 LIKE [F,country,eq,(?<Country>.*)]
			    B2 LIKE [F,country,ne,(?<Country>.*)]
			A2 LIKE [G,.*] and CHILDREN <C1, C2>
			    C1 LIKE [F,country,eq,(?<Country>.*)]
			    C2 LIKE [F,country,ne,(?<Country>.*)]

T2	NETFLIX	Successful TV-shows	BEGIN CHILDREN <A1>
			    A1 LIKE [F,type,eq,TV Show] and CHILDREN <B1>
			        B1 LIKE [F,duration,ge,2] and CHILDREN <C1, C2>
			            C1 LIKE [G,(?<col1>.*),.*]
			            C2 LIKE [F,(?<col1>.*),.*] and CHILDREN <D1, D2>
			                D1 LIKE [G,(?<col2>.*),.*]
			                D2 LIKE [F,(?<col2>.*),.*]

T3	NETFLIX	Compare titles of three different actors	A1 LIKE [F,cast,contains,.*] and SIBLINGS {A2,A3} and CHILDREN {B1}
			    B1 LIKE [G,(?<col>.*),(?<func>.*),(?<agg>.*)]
			A2 LIKE [F,cast,contains,.*] and CHILDREN {B2}
			    B2 LIKE [G,(?<col>.*),(?<func>.*),(?<agg>.*)]
			A3 LIKE [F,cast,contains,.*] and CHILDREN {B3}
			    B3 LIKE [G,(?<col>.*),(?<func>.*),(?<agg>.*)]

T4	NETFLIX	Investigate movies' duration	BEGIN CHILDREN {A1}
			A1 LIKE [F,type,eq,Movie] and CHILDREN <B1, B2>
			    B1 LIKE [F,.*] and DESCENDANTS {D1,*}
			        D1 LIKE [G,.*,AVG,duration]
			    B2 LIKE [F,.*] and DESCENDANTS {D2,*}
			        D2 LIKE [G,.*,AVG,duration]

T5	FLIGHTS	Properties of summer-months flights	BEGIN CHILDREN <A1,A2,A3>
			    A1 LIKE [G,(?<col1>.*),.*]
			    A2 LIKE [G,(?<col2>.*),.*]
			    A3 LIKE [F,Month,ge,7] and CHILDREN <B1>
			        B1 LIKE [F,Month,le,8] and CHILDREN <C3,C4,*>
			            C3 LIKE [G,(?<col1>.*),.*]
			            C4 LIKE [G,(?<col2>.*),.*]

T6	FLIGHTS	Explore reasons for delay	BEGIN CHILDREN <A1,A2,A3,A4>
			    A1 LIKE [G,DelayReason,.*]
			    A2 LIKE [F,DelayReason,eq,.*] and CHILDREN {B2}
			        B2 LIKE [G,.*]
			    A3 LIKE [F,DelayReason,eq,.*] and CHILDREN {B3}
			        B3 LIKE [G,.*]
			    A4 LIKE [F,DelayReason,eq,.*] and CHILDREN {B4}
			        B4 LIKE [G,.*]

T7	FLIGHTS	What type of flights have long delays	BEGIN CHILDREN {A1}
			    A1 LIKE [F,DepartureDelay,eq,LARGE_DEALY] and CHILDREN <B1,*>
			        B1 LIKE [G,(?<col>.*),.*] and SIBLINGS {B2, B3, B4}
			        B2 LIKE [F,(?<col>.*),eq,.*]
			        B3 LIKE [F,(?<col>.*),eq,.*]
			        B4 LIKE [F,(?<col>.*),eq,.*]

T8	FLIGHTS	What flights are delayed due to weather?	`BEGIN DESCENDANTS {D1,*}`
			`  D1 LIKE [F,DelayReason,eq,WEATHER] and CHILDREN {B1, B2}`
			`    B1 LIKE [G,.*]`
			`    B2 LIKE [G,.*]`

T9	PLAY STORE	Properties of apps with $\geq$ 1M installs	`BEGIN CHILDREN <A1,A2>`
			`  A1 LIKE [G,(?<col>.*),(?<func>.*),(?<agg>.*)] and CHILDREN {B1}`
			`    B1 LIKE [G,installs,.*] and CHILDREN {}`
			`  A2 LIKE [F,installs,ge,1000000] and DESCENDANTS {D1,*}`
			`    D1 LIKE [G,(?<col>.*),(?<func>.*),(?<agg>.*)]`

T10	PLAY STORE	Compare highly-rated with low-rated apps	`BEGIN CHILDREN <A1,A2>`
			`  A1 LIKE [F,rating,le,2.5] and CHILDREN <C1,C2>`
			`    C1 LIKE  [G,(?<col1>.*),(?<func1>.*),(?<agg1>.*)]`
			`    C2 LIKE  [G,(?<col2>.*),(?<func2>.*),(?<agg2>.*)]`
			`  A2 LIKE [F,rating,ge,4.7] and CHILDREN <D1,D2,*>`
			`    D1 LIKE  [G,(?<col1>.*),(?<func1>.*),(?<agg1>.*)]`
			`    D2 LIKE  [G,(?<col2>.*),(?<func2>.*),(?<agg2>.*)]`

T11	PLAY STORE	Explore apps with backwards compatibility	`BEGIN CHILDREN <A1,A2,A3,A4>`
			`  A1 LIKE [G,.*] and CHILDREN <B1>`
			`    B1 LIKE [G,min_android_ver,CNT,app_id] and CHILDREN {}`
			`  A2 LIKE [G,.*] and CHILDREN <C1,*>`
			`    C1 LIKE [G,min_android_ver,.*]`
			`  A3 LIKE [F,(?<col>.*),.*] and CHILDREN <D1,*>`
			`    D1 LIKE [G,min_android_ver,.*]`
			`  A4 LIKE [F,(?<col>.*),.*] and CHILDREN <E1,*>`
			`    E1 LIKE [G,min_android_ver,.*]`

T12	PLAY STORE	Compare paid apps to free apps	`BEGIN CHILDREN <A1,A2>`
			`  A1 LIKE [G,.*] and CHILDREN <B1>`
			`    B1 LIKE [G,.*] and CHILDREN <D1, D2>`
			`      D1 LIKE [F,type,eq,Paid]`
			`      D2 LIKE [F,type,eq,Free]`
			`  A2 LIKE [G,.*] and CHILDREN <C1, C2>`
			`    C1 LIKE [F,type,eq,Paid]`
			`    C2 LIKE [F,type,eq,Free]`

Table 7.1: LDX Queries for Different Analysis Tasks

Netflix Movies and TV-Shows Dataset [41]. This dataset lists about 8K Netflix titles. Each title is described using 11 features such as the country of production, duration/num. of seasons, etc. The corresponding tasks are (T1) "Atypical country" and (T2) "Successful TV-shows" described in our running example, as well as (T3) "Compare the characteristics of titles of 3 different actors", and (T4) "Investigate properties of movies' duration (length)".

Flight-delays Dataset [38]. Each of the 5.8M records describes a domestic US flight, using 12 attributes such as origin/destination airport, flight duration, issuing airline, departure delay times, delay reasons, etc. The Tasks are: (T5) "Discover properties of summer-months flights", (T6) "Explore reasons for delay", (T7) "Discover properties of flights with long delays" and (T8) "What flights are likely to be delayed by weather?"

Task No.	Dataset	Description	Sess. Size	AS-BIN	AS-DST		AS-IMM		**ANON-SYS**	
				opr.	*str.*	*opr.*	*str.*	*opr.*	*str.*	*opr.*
T1	NETFLIX	Atypical country	6	✗	✓	✓	✓	✓	✓	✓
T2	NETFLIX	Successful TV-shows	7	✗	✗	✗	✓	✗	✓	✓
T3	NETFLIX	Compare titles of three different actors	9	✗	✓	✗	✓	✓	✓	✓
T4	NETFLIX	Investigate movies' duration	8	✗	✓	✗	✓	✗	✓	✓
T5	FLIGHTS	Properties of summer-months flights	7	✗	✓	✗	✓	✗	✓	✓
T6	FLIGHTS	Explore reasons for delay	7	✗	✓	✗	✓	✗	✓	✓
T7	FLIGHTS	What type of flights have long delays	8	✗	✓	✗	✓	✗	✓	✓
T8	FLIGHTS	What flights are delayed due to weather?	7	✗	✓	✓	✓	✓	✓	✓
T9	PLAY STORE	Properties of apps with $\geq$ 1M installs	7	✗	✓	✓	✓	✓	✓	✓
T10	PLAY STORE	Compare highly-rated with low-rated apps	7	✗	✗	✗	✓	✗	✓	✓
T11	PLAY STORE	Explore apps with backwards compatibility	12	✗	✓	✗	✓	✓	✓	✓
T12	PLAY STORE	Compare paid apps to free apps	7	✗	✓	✗	✓	✗	✓	✓
Total				**0/12**	**10/12**	**3/12**	**12/12**	**5/12**	**12/12**	**12/12**

Table 7.2: LDX-Compliance Results for Different LDX Queries

Google Play Store Apps [39]. A collection of 10K mobile apps available on the Google Play Store. Each app is described using 11 features, such as name, category, price, num. of installs, reviews, etc. Tasks 9-12 are: (T9) "Discover properties of popular apps with more than 1M installs", (T10) "Compare highly-rated with low-rated apps", (T11) "Explore apps with backwards compatibility", and (T12) "Compare paid apps to free apps".

7.2 LDX Specifications Compliance Results

We examine whether ANON-SYS can consistently generate compliant exploration sessions for a variety of LDX queries. In addition, we test the performance of degraded versions of ANON-SYS, in order to validate the significance of the system components.

LDX queries, one for each of the twelve tasks (as in Table 7.1), were composed as follows. Q_1 and Q_2 were composed by the authors of this work, and Q_3-Q_{12}, for tasks T3 to T12, were composed by three expert data scientists (after performing a short LDX syntax tutorial).

We compared the performance of ANON-SYS to the following baselines, each missing one or more system components: **(1) AS-BIN** only uses the binary, end-of-session reward, that returns a positive reward if the exploration tree satisfies the LDX query, without using our full reward scheme (Algorithm 1) and specification-aware network (instead, it

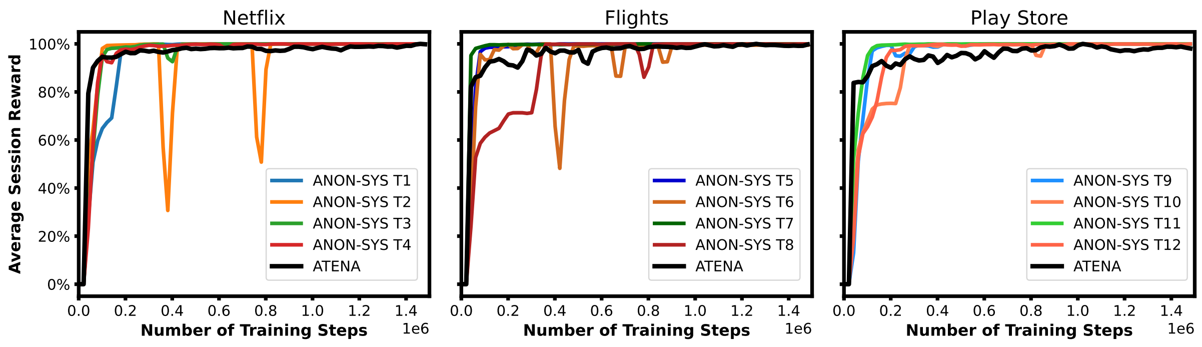


Figure 7.1: Convergence Comparison to ATENA

uses the basic neural network of [3]). **(2) AS-DST** uses the reward scheme, as described in Section 4, without the immediate, per-operation reward and the specification-aware network (using, again, the one from [3]). **(3) AS-IMM** uses the full reward scheme (including the immediate reward) in combination with the basic neural network of [3].

Table 7.2 presents the specification compliance results for the 12 analysis tasks. For each baseline we report whether it successfully generated a session that complies with the structural and the operational specifications (including continuity variables). A full compliance is achieved if a baseline is compliant with both type of specifications.

First, see that AS-BIN, 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 reward and the large size of the action space. AS-DST, which uses the more flexible compliance reward (at the end of each session) obtains better results – fully complying with 3 queries, and partially complying (structure only) of 7 additional ones. The performance improvement here is due to the differentiation we make in Algorithm 2, between structural and operational specifications, hence providing a finer-grain feedback. Next, AS-IMM obtains a significant improvement – it is able to comply with the structural specifications of 12/12 queries, however, was fully compliant only for 5/12 queries. This shows the effectiveness of the immediate reward, as described in Section 4.2 to detect structural violations. Yet this baseline is still ineffective in generating the adequate query operations, as instructed in the operational constraints.

Finally, see that only ANON-SYS, which uses both the reward-scheme *and* the specification-aware neural network, is able to generate compliant sessions for all (12/12) LDX queries. This shows that our adaptive network design, as described in Section 5, is particularly useful in encouraging the agent to perform specification-compliant operations – despite the inherently large size of the action-space.

7.3 Convergence & Running Times

Next, we examine the convergence and running times of ANON-SYS, and compare them with ATENA [3], in order to assess whether the complex specification-compliance reward scheme of ANON-SYS significantly affect its execution times.

Convergence. Figure 7.1 depicts the convergence plots for each of the three experimental datasets, namely the process of learning to produce a compatible session while optimizing on all the reward signals. The convergence for each LDX query i (corresponding to Task i) is depicted using a line labeled *'ANON-SYS Ti'*, whereas the black line in each figure depicts the convergence process of ATENA [3] which serves as a baseline (Recall that ATENA can only produce one, generic analysis session 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 ANON-SYS are roughly similar, both fully converging to 100% of the maximal reward after 1M steps at most. ANON-SYS sometimes shows sudden drops in reward (e.g., for ANON-SYS T2 and T6). These drops stem from the heaviy penalty on violations of structural specifications. As can be seen in the plots, the agent soon "fixes" these violations and stabilizes the reward. Furthermore, see that ANON-SYS sometimes converges faster than ATENA (e.g., for the Play Store dataset), in which ATENA takes 0.85M steps and ANON-SYS only 0.4M (on average). This happens as the user's specifications assist the agent in focusing on a narrower space of promising sessions, and eliminate the need for further exploring areas of lesser relevance.

Running times. We further examined whether the LDX-compliance reward scheme affect the running times of ANON-SYS.In practice, for all LDX queries, the computation times of the full compliance reward were negligibly small compared to the entire learning process – $0.26\%(\pm 0.0008\% SD)$. 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. Particularly, the compliance verification (Algorithm 1) takes only 0.001%, the operational-based reward (Algorithm 2, Lines 9-12) 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.

7.4 User Study

We performed a user study with the goal of (1) Assessing whether the generated exploratory sessions are indeed relevant to the analysis tasks, and also interesting and coherent, and

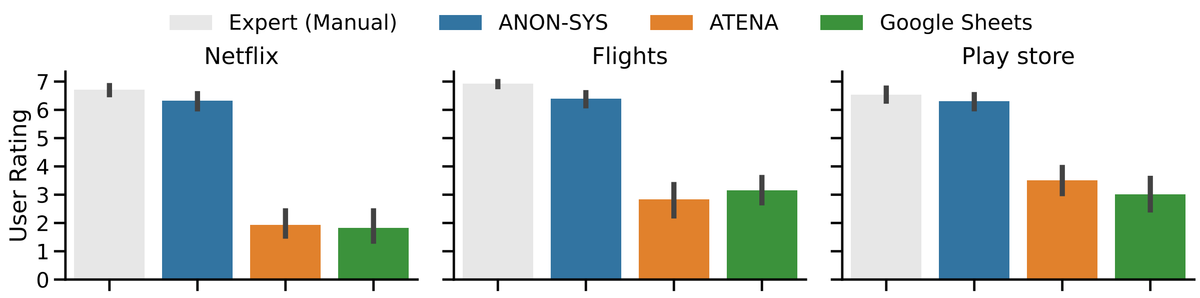


Figure 7.2: User Study – Rating of Relevance of the Analytical Session to the Given Tasks

(2) comparing ANON-SYS to alternative approaches.

We therefore recruited 20 participants, all computer-science graduate/undergraduate students with some experience in data analysis or science. Each participant was presented with a dataset and an analysis task (selected at random from Tasks 1,2,5,6,9 and 10 in Table 7.2). After a brief explanation on the dataset's schema and task, we referred the participants to a Jupyter notebook, containing a corresponding exploration session generated by either ANON-SYS or one of the following baselines: **(1) Experts-Manual.** The same three expert data scientists who composed the LDX queries, all CS graduates with industry experience of more than 3 years, were further asked to manually build exploration notebooks as follows: Each expert was assigned a dataset and its corresponding tasks. We then asked the experts to first freely explore the dataset, and then compose a notebook of selected, interesting query operations that are relevant for the given task. We allowed the experts to use the same query operations supported in ANON-SYS (filter, group-by and aggregate). **(2) ATENA [3].** we ran ATENA on each dataset, which automatically outputs an exploration session, yet cannot take user's specifications. Therefore, for all different analysis tasks on a dataset, the same session was generated. **(3) Google Sheets [40] Explorer.** We also compared ANON-SYS to a commercial alternative, which is Google Sheets' ML-based, automatic exploration tool. This is the only available exploration tool, to our knowledge, that can consider user specifications (yet rather limited ones, such as a column name, or a data subset to focus on). The tool then generates a sequence of charts and visualizations, w.r.t. the given specifications. The specifications were again composed by our expert users (the same ones who built the notebooks for Baseline 1). For example, for task T5, "Discover properties of summer-months flights", the expert chose the columns 'month', 'airline', 'delay-reason' and 'scheduled arrival'/'departure' and focused on the subset of flights form July and August.

Each exploration session generated by the baselines was presented in a Jupyter notebook interface (except for Google Sheets, which has its own, web-based interface). After reviewing each notebook, participants were asked to rate it on a scale from 1 (lowest) to 7 (highest) according to the following criteria: (1) *Relevance* - To what degree is the exploration notebook relevant to the given analysis task? (2) *Informativeness* — How informative is the exploration notebook? (3) *Comprehensibility* - To what degree is the notebook comprehensible and easy to follow?

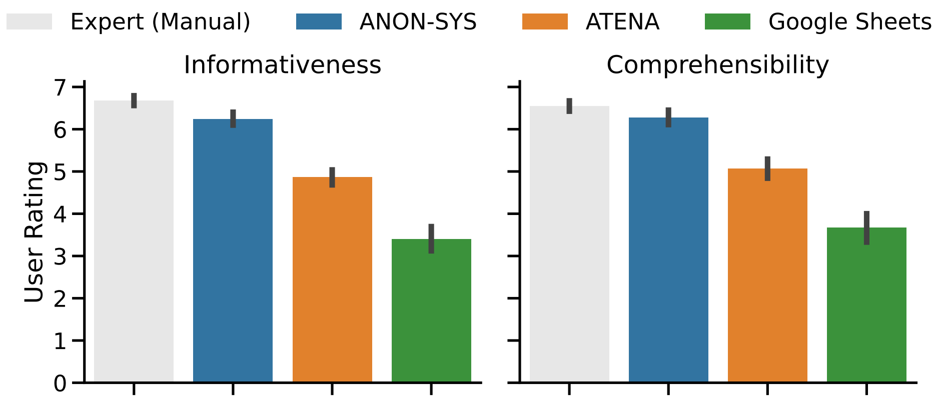


Figure 7.3: Informativeness & Comprehensibility Rating

Results. Figure 7.2 presents the Relevance score of ANON-SYS and the baselines, for each of the three dataset (the results are averaged across all participants and the two tasks for each dataset). The vertical line depicts the .95 confidence interval. Indeed, the manually-prepared notebooks 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 ANON-SYS obtains almost the same scores – 6.32, 6.39, 6.30 (resp.) for its automatically generated exploratory sessions. 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 specification options – makes their solutions insufficient for generating *relevant* exploration notebooks to the given tasks.

Next, we inspected the *informativeness* and *coherency* scores. Figure 7.3 depicts the average scores, over all three datasets (Black vertical lines represent the .95 confidence interval).

The manually-prepared notebooks again obtain the highest scores, and both ATENA and Google Sheets now obtain higher scores: ATENA with 4.86 and 5.07, and Google Sheets is slightly behind with 3.40 and 3.67 (for informativeness and coherence, resp).

Interestingly, ANON-SYS is able to obtain higher scores (6.24 and 6.28 for informativeness and coherency, resp.). This particularly shows that the high relevance scores obtained by ANON-SYS are *not* granted on the account of informativeness and coherency, and *ANON-SYS generates exploration notebooks that are both highly relevant to the task, as well as informative and coherent.*

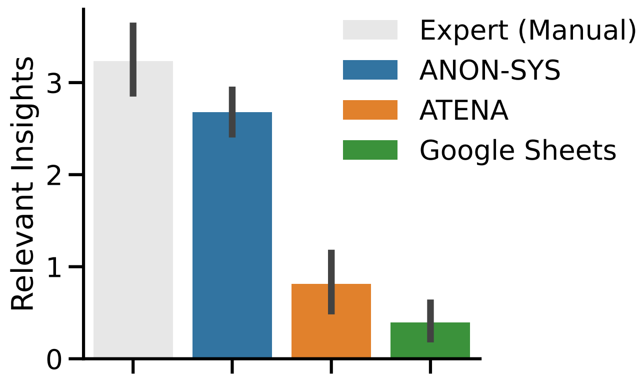


Figure 7.4: Average Num. of Insights

Task	Insight
T1	"The ratio of movies-series in India is higher than the movies-series ratio anywhere else."
T2	"US TV Shows with more than 1 season are mostly dramas and comedies."
T5	"While about 1/3 of the flights are in summer months, the number of monthly delayed flights is roughly the same as for the rest of the year."
T6	"While long flights are not delayed often, if they are this is mainly for a security reason."
T9	"Apps with 1M installs are often free, having high rating, and support Android 4."
T10	"Low rated apps have much less reviews and are less updated than apps with high rating."

Figure 7.5: Insight Examples Derived Using ANON-SYS

7.5 Analysis of User-Derived Insights

In the same user study, as presented in Section 7.4, we additionally asked the participants to provide a list of insights they derived from viewing an exploration notebook. To avoid any information leak between notebook, we only asked the participants to list insights *only for the first notebook they reviewed.* As expected, some insights were correct, but irrelevant to the task at hand. For example, User #12 who reviewed the Google Sheets output for Task $T2$, inferred that *"Netflix has more movies than TV shows"*, which is correct, yet irrelevant to the analytical task $T2$ ("investigate successful TV shows"). We therefore asked the experts (from Baseline 1) to review the insights derived by the users, and filter out the irrelevant ones.

Figure 7.4 shows the average number of task-relevant insights derived using each baseline. Using ANON-SYS, user derived an average of 2.7 relevant insights per task, which is second only to the manually-built notebooks by experts (3.2 insights per task). ATENA and Google Sheets are again far behind with an average of 0.8 and 0.4 relevant insights per task (resp). Last, to further examine the nature and quality of the insights derived from ANON-SYS's data stories, we provide in Table 7.5 an example insight for each task, as obtained by selected participants in the user study. Observe that participants derived compound, non-trivial insights that are indeed relevant to the corresponding tasks.

VIII Related Work

Automated/Guided Data Exploration Numerous previous works focus on automating (whether fully or partially) data exploration, by suggesting the user potentially promising exploration steps or entire sessions. We divide this line of work to three types of systems, as is also summarized in Table 8.1. First, exploration recommender systems [11, 27, 50, 17, 10, 33], present the user with recommendation for her next exploratory operations. Within this category, *Data driven* systems like [17, 10, 33] use heuristic notions of *interestingness* and present the user with query operations achieving high such scores. Naturally, such systems consider only the data at hand and are not fitted to the user's personal needs. In comparison, *Log-based* recommender systems such as [11, 27, 50] leverage former exploratory sessions in a collaborative filtering manner, hence provide recommendations w.r.t. the current user's previous steps. In any case, both types of systems do not consider user preferences, and generate only single-step recommendations (rather than a full coherent session).

The second type of systems are based on *active learning* [8, 15, 7]. These system interactively harvest feedback on presented tuples ("interesting" or "not interesting"), in order to guide the user to a desired subset of the data. These system require the full attention and feedback from the users (therefore cannot self-train), and do not account for information needs that require multiple query operations.

Closer to our work, *DRL-based* systems, e.g., [3, 30, 29] can generate entire exploratory sessions without requiring any training data. Yet unfortunately, these systems do not support user preferences as input, therefore cannot generate personalized, task-depended sessions like ANON-SYS.

Last, note that there are a plethora of previous work devoted to facilitate data exploration in complementary means to ANON-SYS. Examples are simplified exploration interfaces for non-programmers [19, 40], recommendation of data visualizations [46], data summaries [36], explanations [6], and insights extractions [43].

Visualization Specifications Full and partial specification languages have been suggested in previous work in order to facilitate the programmatic creation of data visualizations. Examples are Matplotlib [16] and Vega [48], which support only full specifications, and the declarative Vega-lite [34] which features higher-level, simpler constructs. Closer to our work, visualization recommender systems such as [47, 49, 20] allow users to provide

Approach		**Self-train**	**Multi-step**	**Personalized**
Exploration rec. systems	[17, 10, 33]	Yes	No	No
	[11, 27, 50]	Needs training data	No	Partially
Active learning systems	[8, 15, 7]	Needs user feedback	No	Yes
DRL systems	[3, 30, 29]	Yes	Yes	No
ANON-SYS		**Yes**	**Yes**	**Yes**

Table 8.1: Summary of works in Automated Exploration

partial specifications (using the same syntax as, e.g., Vega-lite) and obtain recommended visualizations that satisfy the constraints.

While ANON-SYS is different in its nature (as it focuses on data operations), its specification language LDX is also the first one, to our knowledge, designed for specifying a full *sequence* of interconnected operations, rather than a single desired object.

Deep Reinforcement Learning in the Data Research Communities The databases and data science research communities have been successfully integrating Deep Reinforcement Learning (DRL) components for solving difficult, planning-based tasks: For example, DRL-based query optimizers such as NEO [23] and SkinnerDB [44] learn to generate efficient query execution plans; Automated Machine Learning (AutoML) frameworks such as AlphaD3M [9] and DeepLine [14] uses a DRL engine to intelligently search the vast space of the data scientific operations.

ANON-SYS also uses a DRL framework, but focuses on the required solutions to support input user specifications. As mentioned above, user specifications introduce a significant *reward sparsity* [24] problem, overcame in ANON-SYS by a complex, flexible compliance reward scheme as well as a a specification-aware neural-network architecture. The overall solution allows ANON-SYS to consistently generate useful exploration sessions that meet users' custom specifications.

IX Conclusion & Future Work

This work presents ANON-SYS, a framework for auto-generating personalized exploration notebooks. We introduced LDX, a specification language for exploratory sessions with dedicated continuity constructs for the desired query operations. We then empirically shown that all our DRL engine components, including the flexible compliance reward scheme and the specification-aware architecture, are required for consistently generating useful compatible notebooks.

There are many potential directions for future work. First, developing higher-level, more intuitive interfaces for composing LDX queries, such as a NL-to-LDX (as was previously developed, e.g., for SQL [22]) is necessary for reaching a broader audience. Second, automatically embedding textual captions and visualizations in the notebook is also a promising direction (e.g., by incorporating systems such as [49, 20]) to allow for a richer, more compelling output notebook.

References

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

[2] O. Bar El, T. Milo, and A. Somech. 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*, pages 2873–2876, 2019.

[3] O. Bar El, T. Milo, and A. Somech. Automatically generating data exploration sessions using deep reinforcement learning. In *SIGMOD*, 2020.

[4] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[5] N. Dershowitz and S. Zaks. Enumerations of ordered trees. *Discret. Math.*, 31(1):9–28, 1980.

[6] D. Deutch, A. Gilad, T. Milo, and A. Somech. Explained: explanations for eda notebooks. *Proceedings of the VLDB Endowment*, 13(12):2917–2920, 2020.

[7] L. Di Palma, Y. Diao, and A. Liu. A factorized version space algorithm for" human-in-the-loop" data exploration. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1018–1023. IEEE, 2019.

[8] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Aide: An active learning-based approach for interactive data exploration. *TKDE*, 2016.

[9] I. Drori, Y. Krishnamurthy, R. Rampin, R. d. P. Lourenco, J. P. Ono, K. Cho, C. Silva, and J. Freire. Alphad3m: Machine learning pipeline synthesis. *arXiv preprint arXiv:2111.02508*, 2021.

[10] M. Drosou and E. Pitoura. Ymaldb: exploring relational databases via result-driven recommendations. *VLDBJ*, 22(6), 2013.

[11] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. Querie: Collaborative database exploration. *TKDE*, 2014.

[12] Y. Fujita, P. Nagarajan, T. Kataoka, and T. Ishikawa. Chainerrl: A deep reinforcement learning library. *arXiv preprint arXiv:1912.03905*, 2019.

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

[14] Y. Heffetz, R. Vainshtein, G. Katz, and L. Rokach. Deepline: Automl tool for pipelines generation using deep reinforcement learning and hierarchical actions filtering. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2103–2113, 2020.

[15] E. Huang, L. Peng, L. D. Palma, A. Abdelkafi, A. Liu, and Y. Diao. Optimization for active learning-based interactive database exploration. *Proceedings of the VLDB Endowment*, 12(1):71–84, 2018.

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

[17] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Smart drill-down. *Target*, 6000:0, 2014.

[18] M. B. Kery, M. Radensky, M. Arya, B. E. John, and B. A. Myers. The story in the notebook: Exploratory data science using a literate programming tool. In *CHI*, 2018.

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

[20] D. J.-L. Lee, D. Tang, K. Agarwal, T. Boonmark, C. Chen, J. Kang, U. Mukhopadhyay, J. Song, M. Yong, M. A. Hearst, et al. Lux: always-on visualization recommendations for exploratory dataframe workflows. *PVLDB*, 15(3):727–738, 2021.

[21] R. Levy and G. Andrew. Tregex and tsurgeon: Tools for querying and manipulating tree data structures. In *LREC*, pages 2231–2234. Citeseer, 2006.

[22] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1), 2014.

[23] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul. Neo: A learned query optimizer. *PVLDB*, 12(11):1705–1718, jul 2019.

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

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

[26] T. Milo, C. Ozeri, and A. Somech. Predicting "what is interesting" by mining interactive-data-analysis session logs. In *22nd International Conference on Extending Database Technology (EDBT)*, pages 456–467, 2019.

[27] T. Milo and A. Somech. Next-step suggestions for modern interactive data analysis platforms. In *KDD*, 2018.

[28] J. M. Perkel. Why jupyter is data scientists' computational notebook of choice. *Nature*, 563(7732):145–147, 2018.

[29] A. Personnaz, S. Amer-Yahia, L. Berti-Equille, M. Fabricius, and S. Subramanian. Balancing familiarity and curiosity in data exploration with deep reinforcement learning. In *Fourth Workshop in Exploiting AI Techniques for Data Management*, pages 16–23, 2021.

[30] A. Personnaz, S. Amer-Yahia, L. Berti-Equille, M. Fabricius, and S. Subramanian. Dora the explorer: Exploring very large data with interactive deep reinforcement learning. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4769–4773, 2021.

[31] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3), 2017.

[32] A. Rule, A. Tabard, and J. D. Hollan. Exploration and explanation in computational notebooks. In *CHI*, 2018.

[33] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *EDBT*, 1998.

[34] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2016.

[35] W. Saunders, G. Sastry, A. Stuhlmueller, and O. Evans. Trial without error: Towards safe reinforcement learning via human intervention. *arXiv preprint arXiv:1707.05173*, 2017.

[36] M. Singh, M. J. Cafarella, and H. Jagadish. Dbexplorer: Exploratory search in databases. *EDBT*, 2016.

[37] M. Skala. A structural query system for han characters. *arXiv preprint arXiv:1404.5585*, 2014.

[38] Flights Dataset (Kaggle). `https://www.kaggle.com/usdot/flight-delays`, 2022.

[39] Google Play Store Dataset (Kaggle). `https://www.kaggle.com/lava18/google-play-store-apps`, 2022.

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

[41] Netflix Dataset (Kaggle). `https://www.kaggle.com/shivamb/netflix-shows`, 2022.

[42] Tregex implementation. `https://github.com/yandex/dep_tregex`, 2022.

[43] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang. Extracting top-k insights from multi-dimensional data. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1509–1524, 2017.

[44] I. Trummer, J. Wang, Z. Wei, D. Maram, S. Moseley, S. Jo, J. Antonakakis, and A. Rayabhari. Skinnerdb: Regret-bounded query evaluation via reinforcement learning. *ACM Transactions on Database Systems (TODS)*, 46(3):1–45, 2021.

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

[46] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13), 2015.

[47] K. Wongsuphasawat, Y. Liu, and J. Heer. Goals, process, and challenges of exploratory data analysis: An interview study. *arXiv preprint arXiv:1911.00568*, 2019.

[48] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *TVCG*, 2016.

[49] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 chi conference on human factors in computing systems*, pages 2648–2659, 2017.

[50] C. Yan and Y. He. 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*, pages 1539–1554, 2020.

[51] C. Yan and Y. He. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In *SIGMOD*, pages 1539–1554, 2020.