Leam: An Interactive System for In-situ Visual Text Analysis

Sajjadur Rahman
Megagon Labs
sajjadur@megagon.ai

Peter Griggs
MIT
pggriggs@mit.edu

Çağatay Demiralp
Sigma Computing
cagatay@sigmacomputing.com

ABSTRACT

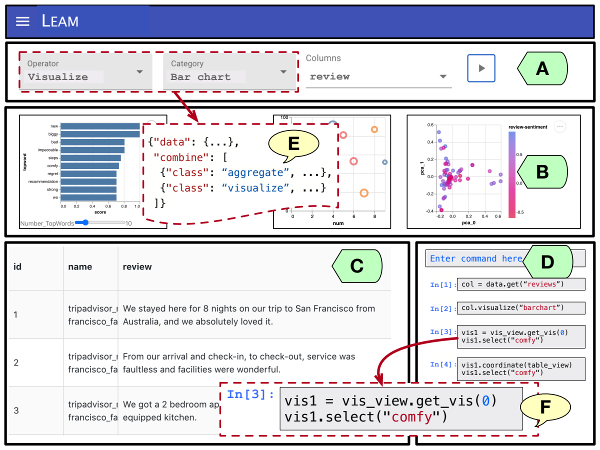
With the increase in scale and availability of digital text generated on the web, enterprises such as online retailers and aggregators often use text analytics to mine and analyze the data to improve their services and products alike. Text data analysis is an iterative, non-linear process with diverse workflows spanning multiple stages, from data cleaning to visualization. Existing text analytics systems usually accommodate a subset of these stages and often fail to address challenges related to data heterogeneity, provenance, workflow reusability and reproducibility, and compatibility with established practices. Based on a set of design considerations we derive from these challenges, we propose Leam, a system that treats the text analysis process as a single continuum by combining advantages of computational notebooks, spreadsheets, and visualization tools. Leam features an interactive user interface for running text analysis workflows, a new data model for managing multiple atomic and composite data types, and an expressive algebra that captures diverse sets of operations representing various stages of text analysis and enables coordination among different components of the system, including data, code, and visualizations. We report our current progress in Leam development while demonstrating its usefulness with usage examples. Finally, we outline a number of enhancements to Leam and identify several research directions for developing an interactive visual text analysis system.

1 INTRODUCTION

With the rapid growth of the e-commerce economy, the internet has become the platform for many of our everyday activities, from shopping to dating, to job searching, to travel booking. A recent study projects the worldwide e-commerce sales to be around six trillion dollars by 2023 [28], nearly a 50% increase over the current market. This growth has contributed to the proliferation of digital text, particularly user-generated text (reviews, Q&As, discussions), which often contain useful information for improving the services and products on the web. Enterprises increasingly adopt text mining technologies to extract, analyze, and summarize such information from the unstructured text data. Like data analysis at large, text data analysis also benefits from interactive workflows and visualizations as they facilitate accessible, rapid iterative analysis. Therefore, we characterize the text data analysis process more formally as visual interactive text analysis (VITA hereafter). A challenging aspect of VITA is its iterative and nonlinear nature—it is a multistage process that involves tasks like data preprocessing and transformation, model building, hypothesis testing, and insight exploration, all of which require multiple iterations to obtain satisfactory outcomes.

CIDR'21, January 10-13, 2021, Chaminade, USA.,

While many commercial and open-source tools support various stages of VITA [17, 26], none of these capture the end-to-end VITA pipeline. For example, spreadsheets support directly processing and manipulating data, computational notebooks enable flexible exploratory analysis and modeling, and visualization systems, typically based on chart templates, facilitate quick interactive visual analysis. There are also many customized text analytics tools [17], often in the form of research prototypes, that support specific use-cases like review exploration [34], sentiment analysis [14], and text summarization [8]. Unfortunately, none of these solutions accommodate the inherently cyclic, trial-and-error-based nature of VITA pipelines in an integrated manner.



```
{"data": {...},
 "combine": [
  {"class": "aggregate", ...},
  {"class": "visualize", ...}
 ]}
```

id	name	review
1	tripadvisor_i francisco_fa	We stayed here for 8 nights on our trip to San Francisco from Australia, and we absolutely loved it.
2	tripadvisor_i francisco_fa	From our arrival and check-in, to check-out, service was faultless and facilities were wonderful.
3	tripadvisor_i francisco_fa	We got a 2 bedroom ap equipped kitchen.

```
In[1]: col = data.get("reviews")
In[2]: col.visualize("barchart")
In[3]: vis1 = vis_view.get_vis(0)
       vis1.select("comfy")
In[4]: vis1.coordinate(table_view)
       vis1.select("comfy")
```

```
In[3]: vis1 = vis_view.get_vis(0)
       vis1.select("comfy")
```

Figure 1: Leam **user interface. (A) Operator View enables users to perform visual text analytics** (VITA) **operations using dropdown menus, (B) Visualization View holds a carousel of interactive visualizations created by users, (C) Table View displays the data and its subsequent transformations, and (D) Notebook View allows users to compose and run** VITA **operations using a declarative language. Inset (E) shows the** VTA ***json*** **specification for the bar chart operator in Operator View. Inset (F) shows a declarative** VTA **command for interacting with the bar chart from Notebook View.**

While supporting the entire VITA life-cycle within a single system does seem natural, developing it leads to several challenges related to (a) extensibility and expressivity of VITA workflows, (b) their continuity and reproducibility, (c) data heterogeneity and provenance, and (d) synchronization of user interactions. We document these challenges in Section 2. These challenges are a culmination of (1) our conversations with practitioners while working in an industry research lab, part of a larger company with more than three hundred e-commerce subsidiaries, (2) prior research, particularly those reporting from interview studies on data analysis workflows [16, 34], and (3) our experience in developing and evaluating interactive data systems. As such, the list of challenges here is intended to be a useful guide informing research and development on text analytics systems, not a comprehensive enumeration, and inevitably reflects our personal taste. We propose a set of design criteria (desiderata) to address the challenges of a VITA system (Section 2). One crucial theme underlying these criteria is the consideration of data analysis as a single continuum, not as discrete steps of tasks performed in isolation.

In this paper, we introduce Leam, a one-stop-shop for visual interactive text analysis. Leam combines the advantages of spreadsheets, computational notebooks, and visualization tools by integrating a Notebook View with interactive views of raw and transformed data (Figure 1). A key component in the design of Leam is a visual text algebra (VTA) that enables users to specify complex VITA operations over heterogeneous data and their visual representations; either using a declarative language in the Notebook View (Figure 1F) or by creating operators in the front-end that translates to *json*-style VTA specifications (Figure 1E). Through usage examples, we demonstrate the expressivity of VTA and how it enables Leam to support diverse tasks ranging from data cleaning to visualization. Moreover, to facilitate efficient execution of VTA on heterogeneous data, we introduce a new data model extending dataframes called VITAframe. Based on our experience in developing Leam, we have identified several system design challenges related to storage model efficiency, scalable computation of VITA workflows, data and workflow versioning, and workflow optimization. To address these challenges, we identify several research directions that may enhance the capabilities of Leam. We have made the current version of Leam open-source at https://github.com/megagonlabs/leam.

The goal of this paper is to explore challenges in and design considerations for VITA systems development, present vital components of Leam that enable interactive text analysis (*e.g.*, VITAframe and VTA), and identify some concrete research directions based on our experience of developing Leam.

2 VITA: CHALLENGES AND DESIGN GOAL

Motivating Example. Ada, a data scientist in the e-commerce department of a retail business, has been tasked to analyze customer reviews of products purchased from their website. Ada would like to capture the underlying topics by performing topic modeling and clustering to characterize the review corpus better. Figure 2 captures the use-case which involves—(a) preprocessing the data (*clean*), creating feature vectors from the text reviews (*featurize*), creating topic vectors from the corpus (*topic modeling*), clustering reviews into topics (*cluster assignment*), and finally, visualizing the clusters by projecting the topics vectors to lower dimensions (2D) using feature transformation techniques such as PCA (*visualize*). In practice, the workflow may be non-linear, and each step may require multiple passes and different tools. In the process, visualizations are useful not only for exploratory analysis or final presentation but also for every other step—VITA workflows resemble the *read-eval-print loop* (REPL) approach where users perform

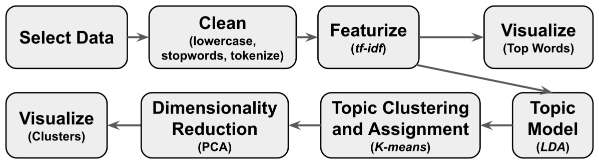

Select Data
Clean (lowercase, stopwords, tokenize)
Featurize (tf-idf)
Visualize (Top Words)
Topic Model (LDA)
Topic Clustering and Assignment (K-means)
Dimensionality Reduction (PCA)
Visualize (Clusters)

Figure 2: An example VITA use-case: topic exploration.

incremental operations on data and examine intermediate results. We now characterize the challenges of the existing VITA workflows in the context of this use-case as follows:

C1. Workflow discontinuity. As mentioned in Section 1, data scientists lack tools that support different VITA operations and workflows within an integrated environment. For example, to define the data cleaning rules, Ada first visually inspects the data using tools like spreadsheets. Next, they execute those rules in a computational notebook, *e.g.*, Jupyter. Upon inspecting the data in the spreadsheet, Ada may revise the rules in the notebook. To visualize top-ranked words after the featurization step (*e.g.*, a bar chart of words ranked by their TF-IDF scores), they need to either use dedicated visualization tools or write scripts in the notebook. Therefore, even completing simple tasks may require accessing different tools, which can be a cumbersome user experience due to, for example, the logistical and cognitive overhead of context switching.

C2. Limited coordination. VITA necessitates coordination among different views (*e.g.*, between visualization and raw data). The high dimensional text data can be difficult to interpret, and users often map different facets of the data to visualizations for better interpretability. However, without coordination between perceptual components and the data space, understanding the relations between the facets of the same entities on demand can be challenging. For example, say Ada wants to inspect which reviews contain a top-word in the bar chart (generated after featurization). However, visualizations in notebooks or visualization tools are decoupled from the data. As a result, Ada has to either open and then filter the data in a spreadsheet or programmatically filter the data from the notebook to inspect the relevant reviews. Therefore, the lack of coordination impacts both workflow continuity and the user's ability to explore data effectively.

C3. Data types and workflow diversity. VITA workflows deal with heterogeneous data (*e.g.*, text, visualizations) and workflows (*e.g.*, in use-cases like text summarization, sentiment analysis). While there are a number of VITA tools for specific workflows [17], more often than not, these tools use a stack of independent solutions for data storage and processing glued together by scripting languages like Python and R. These bespoke solutions may not capture all VITA requirements, like direct data manipulation and interactive visual coordination. As a result, users are often forced to develop new and heavily customized systems on top of these solutions.

C4. On-demand workflow authoring. VITA workflows contain a variety of operations, *e.g.*, cleaning, featurization, interactive visualization, classification. Similar to relational [10] or data visualization algebra [25], VITA operations with similar objectives can be grouped into high-level categories. Moreover, operations in different categories can be combined to compose new operation pipelines. For example, cleaning and featurization can be combined into a preprocessing pipeline. As existing systems lack any formalization of the operations and their application, the onus is on the user to design the optimal analytics workflow for different use-cases.

C5. VITA *Session management.* As demonstrated in the usage example, VITA workflows inspire the trial-error-correct-style iterative approach—users often need to reproduce previous steps of the workflow, make updates, and rerun the subsequent steps. Therefore, ensuring reproducibility of VITA sessions require management of dataset versions produced by various operations, the operation logs, and different states of and interactions on the visual representations of the data. Prior work from the data management community focused on versioning structured datasets [12], versioning code for debugging workflows [6, 19] and managing deep learning models [18]. However, these systems lack support for versioning an end-to-end VITA workflow involving heterogeneous data types and user interactions spanning multiple views.

3 DESIGN CRITERIA

We propose the following design principles to address the challenges related to VITA:

D1. In-situ analytics. VITA systems should provide a one-stop-shop (**C1**) where users can directly manipulate (spreadsheets) and visualize (visualization tools) data while writing scripts (notebooks) to immediately view the effects on data and visualizations without context switching between tools.

D2. Multi-view coordination. Beyond integrating multiple views within a single interface, VITA systems should enable coordination between these views (**C2**). Multiple coordinated views capture the context of the user's exploration across different views [32] and help users understand the data better as they view it through different connected representations.

D3. Heterogeneous data management. VITA systems should support heterogeneous data types (*e.g.*, texts, visualization), treating them as first-class citizens of the underlying data model (**C3**). Instead of developing bespoke data management solutions, VITA systems should adapt their underlying storage model to accommodate these data types and enable tight coupling between the data model and the analytical workflows to ensure fast and efficient data access.

D4. Expressivity and accessibility. VITA systems should provide an expressive specification language to represent and communicate the entire breadth of workflows within the domain (**C4**). Moreover, the specification language should be accessible to existing tools to allow more expressive operations. For example, the specification language can be packaged as a Python library with an interactive widget with support for a subset of VITA operations in a computational notebook.

D5. Provenance. VITA systems should support advanced provenance tracking for heterogeneous data types and various workflows to ensure reproducibility and encourage workflow and data re-use. Moreover, these systems should track user interactions on visual components to enable the versioning of states and dependencies of different views.

4 LEAM: IN-SITU ANALYSIS

LEAM backend features an in-memory dataframe, a versioning database, a compiler for translating VTA commands that the execution engine runs, and a session manager to manage data, code, and visualizations. Figure 3 shows the overview of the LEAM system architecture. The components with dashed borders ("- -") are partially implemented and require further refinement. We discuss the front-end components later.

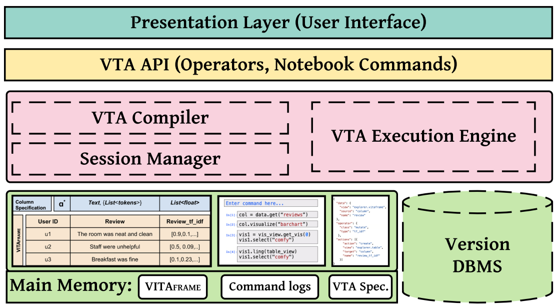
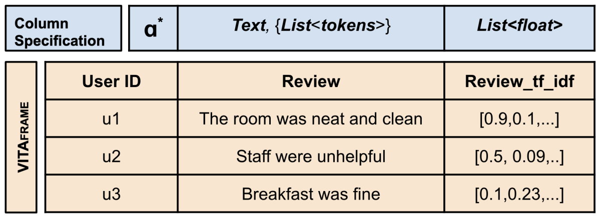


Figure 3: LEAM system architecture.

4.1 Data Model

The underlying data structure of LEAM is a dataframe. A dataframe is a multidimensional array, A_{mn}, with a vector of row labels, $R = \{R_1, R_2, \dots, R_m\}$ and a vector of column labels, $C = \{C_1, C_2, \dots, C_n\}$ [21]. We opted for dataframes instead of a database since VITA tasks (*e.g.*, featurization, classification) cannot always be conveniently performed inside the database [15]. Dataframes are widely used in exploratory data analysis, including VITA due to their coverage of a wide variety of data analysis operators [21]. They also do not require data to be defined schema-first allowing flexibility in supported data types and data structures. Finally, dataframes provide a functional interface suitable for REPL-style VITA workflows to perform "quick and dirty" analysis.



Column Specification	α^*	*Text*, {*List<tokens>*}	*List<float>*
VITAFRAME	User ID	Review	Review_tf_idf
	u1	The room was neat and clean	[0.9,0.1,...]
	u2	Staff were unhelpful	[0.5, 0.09,..]
	u3	Breakfast was fine	[0.1,0.23,...]

Figure 4: VITAFRAME with column and metadata schema specification. The Review column is of type Text and the metadata, the collection of unique tokens, is of type List(token).

To support VITA use-cases, we assign a schema to dataframe columns—each column $C_i \in C$ may have a schema defined over a set of data domains, $D = \{d_1, d_2, \dots\}$ that spans heterogeneous data types like text, visualizations (**D3**). We call this data structure a VITAFRAME. We discuss the underlying data domain for VITA in Section 5. The REPL-style VITA workflows involve users creating and examining intermediate results from the data domain, D. An intermediate result with one-to-one correspondence with a VITAFRAME column (*e.g.*, *n* reviews are featurized into *n* feature vectors) is added as a new column. However, these results may not have one-to-one correspondence (*e.g.*, dictionary of words in the set of *n* reviews) and are stored in a separate data structure as metadata of the corresponding column. Therefore, for each column $C_i \in C$ in a VITAFRAME, there is a schema specification function that assigns a domain $d_j \in D$ to the column and a domain $d_k \in D$ for each column metadata (see Figure 4). Metadata is often used for computing aggregate statistics and visualizations.

4.2 Computation Model

VTA **Compiler.** In Section 5, we present VTA, an algebra for specifying VITA operations. LEAM compiles the user interactions on Operator View (see in Figure 1a) or VTA commands in Notebook View (see in Figure 1d) into VTA specifications. However, the VTA specifications are incomplete, in the sense that they may omit details ranging from visual encoding such as fonts, line widths to input data type. To resolve these ambiguities LEAM currently uses a rule-based compiler that translates a VTA specification into lower-level operators for the execution engine to run backend computation (see Section 5).

Execution Engine. The VITA execution engine takes the following input generated by the VTA compiler: (1) input data schema in D and (2) translated VTA specifications. The text analysis operations—data preprocessing, featurization, feature transformation/selection), estimation), and more advanced post-processing operations like anomaly detection—are mapped to existing ML and NLP libraries like Spacy, Scikit-learn. While other VTA operators related to visual coordination are mapped to built-in implementations (see Section 5).

Version Control. After each operation, LEAM checkpoints the current state of the visualizations, VITAFRAME, and notebook commands. The development of a fully functional system for fine-grained (*i.e.*, at operation level) versioning is currently in progress (**D5**). We discuss how to support fine-grained version management of a VITA session involving heterogeneous data types, notebook commands, and user interactions in Section 6.

4.3 LEAM Front End

LEAM user interface has four components—Operator View, Table View, Visualization View, and Notebook View (see Figure 1)—which enables users to perform in-place text analytics (**D1**). Users can perform various VITA operations using the operators in Operator View (see Figure 1a). For example, cleaning the data in Table View, adding visual summaries in Visualization View. Table View (see Figure 1c) design is inspired by traditional spreadsheets and tabular data visualization tools [27] and enables users to directly operate on the data. Table View data can be transformed into visual summaries like bar charts and scatterplots using the visualization operators or VTA commands (see Figure 1). The visualizations in Visualization View (see Figure 1b) are displayed as a carousel of charts. Interactive visualizations are generated by translating the visualization operators or commands to Vega-Lite specifications [25]. Notebook View design (see Figure 1d) is inspired by computational notebooks and enables users to author VITA workflows using a Python-based VTA library. The different views in LEAM can be linked—interactions on one view are reflected in other views (**D2**). Through VTA, users can declaratively specify interactive coordination (*e.g.*, brush-and-linking) between these views that are translated to Vega signals [24].

5 VTA: AN ALGEBRA OF VITA

We now discuss our visual text algebra, VTA, in detail. We demonstrate how VTA captures various tasks in the usage example in Section 2 (see Figure 1, 5, and 6).

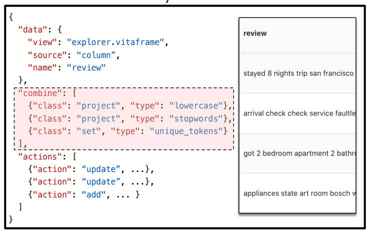
5.1 VTA Specification

Data domain. VITA involves many data types, including different forms of text (*e.g.*, words, tokens, sentences), complex data types like lists, vectors, and even visualizations. We define the data domain of VITA as $D = \{P, S\}$, where P and S are sets of primitive and synthesized (*i.e.*, composite) data types. The domains of primitive data types are taken from, $P = \{\alpha^*, \textbf{int}, \textbf{float}, \textbf{bool}, \textbf{datetime}, \textbf{List}, \textbf{Vector}, \textbf{Dictionary}\}$. The domain α^* is the set of finite strings over an alphabet α. The domains of composite data types are taken from, $S = \{$**Text**, **Visualizations**$\}$. If a schema of the data is not specified upfront, the data is initially assumed to be from the domain α^*. Each domain $d_i \in D$ includes a generator function $g_i : \alpha^* \rightarrow d_i$ which defines the rule for inferring exact data types of the respective domain. For example, the composite **Text** data types (*e.g.*, words, tokens, sentences) are generated using a context free grammar [9]. The generator function of visualization data types is defined based on Vega-Lite [25]: **Visualizations** $= (data, transforms, mark\_type, encodings)$. VTA operators can be specified in a *json* or as declarative commands that are available through a VTA library in Python. The currently implemented VTA operators are listed in Table 1.

Class	Operator	Example
Selection	selection	*select, filter*
Transformation	project	*lowercase, remove_punctuation, remove_stopwords, PCA, remove_emoji, strip_html, remove_url, correct_spellings*
	mutate	*tokenize, tf_idf, k-means, get_sentiment, predict*
	aggregate	*count_tokens, word_scores*
	set	*get_unique_values*
	visualize	*barchart (horizontal, vertical, stacked), scatterplot, heatmap, line chart*
Composition	combine	*combine*
	synthesize	*synthseize*
	udf	*add, apply*
Coordination	internal	*set_selection_type*
	external	*uni_link, bi_link*

Table 1: List of VTA **operators.**

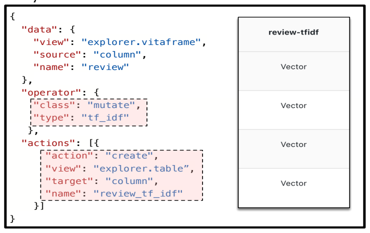
Selection operator. VTA enables the specification of interaction through selections. Selection operations select data points of interest on which subsequent operations in the workflow may be performed (*e.g.*, row(s) in Table View, visualization mark(s) in Visualization View). Supported selection types include a single point (*e.g.*, a table row, marks like a bar or a circle in a chart), a list of points (*e.g.*, rows, bars, or circles), or an interval of points (*e.g.*, ten rows starting from row *i*, circles in scatterplot within *x*-axis range). The selection criteria are specified by a predicate to determine the set of selected points. `Filter` is an example of list selection where the selection



```
{
  "data": {
    "view": "explorer.vitaframe",
    "source": "column",
    "name": "review"
  },
  "combine": [
    {"class": "project", "type": "lowercase"},
    {"class": "project", "type": "stopwords"},
    {"class": "set", "type": "unique_tokens"}
  ],
  "actions": [
    {"action": "update", ...},
    {"action": "update", ...},
    {"action": "add", ... }
  ]
}
```

review
stayed 8 nights trip san francisco
arrival check check service faultle
got 2 bedroom apartment 2 bathr
appliances state art room bosch w

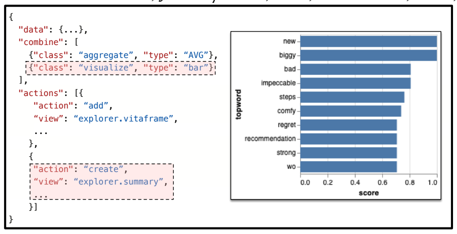
(a) Clean and generate metadata



```
{
  "data": {
    "view": "explorer.vitaframe",
    "source": "column",
    "name": "review"
  },
  "operator": {
    "class": "mutate",
    "type": "tf_idf"
  },
  "actions": [{
     "action": "create",
     "view": "explorer.table",
     "target": "column",
     "name": "review_tf_idf"
  }]
}
```

review-tfidf
Vector
Vector
Vector
Vector

(b) Create TF-IDF feature vector



```
{
  "data": {...},
  "combine": [
    {"class": "aggregate", "type": "AVG"},
    {"class": "visualize", "type": "bar"}
  ],
  "actions": [{
     "action": "add",
     "view": "explorer.vitaframe",
     ...
   },
   {
     "action": "create",
     "view": "explorer.summary",
     ...
   }]
}
```

(c) Visualize top-words by TF-IDF score

Figure 5: VTA **specification. (a) Create a composite cleaning operator (`lowercase` and `remove_stopwords`) using `combine` and generate metadata (`unique_tokens`). (b) Create TF-IDF vectors from reviews using `mutate` operator (added as a new column in Table View). (c) Create a bar chart by combining `aggregate` (average TF-IDF score for each token) and `visualize` operator.**

predicate is the filtering condition. VTA also supports similar types of selections on visualizations (Figure 6a). Besides *json* specification, users can also perform such selections by writing commands in Notebook View. For example, Figure 1F shows how a user can select a bar (*e.g.*, the word "comfy") in the bar chart using a VTA command:

Transformation operators. While developing LEAM, we identified a set of core transformation operators that encompasses the various VITA workflows. These transforms manipulate the components of the selection they are applied to. Note that the core operator set is minimal, and there is room for adding more operators to make VTA more expressive (**D4**). The transformation operation has five subclasses: `project`, `mutate`, `aggregate`, `set`, and `visualize`. The `project` operators change the dimensionality (*e.g.*, LDA, PCA) or cardinality (*e.g.*, stopword removal) of the input data or update the content (*e.g.*, lowercase). The `mutate` operator generates a new representation of the input data (*e.g.*, creates a list of tokens or feature vectors from text). The `aggregate` operator computes summary statistics of the input data (*e.g.*, average review length in a corpus). The `set` operations enable set-like operators on the input data (*e.g.*, get unique tokens in the corpus). The `visualize` operator generates visualizations of data. Figure 5 captures the data preprocessing phase of the workflow discussed in Section 2. Ada first cleans the reviews using `project` operators (Figure 5a),then applies a `mutate` operator (*e.g.*, TF-IDF feature vector creations) to featurize the reviews (Figure 5b). Finally, Ada computes the average TF-IDF score of each word (`aggregate`) and visualizes the top ranking words (`visualize`) using a bar chart (Figure 5c).

Composite Transforms. VTA currently supports two composite transforms that combine unit transformation operations: `combine` and `synthesize`. The `combine` operator enables users to specify an operation pipeline. In Figure 5a, a user combines two `project` operations with a `set` operation into a single operation. Similarly, in Figure 5c, a user combines `aggregate` and `visualize` operators into a single operation for bar chart creation. The `synthesize` operator enables users to create new operations from these combinations which can be reused later. For example, a user can `synthesize` the previous cleaning pipeline to be a `clean` operator which then becomes an operation in the *Operator View* and is used later.

5.2 Multi-view Coordination

So far, we have explained selections that are defined within a single view. However, selections that involve multiple views cannot be captured by a single-view-based specification. We define a coordination operator called `coordinate` that captures multi-view coordination. For example, in Figure 6a, selecting a top-word bar in the bar chart filters relevant reviews in Table View (Visualization View → Table View coordination). The `coordinate` operator needs to also resolve the mapping between views and composite selections across views.

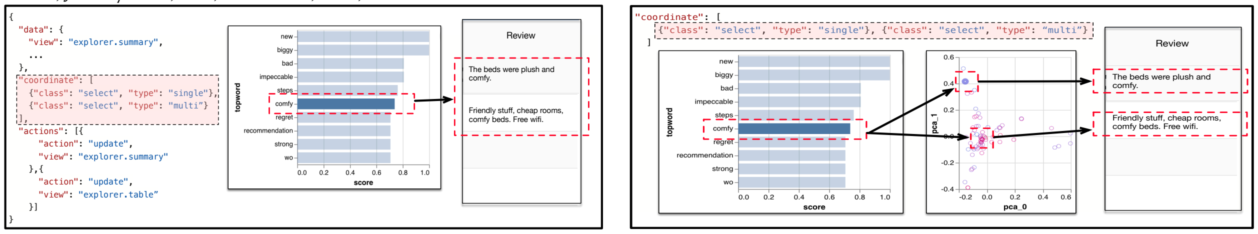
Mapping coordination. In Figure 6a, there is a one-to-many mapping from the bar chart to Table View—selecting a bar filters two that reviews contain the word in "comfy". In Figure 6b, there is a one-to-many mapping from the bar chart to scatterplot and a one-to-one mapping from the scatterplot to Table View. Therefore, the `coordinate` operator needs to resolve the mapping among views, so that relevant visualization marks are selected/highlighted in respective views. Users can specify the mapping using the *type* tag in respective `select` operators of the views. For example, in Figure 6a, the user selects type *single* for the bar chart and *multi* for Table View.

Resolving composite selections. Users can add multiple single-view selections to a view. For example, in Figure 6b, the user adds a link from the bar chart to the scatterplot, which creates a multi-view coordination among the bar chart, scatterplot, and Table View. Selecting a bar in the bar chart highlights multiple circles (scatterplot) and reviews (Table View). However, following the top-word selection, the user may select a rectangular area in the scatterplot or select multiple reviews in the table. Currently, LEAM only supports independent selection. We outline advanced composite specifications such as `union` and `intersection` as future work.

6 LEAM ENHANCEMENTS

We now outline our vision for LEAM development.

Efficient storage model. While VITAFRAME currently supports data that fits in memory, the datasets will be larger in practice. So, the storage layer of LEAM can enable operations on both in-memory and disk-resident data— an essential requirement for scalability. We can leverage scalable dataframes such as *Modin* [21] that allows dataframes to exceed memory limitations. The extension would require integrating the VTA library in *Modin* and adapting its column definition to include metadata schema. Another alternative is embedded analytical systems that tightly integrate RDBMS with analytical tools



```
{
  "data": {
    "view": "explorer.summary",
    ...
  },
  "coordinate": [
    {"class": "select", "type": "single"},
    {"class": "select", "type": "multi”}
  ],
  "actions": [{
      "action": "update",
      "view": "explorer.summary"
    },{
      "action": "update",
      "view": "explorer.table”
    }]
}
```

Review

The beds were plush and comfy.

Friendly stuff, cheap rooms, comfy beds. Free wifi.

```
"coordinate": [
   {"class": "select", "type": "single"}, {"class": "select", "type": “multi”}
 ]
```

(a) Coordination of two views (b) Coordination of three views

Figure 6: Multi-view coordination. (a) Use the `coordinate` operator to link the bar chart and Table View. Selecting a bar (word) in the bar chart triggers a `filter` by the selected word on Table View . (b) Coordinating the bar chart and scatterplot links the three views. Selecting a bar in the bar chart highlights relevant points in the scatterplot, and filters relevant rows in Table View.

and provide fast and efficient access to the data stored within them [22]. LEAM employs PostgreSQL for session management and version control. Designing a storage layer that enables efficient data sharing between VITA sessions and the RDBMS is a possible research direction.

Interaction at scale. LEAM should provide interactive responses even with larger datasets. One approach is to draw samples of data progressively and then display approximate results. Approximate query processing techniques can support operations like `aggregate` [1–3] and `visualize` [11, 23]. However, providing meaningful intermediate results progressively for operations like classification or clustering is challenging—how to determine model meta-parameters without scanning the complete data? Progressive computation can be complemented by *optimistic analytics* [20], where precise computations run in the background as users explore approximate results. When there is a significant difference between the approximate and precise results (*e.g.*, classification results vary from ground truth), the analyst can decide which parts of the exploration have to be redone. Larger datasets also impede direct data manipulation and necessitate the design of interactive and navigable representation of Table View [5].

Versioning VITA **sessions.** The LEAM versioning system can maintain a version graph to keep track of fine-grained changes at the unit operation level. LEAM needs to consider the storage-latency trade-off as a user adds new nodes to the graph: storing entire data ensures faster session reconstruction at the cost of storage while storing delta between subsequent sessions reduces storage overhead at the cost of increased reconstruction time. Designing a fine-grained version control system for LEAM offers unique research challenges—besides VITAFRAME, LEAM also needs to checkpoint: (a) the states of all the front-end components (*e.g.*, formatting like font, color, the opacity of views), (b) the coordination mappings of views and composite selections, (c) user-defined operator pipelines and custom models, and (d) VTA commands in Notebook View. Existing systems address some of these challenges in isolation (*e.g.*, data [12] and model [18] versioning, workflow debugging [6, 19]).

VTA**: coverage, accessibility, and automation.** Our goal is to increase LEAM's coverage of VITA workflows [17] by introducing new VTA operators and adding popular ML and NLP libraries [26] as default operators in Operator View. We can further improve LEAM's extensibility by enabling users to add their custom-built models as new operators in Operator View and reuse them later. To make VTA more accessible to a wider audience, we are working on integrating VTA with an interactive widget that allows users to issue VTA operators from Jupyter notebooks. Other goals include automatically generating VITA workflows given an analysis goal [4], recommending next operator based on users' current workflow [33], and training an autoregressive language model like *GPT-3* [7] on VTA to automatically compose coordinated views or LEAM-style user interface based on user specifications in natural language.

VITA **workflow optimization.** Operator pipelines in a VITA workflow may be executed in various orders. For example, text reviews can be tokenized first and then cleaned or the other way around. However, tokenizing a cleaned text is less expensive due to it's smaller cardinality than the original text. We can leverage VTA to design a VITA workflow optimizer, similar to a query optimizer in databases. Other approaches for workflow optimization can be to introduce parallel execution similar to *Modin* [21], enabling distributed processing of partitions of a VITAFRAME to speed up computation.

Evaluation of LEAM**.** While we demonstrate the expressivity and on-demand workflow authoring capabilities of VTA with several usage examples, we did not report the performance of other LEAM components. Future iterations should include experiments that evaluate the storage model's efficiency in managing large datasets, storage overhead and responsiveness of the versioning system, performance of VTA query optimizer, and overall interactivity of LEAM. Moreover, user studies should be conducted to evaluate the usability of LEAM.

7 RELATED WORK

We now summarize existing work on VITA, data management for data science, and domain-specific algebra design.

Visual interactive text analytics. VITA systems employ visualization techniques—both basic (*e.g.*, scatterplot, line chart, treemap) and complex (*e.g.*, word cloud, steam graph, flow graph) [17]—to numerous uses-cases like review exploration [34], sentiment analysis [14], text summarization [8]. While these earlier works highlight the appeal of integrating interactive visualization with text analysis, they lack the generalizability of LEAM, where users can employ an expressive algebra to author different VITA use-cases within a single system.

Data management for VITA**.** Prior work focuses on designing systems for scalable computation (*e.g.*, scalable dataframe and query/operator optimizers [21], caching and prefetching for visualization [30]), storage models for efficient data access [22, 31]. We discussed related work on versioning [6, 12, 18, 19], approximate query processing [1–3, 11, 23] in earlier sections. Leam builds on the earlier work with specific focus on developing an efficient storage model, enabling scalable computation, and performing fine-grained version control.

Data model and algebra. Our work takes inspiration from existing algebras that provide well-founded semantics for relational databases [10], dataframes [13, 21], and interactive visualizations [24, 25, 29]. Here we introduce a new grammar for visual text data analytics and interactive view coordination, building on earlier work. To the best of our knowledge, VTA is the first algebra defined for VITA.

8 CONCLUSION

This paper presents our vision for Leam, an integrated system that supports VITA workflows end to end. Leam is designed based on several design considerations that we derived by identifying existing challenges in developing VITA systems. Leam enables users to perform interactive text analysis in-situ—direct data manipulation (Table View), REPL-style analytics (Notebook View), and coordinated visual data exploration (Visualization View). We introduce a novel algebra for visual text analysis, VTA, that provides a suite of operators to author any VITA workflow on-demand and enable different modes of coordination among views. We present our current progress in developing Leam's underlying data management system and outline several research directions related to extensibility and coverage of VTA, and storage, computation, and versioning of data and VITA workflows. Addressing these challenges requires interdisciplinary research efforts from DB, NLP, HCI, and VIS communities.

REFERENCES

[1] Acharya et al. The aqua approximate query answering system. In *ACM SIGMOD*, pages 574–576, 1999.
[2] Agarwal et al. Blinkdb: queries with bounded errors and bounded response times on very large data. In *ACM EUROSYS*, pages 29–42, 2013.
[3] Babcock et al. Dynamic sample selection for approximate query processing. In *ACM SIGMOD*, pages 539–550, 2003.
[4] Bar et al. Automatically generating data exploration sessions using deep reinforcement learning. In *ACM SIGMOD*, pages 1527–1537, 2020.
[5] Bendre et al. Faster, higher, stronger: Redesigning spreadsheets for scale. In *ICDE*, pages 1972–1975. IEEE, 2019.
[6] Brachmann et al. Your notebook is not crumby enough, replace it.
[7] Brown et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
[8] Carenini et al. Interactive multimedia summaries of evaluative text. In *IUI*, pages 124–131, 2006.
[9] E. Charniak. Statistical parsing with a context-free grammar and word statistics. *AAAI/IAAI*, 2005(598-603):18, 1997.
[10] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.
[11] Hellerstein et al. Online aggregation. In *SIGMOD*, pages 171–182, 1997.
[12] Huang et al. Orpheusdb: Bolt-on versioning for relational databases. *PVLDB*, 10(10), 2017.
[13] Hutchison et al. Laradb: A minimalist kernel for linear and relational algebra computation. In *ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*, pages 1–10, 2017.
[14] Kucher et al. The state of the art in sentiment visualization. In *Computer Graphics Forum*, pages 71–96, 2018.
[15] Lajus et al. Efficient data management and statistics with zero-copy integration. In *SSDBM*, pages 1–10, 2014.
[16] Lee et al. Demystifying a dark art: Understanding real-world machine learning model development. *arXiv preprint arXiv:2005.01520*, 2020.
[17] Liu et al. Bridging text visualization and mining: A task-driven survey. *IEEE TVCG*, 25(7):2482–2504, 2018.
[18] Miao et al. Modelhub: Towards unified data and lifecycle management for deep learning. *arXiv preprint arXiv:1611.06224*, 2016.
[19] Miao et al. Provdb: A system for lifecycle management of collaborative analysis workflows. *arXiv preprint arXiv:1610.04963*, 2016.
[20] Moritz et al. Trust, but verify: Optimistic visualizations of approximate queries for exploring big data. In *SIGCHI*, pages 2904–2915, 2017.
[21] Petersohn et al. Towards scalable dataframe systems. *PVLDB*, 13(11):2033–2046, 2020.
[22] Raasveldt et al. Data mgmt for data science-towards embedded analytics.
[23] Rahman et al. I've seen" enough" incrementally improving visualizations to support rapid decision making. *PVLDB*, 10(11):1262–1273, 2017.
[24] Satyanarayan et al. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE TVCG*, 22(1):659–668, 2015.
[25] Satyanarayan et al. Vega-lite: A grammar of interactive graphics. *IEEE TVCG*, 23(1):341–350, 2016.
[26] Smith et al. The machine learning bazaar: Harnessing the ml ecosystem for effective system development. In *ACM SIGMOD*, pages 785–800, 2020.
[27] Spenke et al. Focus: the interactive table for product comparison and selection. In *ACM UIST*, pages 41–50, 1996.
[28] Statista. Retail e-commerce sales worldwide from 2014 to 2023, Mar. 2020.
[29] Stolte et al. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE TVCG*, 8(1):52–65, 2002.
[30] Tao et al. Kyrix: Interactive visual data exploration at scale.
[31] TileDB Core Team. Tiledb: The universal storage engine, 2020.
[32] Wang et al. Guidelines for using multiple views in information visualization. In *Proc. AVI*, pages 110–119, 2000.
[33] Yan et al. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In *ACM SIGMOD*, pages 1539–1554, 2020.
[34] Zhang et al. Teddy: A system for interactive review analysis. In *SIGCHI*, pages 1–13, 2020.