A Bidirectional Pipeline for Semantic Interaction in Visual Analytics

Adam Q. Binford

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

Chris L. North, Chair

Denis Gracanin

Nicholas F. Polys

August 11, 2016

Blacksburg, Virginia

Keywords: Visualization, High-dimensional data, Interaction design

A Bidirectional Pipeline for Semantic Interaction in Visual Analytics

Adam Q. Binford

ABSTRACT

Semantic interaction in visual data analytics allows users to indirectly adjust model parameters by directly manipulating the output of the models. This is accomplished using an underlying bidirectional pipeline that first uses statistical models to visualize the raw data. When a user interacts with the visualization, the interaction is interpreted into updates in the model parameters automatically, giving the users immediate feedback on each interaction. These interpreted interactions eliminate the need for a deep understanding of the underlying statistical models. However, the development of such tools is necessarily complex due to their interactive nature. Furthermore, each tool defines its own unique pipeline to suit its needs, which leads to difficulty experimenting with different types of data, models, interaction techniques, and visual encodings. To address this issue, we present a flexible multi-model bidirectional pipeline for prototyping visual analytics tools that rely on semantic interaction. The pipeline has plug-and-play functionality, enabling quick alterations to the type of data being visualized, how models transform the data, and interaction methods. In so doing, the pipeline enforces a separation between the data pipeline and the visualization, preventing the two from becoming codependent. To show the flexibility of the pipeline, we demonstrate a new visual analytics tool and several distinct variations, each of which were quickly and easily implemented with slight changes to the pipeline or client.

Acknowledgments

I want to thank my advisor Chris North taking me on and guiding me through my work. I cannot imagine a more enjoyable advisor to work with. I thank Nicholas Polys for inspiration on my design, and Denis Gracanin for being a great professor for me through numerous classes over the years. I would also like to thank Michelle Dowling and Jagathshree Suryanarayanan Iyer for collaborating with me on some of the web interfaces.

Finally, I would like to thank General Dynamics for their support in this research.

Contents

List of Figures

Chapter 1

Introduction

Visual data analytics tools are made to support the user's sensemaking process by coupling human reasoning with the analytical power of computerized mathematical models [13]. Thus, to be effective, visual analytics tools must transform raw data to a visualization that assists the user in the basic sensemaking tasks of foraging for more data and synthesizing information [16]. Semantic interaction (SI) is one approach for supporting the sensemaking process by improving the usability of the underlying models in visual analytics tools [16, 17, 22, 25]. These tools use visual metaphors, such as a "near = similar" metaphor, to map the data to the visualization. These metaphors enable the user to interact with the displayed data in an intuitive manner. Each interaction is translated into feedback for the underlying models driving the visualization, prompting the visualization to automatically update based on this new information. Interpreting natural interactions rather than forcing users to directly manipulate mathematical parameters eliminates the necessity for users to understand how

the underlying mathematical models function. By continuously processing user feedback and updating the visualization, tools that employ semantic interaction enable the tool to progressively learn about the user's interests within the data. Therefore, with each interaction, the system can provide incrementally better results that more closely match the user's interest [16].

One type of semantic interaction is observation-level interaction (OLI) [17]. Here, the focus is on users interacting with the visualized data points (observations) to update the parameters used to generate the visualization. OLI comes in two distinct varieties: Visual to Parametric Interaction (V2PI) [23, 25] and Bayesian Visual Analytics (BaVA) [22]. BaVA is probabilistic, relying on an assumption of "a sampling distribution for the observed data and an uncertainty over the model parameters" [17]. In contrast, V2PI is a deterministic version of OLI and is the primary focus in our current research. As Endert et al. explain in [17], this method can be used in conjunction with the "near = similar" metaphor where users can express knowledge or test hypotheses by dragging points within the visualization. When the user interacts with the visualization, this triggers an inverted computation of the spatialization algorithm in order to determine what parameters for the algorithm produce that layout. Once found, the system can record this set of parameters for future use and rerun the spatialization algorithm to redisplay the data given these new parameters. The resulting visualization gives the user immediate feedback on the given interaction, including what data is similar or dissimilar to the moved data points. This continues to shield users from the needing to understand these underlying mathematical models while still enabling

the user to alter these models and gain new insights.

While tools using V2PI have been developed [8, 16, 32], each has been designed to analyze a specific type of data, using distinct visual metaphors and mathematical models. For example, in [32], a Weighted Multidimensional Scaling (WMDS) algorithm is used to visualize raw numerical high dimensional data. Additionally, only a small, fixed set of data is visualized, with no method of foraging for new data. Conversely in [8], text data is analyzed using a Force Directed Layout in [8] and uses a multi-model approach to maintain thousands of data points which can be foraged through. However, it is still unable to scale to the level of millions or billions of data points. This work was later expanded to connect with Bing and work with larger datasets [9]. However, the tool does not directly handle millions of data points. All interactions must be translated to text queries to outside services. While this is likely a necessary step in real world scenarios, not controlling this external service limits possible experimentation on how to forage through large data sets. Semantic interactions may result in a set of complex term or attribute weights, but translating this to a simple text query results in a significant loss of information with which to forage.

These existing analytics tools lack a method for experimenting with vastly different types of data, mathematical models, and visual encodings. Each implements a specific pipeline to answer a small subset of research questions. This limits the pace at which research into semantic interaction-enabled visual analytics tools can proceed.

To address this issue, we present a flexible pipeline framework for creating prototypes of visual analytic tools which use semantic interaction and V2PI. Our layered multi-model

interactive pipeline structure can be seen in Figure 1.1. We define the method of communication between different models within the pipeline as well as between the pipeline and the visualization. Thus, the models and data are modular components of the pipeline, with the visualization being a separate modular entity outside the pipeline. This allows for simple, separate modifications to the type and source of data being visualized, the models used, and the visual encodings. Different mathematical models can easily be tested with a single visualization. Similarly, different visualizations can be tested with few, if any, changes to the pipeline. To demonstrate the power of this pipeline, we present several examples that demonstrate the simplicity and flexibility of creating new visual analytic tools with this framework. With these different prototypes, we will be able to research different methods for using semantic interaction and V2PI to support the user's sensemaking process.

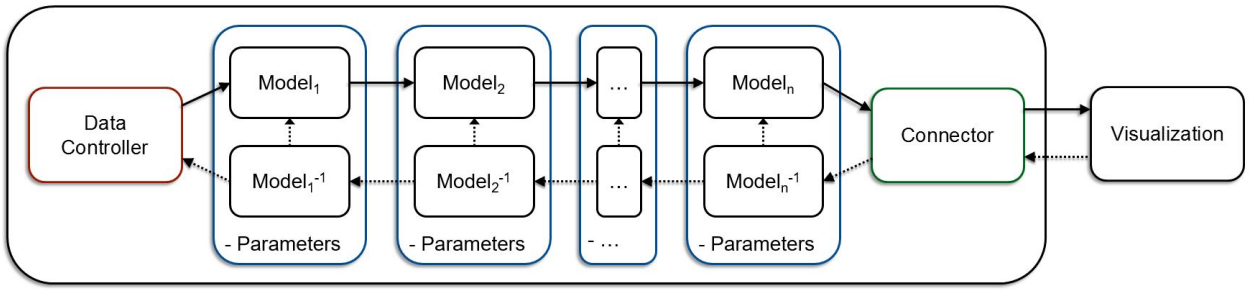

Figure 1.1: The pipeline framework.

Our main contribution is a new framework for quickly creating visual analytics tools that support semantic interaction and V2PI. This framework captures both transforming the raw data to the visualization and interpreting semantic interactions. We exemplify these points through a set of prototypes we developed. Our goal is to accelerate progress in visual

analytics research by providing a platform for semantic interaction research and development. More broadly we seek to expand the traditional visualization pipeline to realize a modern vision of interactive visual analytics [13].

Chapter 2

Related Work

2.1 Semantic Interaction

The sensemaking process was originally defined by Pirolli and Card [27]. As seen in Figure 2.1, it involves several steps. However, these steps can be grouped into two main phases: the foraging loop and the sense-making loop, also know as the synthesis loop. The foraging loop involves discovering new documents, while the synthesis loop involves generating and testing hypothesis from discovered documents. Visualization tools can support this process by using mathematical models to visualize the data. However, in traditional visualizations, interaction is treated as an afterthought, typically involving directly tweaking the parameters of these models, as represented in Figure 2.2. This means that analysts are required to have an expert understanding of how the models work and the meaning of the model parameters.

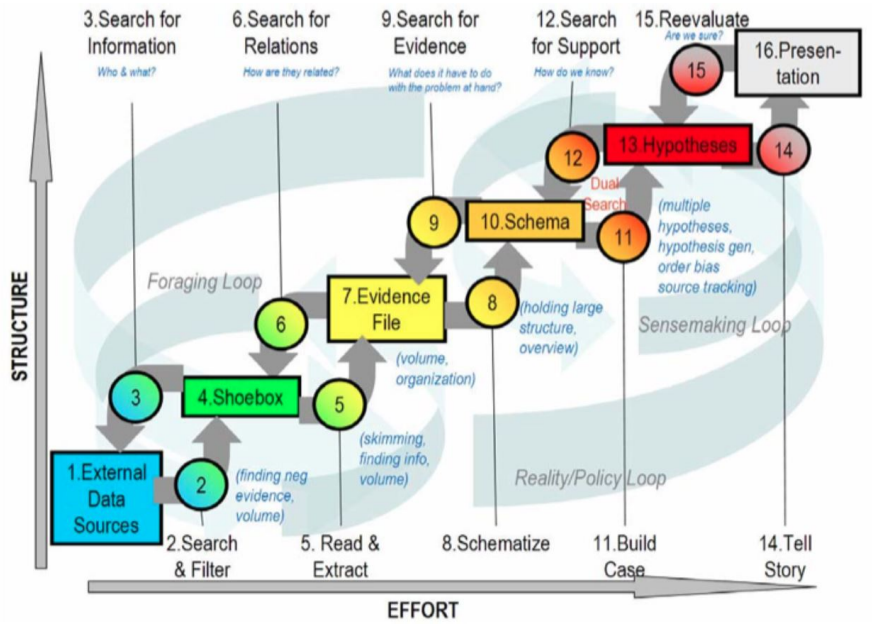

Figure 2.1: The sensemaking loop from [27] illustrates the complex process of turning data into useful insights. Used under Fair Use, 2016.

Semantic interactions address this issue by allowing analysts to alter model parameters by interacting within the visual metaphors [8, 14, 16]. By merging the foraging and synthesizing processes into a unified set of interactions, analysts can test hypotheses, express new knowledge and forage for information without an expert understanding of the underlying models [16]. This means that users do not need to distinguish between foraging and syn-

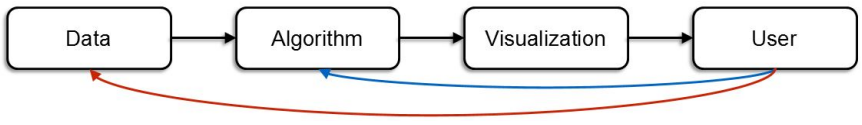

Figure 2.2: The traditional visualization pipeline only allows for direct interaction with the algorithms and raw data.

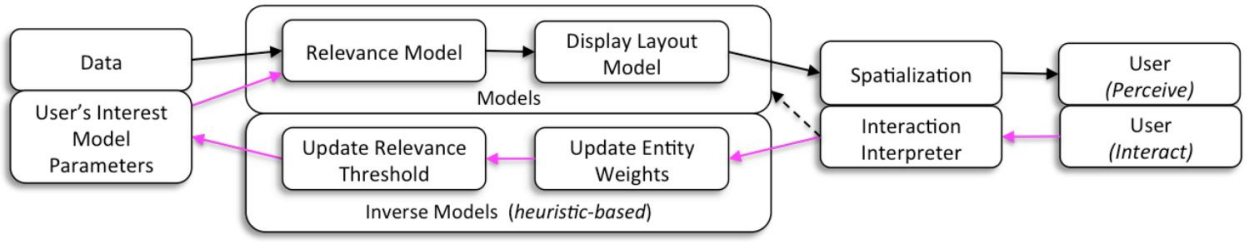

Figure 2.3: The bidirectional pipeline from [8] allows users to interaction directly with the visualization. Used under Fair Use, 2016.

thesizing in their interactions, keeping the sensemaking loop tight. However, when building a tool that enables semantic interaction, the complexity of foraging and synthesis together cannot be captured by a single model. Figure 2.1 illustrates how each of the foraging and synthesis phases may require multiple models.

A multi-model approach to semantic interaction was presented in [8]. Here, Bradel et al. defined a visualization pipeline which uses separate models for the foraging and synthesis processes. Additionally, to enable a feedback loop between the user and the visualization, this pipeline is bidirectional, containing models to transform the data and inverse models to interpret user interactions. This pipeline can be seen in Figure 2.3. After an interaction, the system runs a complete iteration of the pipeline in which each inverse model updates a single set of parameters to describe the user's interest. This implies a codependence between the models. Additionally, the data itself is not factored into this pipeline. It is treated as an initial input that never changes, limiting the types of data that can be visualized and how new data can be brought in. This presents problems for experimentation with different types of data, models, or visualizations. Any changes to a piece of the pipeline

necessitates large changes elsewhere. To remedy this, we are seeking to improve this idea of a bidirectional pipeline. By establishing more formal model entities which contain their own set of parameters, we can reduce codependence between models.

2.2 Streaming Data

Streaming data provides many challenges to data visualization [11, 24]. When adding new data to a visualization, many problems can arise, such as a sudden shift in data scale due to a new maximum or minimum value, or a complete rearrangement of the data on the screen. These issues can disrupt a user's thought process and halt the flow of the sensemaking loop. Additionally, streaming adds a new temporal attribute to the data, and a decision must be made on if and how to convey this temporal component.

Previous research has indicated that scatterplots, which are similar to the similarity based layouts of this research, fare well with streamed data [24]. If the new data points are within the current bounds, very little context is lost on the old set of points. If the new points are outside the current bounds, more significant loss of context may occur, but otherwise the new data does not greatly harm the user's sensemaking process. Unfortunately, this research focused on non-interactive visualizations. In these cases, slight movement of current data points when new ones arrive would not cause much issue to the user. However, once interaction is added to the mix, even slight movement can cause problems. What if a user is trying to select a point, and it moves right as they attempt to select it? Issues like this must

be carefully considered when designing interactive visualizations that support streaming data.

While usability issues are a great concern, enabling streaming data from an architectural standpoint is also nontrivial. Most analytic tools designed to study interactive visualizations do not support true streaming data, where new data may appear in the visualization without a user interaction occurring [8, 9, 16, 32]. Solving the architectural problem is a prerequisite to studying the usability concerns. In this work we provide a solution for easily creating visualization prototypes that support data streaming. By enabling this functionality, we hope to simplify research of the usability problems related to data streaming in interactive visual analytics tools.

2.3 Research Questions

We have generated a list of research questions as a long-term research agenda. The goal of this framework is to provide a platform for creating prototypes and running user studies to answer these questions. Each of the example visualizations created using this framework address one of these issues and can be used to study them in the future. These questions are divided into four main categories: data, models, visualizations, and interactions. For each of these categories, our research questions include:

- Data

  - How can we combine multiple types of data into a single semantic interaction-enabled visualization?
  - How can we support streaming data with semantic interaction?

- Models
  - Which statistical methods would enable semantic interaction with big data in real time?

- Visualizations
  - What visual encodings are most easily understood by users?
  - How can we stream data to the user without causing a loss of context?
  - How can we best use an extra dimension when working with 3D visualizations?
  - How can we combine multiple views in a single visualization?

- Interactions
  - Which interaction methods best support the user's sensemaking process?
  - How can you interact with streaming data that is constantly updating the visualization?
  - How can semantic interaction be extended to 3D or immersive environments?

The limitations of current systems necessitates a method of creating flexible pipelines to answer these kinds of research questions. Although creating a new singular visualization tool

would enable us to address some research questions, it would follow the same pitfall of being inflexible to new changes, prohibiting us from asking any further research questions with that tool. Furthermore, it does not address the issue of the visualization being tightly coupled to the pipeline. This creates a need for a decoupled pipeline that modularizes the data, different models, and the visualization. Separating these concerns enables experimentation with various visualizations or input devices for the same transformed data as well as different methods of transforming the data using the same visualization.

By creating a modular pipeline framework, we enable fast prototyping for visual analytics tools that aid the sensemaking process by using V2PI and other semantic interactions. Thus, we facilitate researching tools that use different data types, models, visualizations, or interactions. We accomplish this by developing the framework with plug-and-play functionality. By modularizing each piece of the pipeline, we can modify the pipeline to test research questions related to data types and models without changing the rest of the pipeline. Additionally, by establishing the pipeline as a separate entity from the visualization, we enable efficient research into different types of visualizations and interactions using the same pipeline. The details of this framework are explained in the following section.

Chapter 3

Pipeline Framework

Several key goals motivated our design of the pipeline framework. We need to allow for multiple models, as previously discussed, without limiting the number of possible models. These sets of models should also be able to operate on multiple types and sources of data. Additionally, pieces of a pipeline should be reusable such that different sets of models can easily operate on the same set of data. These models also need a way to communicate so multiple models can work together for greater functionality, and a suitable language for writing data processing and statistical algorithms must be supported. However, we do not want to limit what languages visualizations can be implemented in, creating a need to separate the visualization from the pipeline. Thus, our framework must be able to communicate with any external program.

Finally, our framework should simplify creating visual analytics tools for large datasets.

Previous works have demonstrated that interactive visualizations cannot scale past tens of thousands of data points while remaining responsive [8, 15, 16]. This is due to the retrieval model having to calculate a relevance for every data point upon each interaction that updates the model, which becomes an expensive process as the size of the dataset grows. While more efficient algorithms, more powerful hardware, and intermediary dimensionality reduction techniques can increase the number of data points that can be handled, there will always be a limit to what can be done in real time. In order to work with datasets expanding into the millions and billions of data points, some processing must be done in the background. The framework should assist developers by hiding as much of the threading issues as possible.

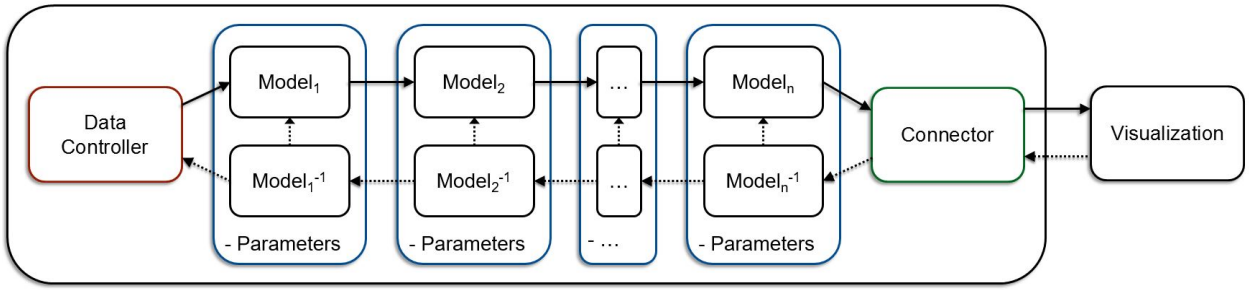

Figure 3.1: An instance of our pipeline consists of a Data Controller, series of Models, and Connector. The visualization is a separate entity from the pipeline. The solid arrows indicate the forward flow in the pipeline, while dotted arrows indicate the inverse flow. Dotted arrows within the Models indicates the ability to short circuit.

These design goals led us to a pipeline made up of three key pieces: a Data Controller, a series of Models, and a Connector. Figure 3.1 illustrates this pipeline and how the pieces fit together. We chose to implement our framework in Python since it is widely used in

data and text processing, with a vast array of available libraries and packages for doing so. Python is also a quick language to prototype with, matching the motivation for this framework. The pieces of our pipeline are fairly simple, but the independence of each module and communication defined between them provides powerful functionality with little effort. This chapter describes the Data Controller, Models, and Connector, how they communicate, and how they were modified to allow for asynchronous behavior. In the following chapter we describe our base implementation of a pipeline for sensemaking visualizations, as well as several variations, to illustrate these different pieces.

3.1 Data Controller

The Data Controller serves as the main access point to the underlying data that is being visualized. Its key purpose is to serve as a method to retrieve the raw data as well as any possible metadata. This can enable users to view the raw data directly or allow the pipeline to pull additional data to process and visualize. Different Data Controllers can be developed to work with different types of data, enabling fast prototyping with different types of data by merely swapping out the Data Controller. The Data Controller can also perform any necessary preprocessing of the data before passing it down the pipeline. For example, a Data Controller could read in raw text documents and calculate a term frequency vector.

3.2 Models

The Models are the next piece along the pipeline. This section can contain multiple Models placed in series between the Data Controller and the Connector, each containing a forward algorithm and an inverse algorithm, and an internal set of parameters. The forward algorithm defines a specific method for how the data is processed as it works its way to the visualization. Thus, these forward algorithms rely on other Models that lie closer to the Data Controller in the pipeline. In contrast, the inverse algorithm of each Model interprets the user's interactions and updates the Model's parameters accordingly. This means that inverse algorithms can receive parameter changes and results from Models that are closer to the Connector. These updates to the Model are then reflected in the output during the next run of the forward algorithm. Parameters in one Model are not directly accessible in others, but a Model can choose to share its parameters using the communication mechanism described below in Section 3.4.

These algorithms are run in response to interactions by the user. An iteration of the pipeline starts by running the inverse algorithm of the Model closest to the Connector. All subsequent inverse algorithms are then run in series, after which the forward algorithms are executed. These forward algorithms use the updated parameters from the inverse algorithms to provide updates to the visualization. After all forward algorithms have completed, the results are sent on to the visualization.

One important feature of the Models is the ability to short circuit the rest of the pipeline.

Short circuiting happens when the inverse algorithm of a Model doesn't need to send the interaction any further up the pipeline. Thus, instead of running the entire pipeline, we short circuit, executing the forward pipeline beginning at the current Model to update the visualization. This is represented by the dotted upward arrow within each Model in the diagrams. This is a key new feature of a multi-model pipeline not found in earlier definitions [8]. There are two main reasons for short circuiting. A Model may interpret an interaction and want to provide immediate feedback for the user without executing any more Models. Additionally, once the pipeline starts to expand and include large datasets, it is infeasible to iterate through the entire pipeline upon each interaction. Models for different scales of data can decide whether they need additional data to fulfill the request or already contain enough relevant data. Speed and response time become limiting factors when working with large datasets, and every possible way of improving this aspect of usability becomes critical.

On the other hand, there is no opposite of short circuiting within the framework, or an arrow downward within a Model from the forward algorithm to the inverse algorithm. The logical flow always transitions from the inverse algorithms to the forward algorithms, never the other way around. There should be no need for this flow, as inverse algorithms should solely be interpreting user interactions. Without a new interaction from the user, there should be no need to execute an inverse algorithm again. Likewise, a forward algorithm should not need to request new data from a forward algorithm earlier in the pipeline, and instead just work with the data it is given. This type of circular communication would greatly complicate the framework and creating new pipeline instances without much foreseeable benefit.

3.3 Connector

The final piece of the pipeline is the Connector. The purpose of the Connector is to mediate messages between the pipeline and the visualization. To do so, the Connector must have a way to receive and respond to three types of messages from the visualization: *Update*, *Get*, and *Reset* messages. The *Update* message signals that the user has performed some type of interaction that warrants an execution of the pipeline, starting with the inverse algorithm of the Model closest to the Connector. An *Update* message may make its way through the entire pipeline, all the way back to the Data Controller, or it may get short circuited by one of the Models along the way. The second type of message is a *Get* message, which is sent in response to a user's request for more information about a specific data point. These are sent straight to the Data Controller for retrieving raw data or metadata directly, which is then returned straight to the visualization. Finally, there is a *Reset* message that signals all the Models and the Data Controller of the pipeline to reset back to their initial state.

The Connector is a very simple piece of the pipeline, yet it allows for greater flexibility in deciding where and how to run the pipeline. With a networked messaging system, the visualization itself could be implemented with any type of language or toolkit that supports the messaging protocol defined in the Connector. Furthermore, the pipeline could be run on a separate computer from the visualization itself. This is another key enabling feature for expanding to large datasets that most other semantic interaction tools don't use [8, 16, 31, 32]. Regardless of the efficiency of the algorithms, there is a limit to the processing speed

and data storage available on personal machines. To account for this, the pipeline could be run on high performance machinery, while the user interacts with the visualization on a personal computer. Offloading the pipeline to these high performance computers will enable interactions with larger datasets not previously feasible.

3.4 Communication within the Pipeline

We have defined the pieces that make up the pipeline, but there needs to be a form of communication between them. This communication is controlled by the pipeline and enables the modularity and flexibility provided by the framework. The various components communicate with each other through a JSON-like data blob maintained by the pipeline. Since the entire pipeline currently exists as a single program, this object is simply passed in memory with Model algorithms implemented as functions operating on this object. Models can choose to interact with one another by operating on the same elements, or keys, within this data blob. To ensure that a pipeline has been properly constructed, each Model specifies requirements for its forward and inverse algorithms. These requirements are specified as keys within the data blob, and are then checked against the output of previous algorithms. If the input requirements of all algorithms are met, then the pipeline is valid. Checking for a valid pipeline prevents unwanted behavior caused by incorrectly constructed Models.

Figure 3.2 illustrates how this communication works. In (a), we have an example Sum Model that takes in a list of values and under the *values* key, calculates their sum, and adds it to

the data blob under the *sum* key. In (b), instead of adding a new key to the data blob, we modify an existing one. The Square Model takes a list of values and squares each of them in place. These are both valid methods for enriching data as it passes through the pipeline, and this same behavior is used for both the forward and inverse algorithms. This allows each Model to contain just the logic required for transforming the data, and not the logic necessary to communicate with other pieces of the pipeline.

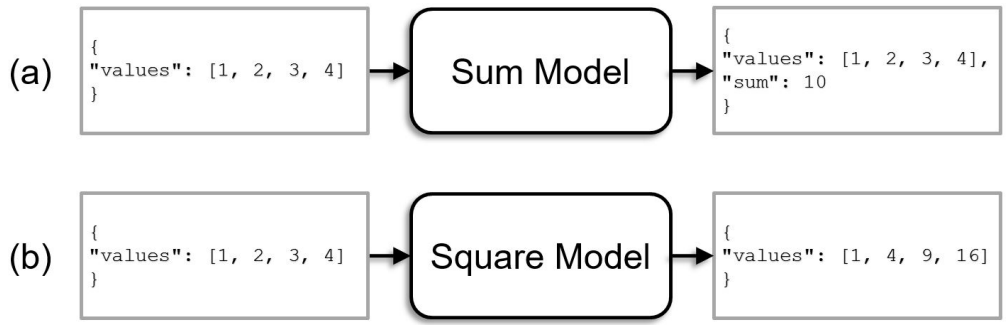

Figure 3.2: Two examples of how a Model can modify the data blob passed through the pipeline. In (a), a new element is added, while in (b), a current element is modified.

With the communication defined, we can see how this framework fits into a model-view-controller (MVC) design pattern [18]. Clearly the Models within the framework directly translate to the model in the MVC design pattern. The Data Controller would also be considered part of the MVC model. The visualization itself acts as the view, and provides the controls to interact with the models. The controller exists as part of the visualization itself providing the interactions, as well as the framework for mediating communication from the visualization to all of the Models and the Data Controller.

3.5 Asynchronous Processing

The presented framework works well for creating simple visualizations that respond to user interactions and work with small, static data sets. However, research into these types of tools is already well covered [8, 14, 15, 16, 23, 32]. What these previous works fail to adequately address are visualizing large data sets and dynamically streamed data [11, 24]. To make meaningful contributions to the study of visual analytics tools, this framework must support these endeavors in some way. This is accomplished through two additions to the framework enabling certain asynchronous behaviors. While multithreading code will undoubtedly need to be included in any complex Models, the approaches taken here aim to reduce as much as the multithreading burden as possible.

3.5.1 Asynchronous Models

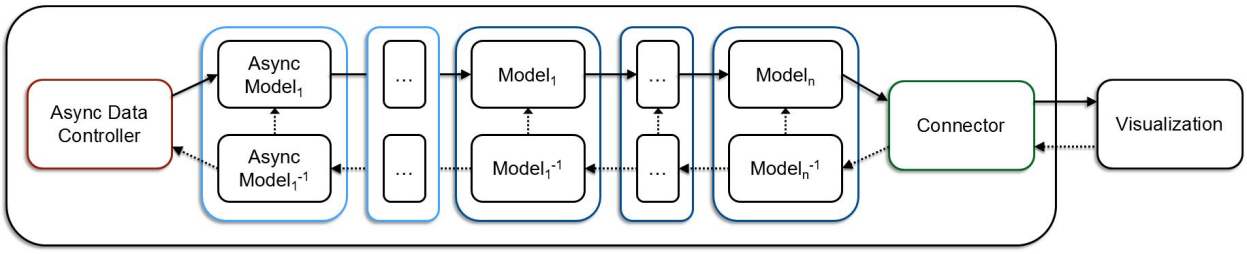

Figure 3.3: A pipeline with asynchronous Models.

In order to provide simple background processing capability, a new asynchronous type of Model was created. The structure of a pipeline using asynchronous models can be seen in Figure 3.3. By grouping a series of asynchronous Models followed by a series of non-

asynchronous Models, we create a simple background threading system. Upon an interaction, the non-asynchronous Models are executed as previously described. When an asynchronous Model is reached along the inverse path, the input data containing the interaction and any new Model parameters are added to a queue. The forward algorithms, starting with the first non-asynchronous Model, are then immediately run, and the visualization is notified of any updates. In a separate thread, the asynchronous Models are run in a similar manner to the original pipeline. Each inverse algorithm is executed in series until the Data Controller is reached or a Model short circuits. The forward algorithms are then run until the first non-asynchronous Model is reached. The forward algorithm of this first Model is run using the data passed down through the asynchronous Models, and the background thread then takes the next item off the queue and iterates again.

This approach enables background processing without requiring any threading code within any of the Models. A lock controls access to the synchronous portion of the pipeline, and the asynchronous part is still run sequentially, preventing any threading issues. While background processing could be done within a specific Model, it would require the developer to write all the multithreading code themselves. While this is still possible, and necessary for complex data processing, the behavior described here can enable simple pipelines to scale up to large sizes of data without requiring any new threading code. The main drawback of this approach is the updates from each interaction may not contain the most up to date data. The synchronous part of the pipeline must return a response back before the asynchronous portion completes, meaning no new data resulting from the asynchronous Models will be

returned in the update. This new data may only be available for the next interaction. But this is a fair tradeoff for requiring no threading code during pipeline development.

3.5.2 Pushing Data

While the previously described technique provides support for larger data sets, it is still limited to responding to user interactions to update the visualization. What if we obtain some new data that is extremely relevant to the user, such as a tweet that was just published or an email that was just intercepted? We would have to wait for the user to initiate some form of interaction with the visualization in order to provide them with this new data. The framework so far described is pull oriented, in that the front-end initiates updates and pulls new data from the pipeline. By enabling pushing, the pipeline can actually update the visualization on demand.

A push can originate from either the Data Controller or from a Model, and is as simple as a single function call. When data gets pushed, it is sent through all or a portion of the forward pipeline, just as data would during an update triggered by a user interaction. Models need not know whether they are processing push-generated data or interaction-generated data. Once the forward pipeline is complete, the results are pushed to the client.

In order for this technique to be possible, the Connector must support pushing data as well. Certain approaches to implementing a Connector may make this impossible, such as those based on RPC that can only respond to outside requests. We will see in the next chapter that

we had to create a new Connector for our data streaming pipeline. Additionally, pushing data involves running the pipeline over multiple threads. While there is a pipeline lock that protects the synchronous portion, the asynchronous models must be sure to include their own locking mechanisms to protect their data. Algorithms of an asychronous model could be run multiple times concurrently due to multiple pushes, or from a push and the asychronous background thread described previously.

Chapter 4

Pipeline Implementations and Visualizations

Here, we illustrate some fully implemented pipelines, along with corresponding visualizations that work with the pipelines. A key goal of the pipeline structure is allowing the visualization and data processing to run on separate machinery. With the recent trends of transitioning from native applications to web applications [21, 34], we saw a great opportunity to move visualization research to the web, and mature web-based graphical frameworks enable complex and detailed visualizations in the browser [5, 12, 36].

Utilizing these frameworks requires a server to serve these web pages. This resulted in the creation of the visualization controller, which together with the served web pages make up a client to the pipeline framework. A server was not implemented within the framework

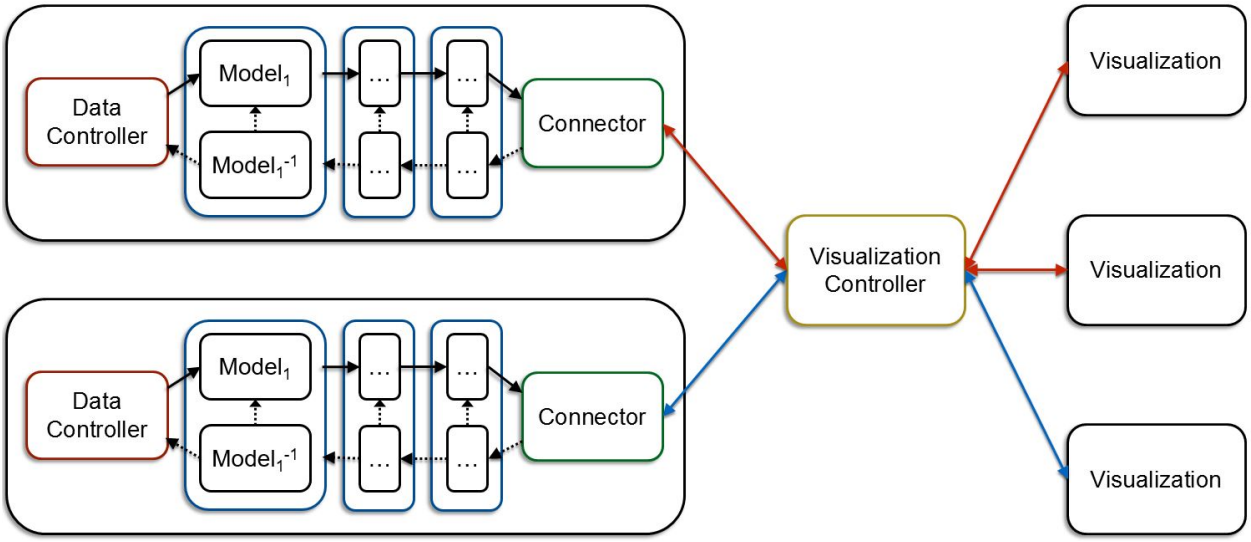

Figure 4.1: The visualization controller mediates communication between multiple pipelines and visualization clients. A single visualization can be mirrored across multiple clients, represented by the red arrows.

itself to maintain generality with the pipeline. While web technologies are the direction we chose to head in, native applications could just as easily interact with the pipeline framework through any networking protocol. The visualization controller has many practical benefits as well. WebSockets [36] allow visualizations to be mirrored to multiple users simultaneously, enabling new topics of research through collaboration [28]. Additionally, the visualization controller does not have to be collocated with the pipeline instances, allowing separate hardware for the web server and pipeine data processing. Finally, in the future it could be used to mediate access to different data sources and visualizations for each user.

Figure 4.1 illustrates this setup. A single visualization controller can serve multiple visualizations to users, and each visualization results in the creation of a new pipeline instance.

The pipeline instance created is tied to the visualization used, as the pipeline outputs must match what the visualization expects. We chose to implement the visualization controller using Node.js [26]. This controller communicates with the pipeline in a couple different ways, described along with the Connectors for each pipeline below.

Adding in this visualization controller does not change how the whole system relates to the MVC design pattern. The visualization controller mostly acts as an intermediary that passes messages between the pipeline and the visualization. The main end points of the controller remain the visualization itself and the pipeline framework.

The rest of this chapter details the pipelines we have created and the visualizations that go along with each pipeline.

4.1 Text Analysis Pipeline

The first pipeline we created aims to aid the sensemaking process with a set of text documents. It combines the relevance-based retrieval model from [8] with the similarity-based layout model from [32]. The structure of this pipeline can be seen in Figure 4.2.

4.1.1 Data Controller

For this pipeline, we created a simple CSV-based Data Controller that loads a preprocessed generic high-dimensional data. To work with text documents, we created a Term

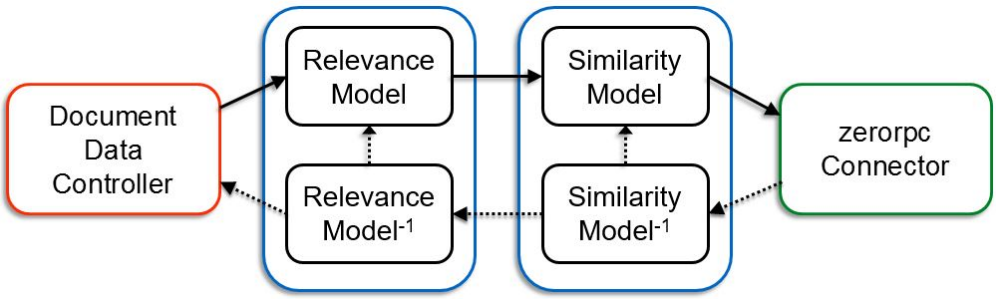

Figure 4.2: Our base pipeline implementation.

Frequency-Inverse Document Frequency (TF-IDF) matrix of our set of documents for this high-dimensional data, where each term represents an attribute or dimension in the data. This numerical data for the text documents allows us to mimic the behavior in [32] to spatialize the data. The TF-IDF metadata is passed down the pipeline to the Models for further processing. Our Data Controller also has references back to the underlying raw text, which the pipeline gives to the user when requested using the *Get* method. For example, a user can open and view documents within the visualization, giving the user direct access to the raw text. This ability to retrieve the raw text is the only part of the entire pipeline that makes it specific to text. The Models described below have no requirement for working with text data, and simply work with numerical high-dimensional data. Later we will demonstrate an alternative use that does not work with text data.

Upon load, this Data Controller loads the data from the specified CSV file, and sends the data down to the Models. Because it is designed to work with small to medium scales of data, it does not interpret any user interactions or updated Model parameters to provide new documents upon an iteration of the pipeline. Instead, it expects the Models in the pipeline

to keep track of the documents after load, and simply listens for *Get* requests to retrieve the raw text of a specific document. Additionally, a different *Get* request can be used to retrieve the names of the attributes, which in the TF-IDF case is the terms representing each dimension.

4.1.2 Models

This pipeline consists of two models: a Relevance Model and a Similarity Model.

Relevance Model

Our Relevance Model draws much of its functionality from [8]. It maintains a set of weights for the attributes in the data, representing how relevant the user thinks each of those attributes are. Interactions such as text queries, node deletion, and increasing the relevance of a document are interpreted by our Relevance Model to make changes in these attribute weights. Additionally, to prevent overwhelming the user by displaying too many data points at once, our Relevance Model also maintains two lists of documents, which we call the *Active Set* and the *Working Set*. The *Active Set* may scale up to thousands of documents that are pulled from the Data Controller, and the *Working Set* contains only the most relevant documents from the *Active Set*. By limiting the *Working Set* to dozens of data points, it remains small enough to be visualized. Maintaining these separate lists in a Model allows the Data Controller to focus on responding to *Get* requests instead of maintaining any parameters

itself.

This Model has two key functions. First, if new documents are passed into the forward algorithm, we insert them all into the *Active Set* and the most relevant ones into the *Working Set*. This relevance is calculated as a dot product between the current attribute weights and the high dimensional attributes for each document. After this, the relevance of all pre-existing documents in the *Working Set* is recalculated. This *Working Set*, along with the relevance for each document, is passed through the forward pipeline.

The inverse algorithm focuses on looking for interactions that will influence our relevance weights. The interactions currently implemented include text queries, changing the relevance of a document, and deleting a document. With text queries, all attributes containing the query are upweighted by a constant. When the relevance of a document is altered by the user, the weights for attributes present in that document are updated based on the magnitude of the change and the values of the attributes in the document's high dimensional data. Finally, deleting documents downweights all attributes present in that document. After any of these interactions, the inverse algorithm uses the new set of attribute weights to search the *Active Set* to see if any new documents should be included in the *Working Set*. If no new documents are found, the interaction and new set of weights are passed along toward the Data Controller to pull more documents into the *Active Set*. Otherwise, this Model short circuits and goes straight to the forward algorithm.

A key aspect is that the Relevance Model does not care about the source of the data. New documents are pulled into this Model from the Data Controller, or other Models closer to the

Data Controller. When the *Active Set* does not contain any new documents to satisfy the request, the interaction continues further inverse Model algorithms or the Data Controller, which can use the interaction to find new documents however it chooses. Additionally, these Models or Data Controller can use the set of attribute weights calculated with the Relevance Model's inverse algorithm for its document finding. This enables interactions such as the changing of a documents relevance to be used by Models that do not interpret such interactions, as the Relevance Model has transformed the interaction into a more usable form.

Similarity Model

The Similarity Model's role is to layout documents according to their similarity, and its logic is based on [32]. This is done using Weighted Multidimensional Scaling (WMDS), a form of dimensionality reduction, on the high dimensional data passed down the pipeline. The Similarity Model stores a set of attribute weights, one for each dimension in the high dimensional data. The forward algorithm uses these weights to project the high dimensional data down to a lower set of dimensions. This dimensionality reduction is performed by optimizing the location of each point in the low dimensional space so that it minimizes the stress between all pairs of points. Stress in this case is defined as the difference between the distance of two points in high dimensional space and in low dimensional space. This optimization is captured by the equation:

$$r = \min_{r_1,...r_n} \sum_{i=1}^{n} \sum_{j>i}^{n} |dist_L(r_i, r_j) - dist_H(w, d_i, d_j)|$$

where r is the low dimensional position of each point, d is the high dimensional position of each point, n is the total number of points, w is the set of weights over the high dimensional space, $dist_L$ returns the distance between two points in low dimensional space, and $dist_H$ returns the weighted distance between two points in high dimensional space.

The inverse algorithm updates this set of weights over the high dimensional space based on OLI interactions within the visualization. This occurs when the user moves data points within the visualization, asserting some knowledge they have that certain data points are either more or less similar than the visualization indicated. The new low dimensional positions of these moved points are then used by an optimization algorithm described in [33] to create a new set of similarity weights to describe the user's layout. This optimization is captured in the following equation:

$$w = \min_{w_1,...w_n} \sum_{i=1}^{n} \sum_{j>i}^{n} |dist_L(r_i^*, r_j^*) - dist_H(w, d_i, d_j)|$$

Here r^* is the new low dimensional position of a point as supplied by the user. This new set of weights is then used on the next forward projection of the data. When this interaction occurs, the Similarity Model short circuits the pipeline to immediately update the visualization based on the new similarity weights.

4.1.3 Connector

For our initial pipeline implementations we used zerorpc to communicate between the visualization controller and the pipelines. zerorpc [38] a Remote Procedure Call (RPC) implementation of ZeroMQ, an asynchronous messaging library for distributed applications [37]. This Connector creates a zerorpc server with RPC bindings for the *Update*, *Get*, and *Reset* messages required of a Connector. The Node.js server then connects as a zerorpc client to the pipelines it creates, establishing communication between the two.

4.1.4 Visualization

For our first prototype, we decided to make a simple web-client using D3, which is a JavaScript library that enables creating interactive visualizations within a web page [12]. As shown in Figure 4.3, this interface's main feature is an interactive graph that allows documents to be displayed as data points. Initially, this graph is empty, requiring the user to search for a term. Searching sends an *Update* message with the query to the pipeline. The results from this query cause the documents to be displayed as data points in the graph using a "near = similar" metaphor. With the graph populated, the user can use OLI, expressing knowledge or testing hypotheses by moving the data points within the visualization. After the points are moved to their desired locations, the user then clicks the *Update Layout* button to send a type of *Update* message to the pipeline along with the coordinates for the moved points. This information is used by the Similarity Model to determine how to replot

Figure 4.3: An initial 2D web visualization showing the general layout of the graph and data fields as well as the visual encodings used in the graph. Points are plotted on the screen using an interactive WMDS algorithm. Moved data points are highlighted in green, while data points that are selected to view the raw text data and metadata are highlighted in purple.

the points. Thus, the user receives immediate visual feedback on the interaction. When the user wishes to return to the interface's initial state, the *Reset* button can be pressed. This causes a *Reset* message to be sent to communicate the user's action to the pipeline, resulting in all Models resetting their data as well.

By double clicking on a data point, the user can view the raw data and some metadata for the document corresponding to that point. This interaction causes the data fields to the right of the graph to be populated using a *Get* message. The Data Controller responds to this message directly without having to run an iteration of the pipeline. The data fields show

the data point's label, the relevance of that document, the raw text from the document, and any notes that the user takes. Although the user cannot interact directly with the text, the label, relevance, and note fields can be updated. Our current implementation only has the pipeline respond to an update in the document relevance, which is handled by the Relevance Model via an *Update* message. Additionally, after a point is double clicked, the user can also choose to delete that node from the graph, indicating that this document is no longer relevant to them. This is reflected in the pipeline through a type of *Update* message, which is also handled by the Relevance Model.

Use Case Scenario

Here we present an example scenario demonstrating the features of this initial prototype. We have a small dataset containing a hidden terrorist plot spread over a few dozen intelligence collections, such as intercepted phone calls and emails. This visualization is initially empty, so we need to have some intelligence to begin our analysis. In our example, we have heard that a man named Mr. Ramazi might be involved in a potential terrorist plot, so we begin by searching for his name. This search returns five documents relating to Mr. Ramazi. After reading through them, we find one document indicating that a Mr. Hallak had withdrawn some money from Mr. Ramazi's account. To remember what this document contains, we rename it accordingly. Figure 4.4 demonstrates the state of our visualization after the initial search about Mr. Ramazi and discovering and renaming the document relating to Mr. Hallak.

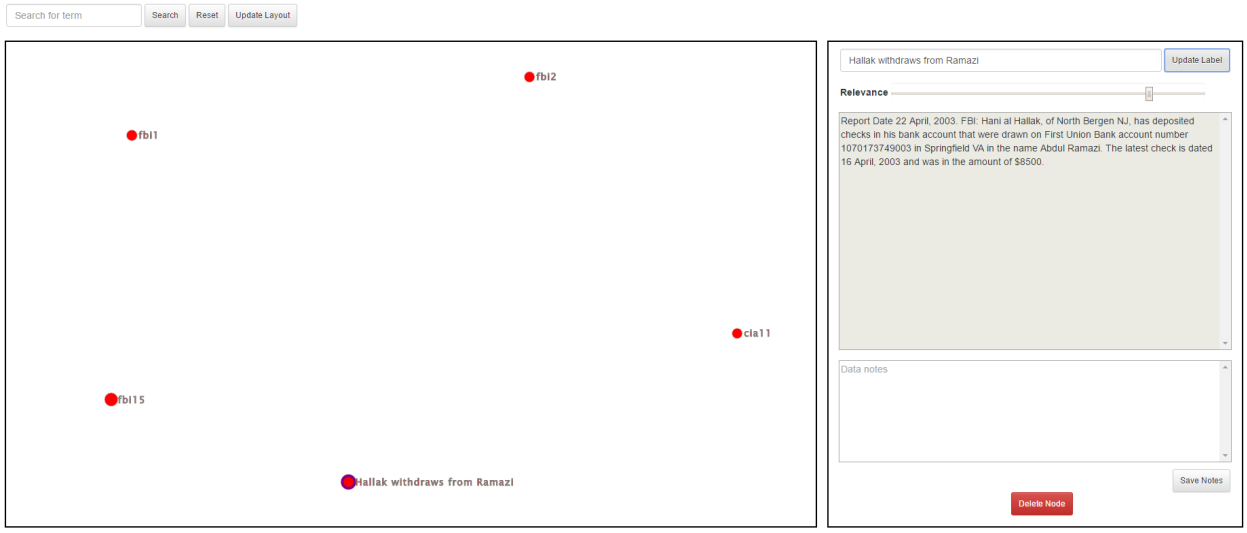

Figure 4.4: The initial search for Mr. Ramazi led us to finding Mr. Hallak has withdrawn some money from him.

After this discovery, we decide we want to learn more about Mr. Hallak. By moving the Relevance slider to the right, we indicate that this document is relevant to us and we want more documents like it. Several new documents appear, one of which contains information about C4 that was found at a shop owned by Mr. Hallak. This helps us confirm that this Mr. Hallak is likely involved in terrorist activities of his own, and might be involved with Mr. Ramazi. We rename this document to remember it's contents as seen in Figure 4.5.

We continue reading the new documents relating to Mr. Hallak and find someone has made phone calls to several numbers including Mr. Hallak from a 718 area code. This may give us some link to others involved in a plot, so we rename this node to remember it's contents and search for more documents relating to the 718 area code, as seen in Figure 4.6.

The search for the 718 number results in a couple new documents appearing on the screen.

Figure 4.5: Mr. Hallak appears to be involved in terrorist activities.

Opening one of them we find another document with phone call information. This time, it is someone making a phone call to this same 718 number. This gives us a series of phone calls that may be connected, but we have no way of knowing if this is coincidental or related to a potential terrorist plot.

We now have examined four different documents, two regarding a list of phone calls and two about Mr. Hallak unrelated to his phone calls. We want to know if there are any other documents on the screen that might provide more insights about these phone calls, so we perform an OLI interaction. By dragging the phone call-related documents together in one corner, and those unrelated to the phone calls in the other corner, we can tell the visualization to emphasize how these pairs of documents are similar and dissimilar when laying out all other points. This interaction can be seen in Figure 4.7.

We execute this OLI interaction by clicking the "Update Layout" button on top. This lays

Figure 4.6: Someone has made phone calls to Mr. Hallak from a 718 number, so we search for more documents relating to this area code.

out the points in a new way, and one of the documents ends up close to the phone number-related documents. This can be seen in Figure 4.8. We find out more about these same phone numbers, including a message relayed to each translated to "I will be in my office on April 30 at 9:00 AM. Try to be on time." While not clear proof of anything, it may be some form of code that we should continue looking into. And we now have several other phone numbers to investigate.

This initial analysis has highlighted the key interactions available within this prototype. The remainder of this chapter explores variations of this initial prototype we have developed.

Figure 4.7: Initiating an OLI interaction to learn more about the phone numbers.

4.1.5 Alternative Visualizations

Here we demonstrate different visualizations that make use of this same pipeline.

Radar Visualization

As an alternative mapping for similarity and relevance in a visualization for data foraging, Ruotsalo et al. propose the Intent Radar [31]. Within this interface, documents are mapped onto the radar based on their relevance to the user's searches and their similarity to each other. More relevant documents will be closer to the center, while similar documents will have a similar angle around the radar. After performing a user study, Ruotsalo et al. found that this new interface enabled users to search through the data more quickly and efficiently than interacting with a list of keywords or traditional query searching.

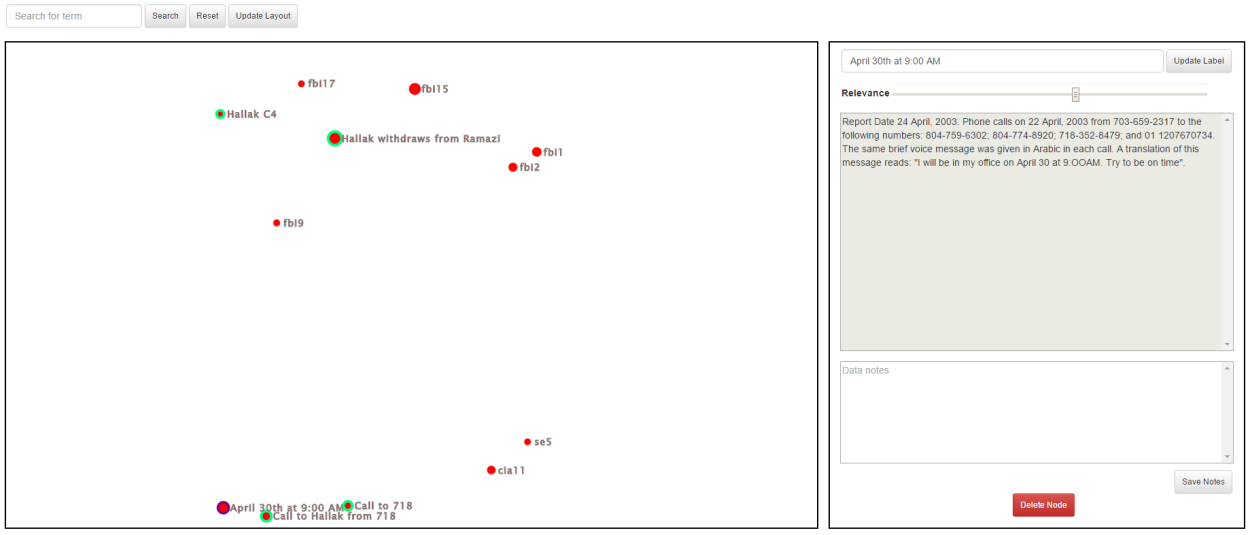

Figure 4.8: The results of the OLI interaction led us to discover another document relating to these phone numbers.

To implement this idea with our pipeline is simple, requiring a new visualization to sit at the end of the pipeline and a simple change to how we use similarity and relevance. Since our pipeline already modularizes similarity, the only necessary change to the pipeline is altering the Similarity Model so that data points are mapped to one dimension instead of two. Our original 2D implementation was adapted to handle the specific mapping of the data of the visualization, translating the relevance metric for each document to the distance from the center and the similarity metric to the angle. This prototype can be seen in Figure 4.9.

3D Visualization

While the richness of 2D web visualization is growing, so also are the capabilities for interactive 3D visualizations. As we explore new sensemaking interfaces and data exploration tools,

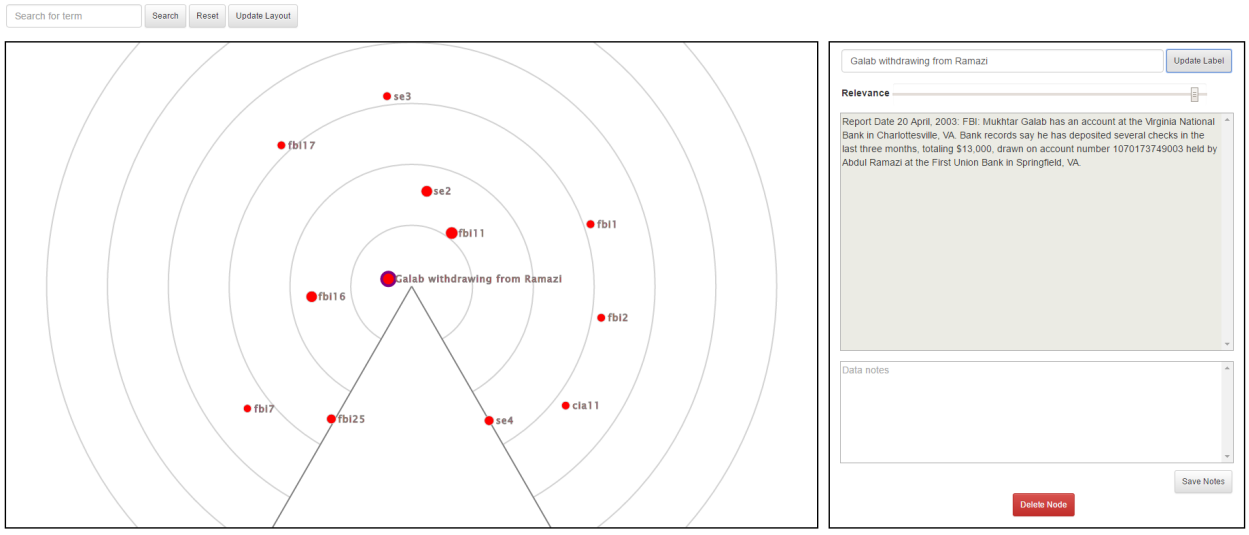

Figure 4.9: An alternative 2D web visualization. This visualization maps relevance to the distance from the center of the radar and similarity to the angle. All other visual encodings are pulled from the original 2D web visualization.

we are interested in the representations and affordances 3D environments can provide. To extend our research to 3D, we used the ideas from our 2D visualization to create a new 3D visualization. We accomplished this by replacing the D3 code in the 2D visualization with X3DOM. X3DOM is a framework developed by the Web3D Consortium that enables the display 3D content natively in the browser. To do so, X3DOM uses Extensible 3D (X3D), the ISO/IEC standard XML format for interactive 3D graphics, as part of HTML5 Document Object Model (DOM) [5]. Since the 3D objects are part of the DOM, they can be manipulated using mouse and keyboard events using HTML5/JavaScript.

The 3D prototype is shown in Figure 4.10, which only differs from the original 2D prototype by plotting the data points in the graph using X3DOM and contains minor changes to the

Figure 4.10: A screenshot of the 3D web visualization. The layout of the web page is based on the layout in the original 2D visualization. With minor changes to the base implementation of the pipeline, we are able to graph the data points represented as spheres in 3D.

CSS. In this new graph, the data points are represented as spheres instead of circles where the radius of the sphere is proportional to the relevance of the document represented by the sphere. To plot the data points in this 3D space, the Similarity Model was configured to project to three dimensions instead of two. This parameter is set from the visualization controller and required no changes in the pipeline itself.

In this new prototype, we also experimented with different interaction options. Instead of double clicking a node to populate the data fields to the right of the graph, we chose to use right clicking. To handle new interactions for exploring the 3D space, there are also buttons to view the scene in multiple angles. These different angles include the Top, Bottom, Front, Back, Left, and Right views. The "Reset View" button resets the view to the default viewpoint. Apart from these, all other interactions are pulled directly from the original 2D

Figure 4.11: A 2D web visualization with attributes mapped in the same graph as the data points. Attributes are yellow, whereas data points are red. All other visual encodings are pulled from the original 2D web visualization.

prototype.

4.2 Displaying Attributes

With our base implementation, we are able to quickly implement new ideas by altering the pipeline in small ways. One example of this comes from a concept introduced by the Data Context Map developed by Cheng et al [10]. The Data Context map provides a method for visualizing the attributes within a given data set alongside the data points using an MDS projection. To accomplish all this, the typical distance matrix used in MDS calculations is augmented with additional data. In standard MDS calculations, the pairwise distances

between data points are calculated to create a square matrix of these distances. Additionally, the Data Context map also calculates matrices for the pairwise distances between attributes, the distances between data points and attributes, and the distances between attributes and data points (i.e. the inverse of the previous matrix). These additional matrices are combined with the original matrix, creating a new square matrix of pairwise distances between all data points and all attributes that is used to visualize attributes in the same space as the data points.

To demonstrate the plug-and-play nature of models in the pipeline, we modified the base implementation to fuse data points and attributes into a single display. Using the approach laid out in [10], we were able to create a new Model that expands the behavior of the Similarity Model in our original pipeline to achieve this behavior. This alteration takes the pairwise distance matrix already generated and computes a composite distance matrix, which includes distances between points and attributes as well as distances between attributes themselves. This composite distance matrix is then treated like any other pairwise distance matrix in the MDS algorithm. We simply specify that point as an attribute so that the visualization can represent it differently than the data points. By defining this behavior in a separate Model, we can easily switch between this new visualization and the original without the attributes. Figure 4.11 shows our original 2D visualization with the addition of these attributes.

The flexible pipeline allowed us to create this fused display extremely quickly. But that is not its only benefit. In addition to simply reimplementing the idea of a composite distance

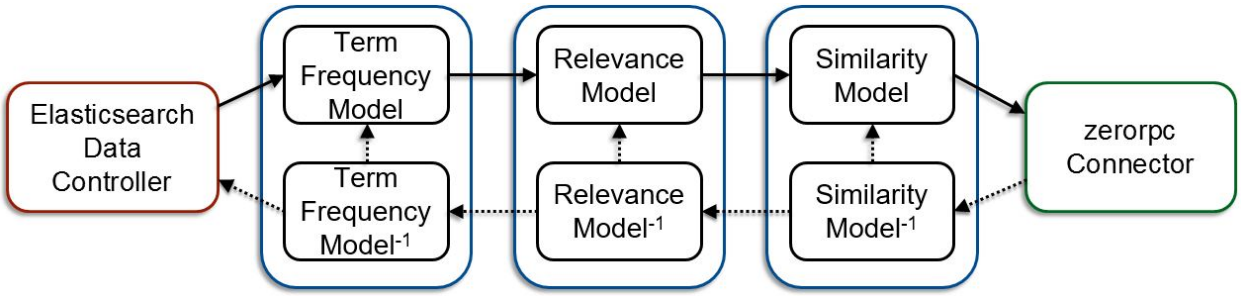

Figure 4.12: A pipeline that connects to an external search engine and calculates term frequencies dynamically.

matrix, the nature of the pipeline makes it interactive. This can greatly aid the data foraging process by helping layout the space. It may be difficult for users to understand why the dimensionality reduction algorithms laid out the data points the way they did, but plotting most relevant attributes appear alongside the related data points may help users with this.

4.3 Multi-source Text Analysis

The examples presented thus far have demonstrated the visual analysis of static data sets. While this can be useful for experimenting with different visual encodings, the types of models used and interactions implemented do not translate well to real world analysis scenarios. Here we present a pipeline for analyzing text documents stored in an external database. The pipeline for this example can be seen in Figure 4.12.

4.3.1 Data Controller

A new Data Controller was created to connect to an external search engine, rather than load local data on startup. In order to have complete control over this search engine, we chose to implement our own basic text search engine using Elasticsearch [19]. After connecting to our Elasticsearch instance, the Data Controller can retrieve documents in two ways. First, through simple text queries. If a text query interaction arrives at the Data Controller, this query will simply be forwarded to the search engine to retrieve new documents. The other method is through a set of attribute weights that have been calculated from one of the inverse Models. In this case, the Relevance Model can interpret interaction such as increasing the relevance of a document within the visualization, to update the set of relevance weights that control which documents are currently displayed. These same set of weights can be used to query the search engine by simply selecting the most relevant words to search for.

By working with an external service, we eliminate the static data limitation. Because we constantly issue new queries to the search engine, we can add new documents into the store while the pipeline is running. Interactions that occur after these documents are added may then pull in these documents into the visualization.

4.3.2 Models

Because all the data is not known ahead of time, little preprocessing can be done. Term frequency data must be calculated dynamically within the pipeline. This requirement led to

the creation of a new Model within the pipeline, the Term Frequency Model. It's behavior is simple: it takes in the raw text for documents and converts them to a normalized term frequency. This is done by calculating the raw counts of each term in the text, after some preprocessing to remove common words, and then dividing each count by the count of the most frequent word in the document. This means that the most common word in a document will have a frequency value of 1, and all others will be less than or equal to 1. By calculating term frequency this way, we enable comparisons between documents of different lengths. For example, tweets and news articles vary greatly in their length, and would have vastly different scales of term frequencies. But normalizing it this way allows us to make comparisons between these two types of text data.

Additionally, the Relevance and Similarity Models required slight changes to work with this new type of dynamic data. These Models were initially designed to work with data containing a static set of attributes. When you have all of your data that you will visualize ahead of time, this makes sense. However, with this type of visualization where you have new, unknown data coming in, the set of attributes will change and potentially grow drastically over time. These Models were modified to account for this, and in fact are completely compatible with the first pipeline presented, requiring only one small change to the CSV Data Controller.

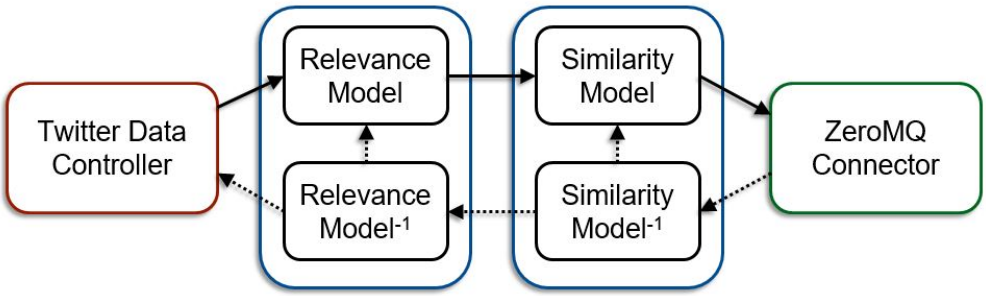

Figure 4.13: A pipeline that pushes data asynchronously from the Data Controller.

4.4 Streaming Data Analysis

Finally, we present an example that demonstrates asynchronous push behavior. This feature allows us to research interactive visualizations of truly streaming data, where the visualization can update itself with new data without an interaction occurring. In this example, we stream live tweets to the visualization, based on filters set by the user. The Models required no further changes, and only a new Data Controller and Connector had to be created. This pipeline can be seen in Figure 4.13.

4.4.1 Data Controller

The Data Controller for this pipeline does not load any data at startup. Rather, it initiates a connection to Twitter using the Tweepy package [35], and waits for an interaction to indicate what tweets it should filter for. When an interaction arrives to set the words to track, the Data Controller sends the request to Twitter to begin sending any tweets matching the filter. When each tweet arrives, it is processed to eliminate common words, punctuation,

and URL's. Term frequency values are then calculated for each word present in the tweet, and this data is stored in a list within the Data Controller. After a certain number of tweets have arrived or a certain amount of time has passed, which are both tunable parameters, the collected tweet data is asynchronously pushed down the pipeline. This push mechanism described in the previous chapter is the only way new data gets sent out of this Data Controller, as only live tweets are considered.

4.4.2 Connector

A new Connector had to be created for this pipeline to enable the push functionality. The initial RPC based Connector was only able to respond to requests from the visualization controller, so no updates to the visualization could occur without an interaction. This new Connector is built directly on ZeroMQ's PAIR-PAIR socket messaging pattern [37]. A socket is created on both the Connector and the visualization controller, with the former acting as the server and the latter connecting as a client. Messages are then sent using JSON strings to communicate the type of request (*Update*, *Get*, or *Reset*), and the corresponding arguments to the request. With this, the Connector can send an *Update* message to the client without requiring a request to respond to, enabling asynchronous pushes of data.

Figure 4.14: A visualization for streaming tweets. A new text box on the top lets users set a filter for tweets to pull in.

4.4.3 Visualization

The visualization client for this pipeline can be seen in Figure 4.14. It is nearly identical to the first visualization presented in this chapter, as it uses the same Models to transform the data and interpret interactions. The main addition is the second text box on the top panel that allows for input of a tweet filter. Submitting a filter triggers the interaction interpreted by the Data Controller to initiate a collection of tweets matching the filter. These tweets will then be asynchronously pushed through the forward pipeline as they arrive, and the user can use the same set of interactions previously described to start organizing and making sense of the tweet space. Additionally, the default label for each data point is the contents of the tweet since they are limited in length.

This example demonstrates how the architectural problems of working with streaming data can be solved with the pipeline framework. Any module can listen for new incoming data and update the visualization accordingly. However, asynchronously updating an interactive visualization can cause serious usability issues. If data is constantly moving around the screen, how can you interact with it? Producing prototypes of different streaming visualizations will be necessary to solve the usability problems that go along with it. By solving the architectural problem, researching solutions to the usability problem will become much simpler.

4.5 Raw High Dimensional Data Analysis

The previous examples were designed for visualizing and analyzing text data. They used a term frequency based approach to transform the text into numerical high-dimensional data. Over a large corpus of data, this leads to a number of attributes or features in the thousands or higher. Because of this, we saw no reason to include the manual weight manipulation techniques present in one of the existing tools we based the features off of [32]. However, the only text-specific piece of the previous pipelines is the Data Controller, and the remainder may be used for any type of high-dimensional data. As such, we created an alteration of the pipeline and visualization based on the aforementioned tool to allow for the visualization and manipulation of the attribute weights for high-dimensional data sets with a relative small number of dimensions.

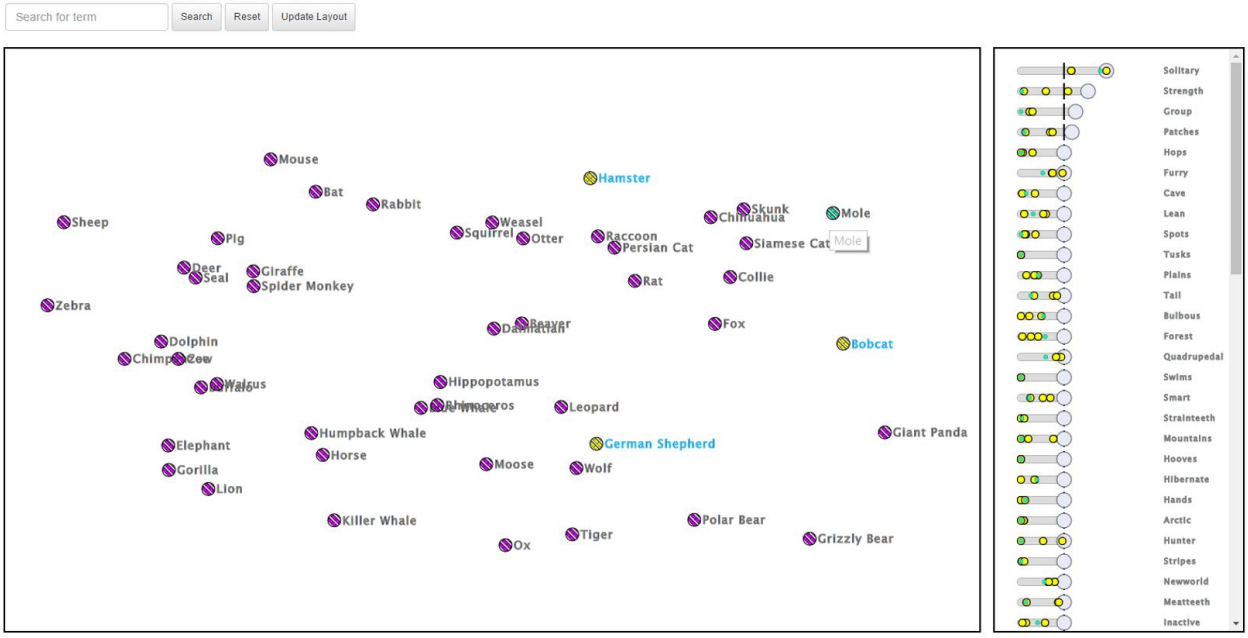

Figure 4.15: A visualization that includes the current weight for each attribute in the data. These weights can be manipulated directly to create a new projection of the data.

The slider based approach presented in [32] was simple to implement. An extended version of the Similarity Model was created to add in the functionality for interpreting interactions where users manually change attribute weights. We simply took out the Relevance Model, assuming we are working with a small, fixed set of data as in [32], and we were able to obtain the same core functionality. The only additional work was creating a web-based interface for these slider displays and interactions. Again, we built off our previous designs, swapping out the data information panel with the panel of attribute weights. This interface can be seen in Figure 4.15.

Implementing functionality from existing tools may be no great feat, but the simplicity in doing so demonstrates how quickly new features could be prototyped and tested. One

new approach that could be very easily examined is using the Relevance Model with pure numerical data. For example, you could have thousands of survey results you want to explore, and adding the Relevance Model to the pipeline would enable sorting through large data sets using a combination of OLI with the data points and direct parametric interaction with the attribute weights themselves.

Chapter 5

Discussion and Future Work

Designing interactive visualizations in a highly modular fashion provides many benefits toward the research of visual analytics tools. Here we explore some of these benefits we have already encountered, and reflect back on the research questions proposed earlier. Additionally, we discuss possible improvements to the framework.

5.1 A Modular Framework

The modularity of the framework provides many benefits towards creating visual analytics tools. As seen in the previous examples, we were able to create many different prototypes by swapping out individual pieces of the overall pipeline. The Relevance Model and Similarity Model provided the foundation for the data transformation. These Models, or some extension of them, were used in each of the examples aside from the raw high dimensional data analysis

pipeline, which solely used the Similarity Model. Designing the Models in this modular fashion forced us to make little assumptions about what type of data could be processed. While originally these Models were designed to work with a fixed set of attributes, the creation of more complex pipelines led us to remove this restriction and work with a variable and ever expanding set of attributes.

We have demonstrated how a new Data Controller can enable the pipeline to work with several different types and sources of text data. But this is only one possible type of data to be analyzed. The Similarity and Relevance Models described in the previous section are designed to work with numeric high dimensional data. In the examples we presented for visualizing text data, the raw text had to be transformed into some form of numerical data, namely term frequency data, to work with these Models. While this may seem like a limitation, it actually provides more generic functionality that can enable data exploration with many different types of data. Any type of data could be transformed to numerical data and work with this pipeline. For example, pixel data for images could be transformed into numerical vectors in some way to enabling a similar type of exploration of images.

Moving beyond analyzing specific types of data, new pipelines could be created to analyze multiple types or sources of data simultaneously. In the simplest case, text data from varying sources could be analyzed together just through the creation of a new Data Controller. It could connect to multiple search engines or text databases, and the remaining pipeline would not have to be changed. A more complex scenario is analyzing different types of data together, such as text and image data. One possibility is to have separate Models to work

with each type of data, and visualize them separately.

However, to truly compare visualize and interact with different data types together, there must be a method to numerically compare them. With a statistical method to compare the similarity between text and images, the framework could easily support this type of analysis. For example, a new Model could be created and inserted before the Relevance Model that takes multiple types of data and computes some singular form of numerical data with meaning across all types. With this common numerical data, the Relevance and Similarity Models would function just the same and provide a means of analysis. The burden is solely discovering the mathematical techniques for comparing these data types, not creating a system for doing so.

Modularizing the features and algorithms from other works also enables us to further experiment with and improve upon them. For example, working with the inverse MDS code previously developed in [32] was much simpler once we created our own library for it. Optimal distance measurements between points become complicated in high dimensional space. While Euclidean distances may still be valid, they may not make the most sense, because as the dimensionality grows, points tend to be near the edge of the hypercube and distances between all points converge [1]. We were easily able to include other distance functions, such as cosine distance, which measures the angular distance between two points and tends to work better for sparse text documents [20]. On top of this, we discovered a small bug in the inverse MDS algorithm that was not comparing correctly scaled distances in low and high dimensional space. We were able to modify it to make it completely scale independent, as

it was designed to be.

While most of our examples focused on creating a new Data Controller to visualize some new type of data, these same pipelines could easily be used to test different visual encodings and interaction techniques on the front end. The radar example demonstrates how relevance can be mapped in multiple ways. Clients can interpret the low dimensional projection and relevance however they choose. Making use of the third dimension or other visual features such as color or brightness could enable more information to be displayed to the user without overwhelming them.

Additionally, the same Data Controllers and front end clients could be used to test different Models. While we chose to use simple term frequency based approaches for analyzing text data, new Models could easily be created to perform more complex clustering and categorization operations. One such method we are currently investigating for text data is using Latent Dirichlet Allocation (LDA) to create topics over the document space [6]. With LDA, each document gets assigned a probability vector that indicates how likely it is part of each topic. This allows for an easy combination of different types, sources, and scales of text data into a single format that can be easily compared against each other. Additionally, it provides an initial dimensionality reduction to reduce the number of attributes. This enables algorithms operating on the high dimensional data to execute much faster, allowing for real time interactions on larger data sets.

5.2 Research Questions

We previously discussed a set of research questions we hope this framework will help in answering. Here we discuss how some of the example visualizations presented can be used to answer these questions.

The multi-source text analysis is the first step toward studying the combination of different types of data in a single visualization. The simplest case of multiple types of data is vastly different types of text data, such as short and sometimes incoherent tweets, to length and grammatically correct news articles. However, this could be further expanded to more disparate types of data. Structured text data, such as spreadsheets, is one extension of basic text data that is non-trivial to analyze. Adding a completely different data type, such as images, adds a whole new aspect. Studying streaming data within a visual analytics tool should be greatly simplified by this framework. We have provided a solution to the architectural challenge of enabling streaming data, demonstrating this with the Twitter example. This will enable the future work to focus solely on how to spatialize and interact with such data.

As previously discussed, new Models could enable us to interact with larger data sets in real time, such as one implementing LDA [6]. The multi-source example also demonstrates how this would be possible. Rather than working on preprocessed data, it calculates term frequency data on the fly to work with non-static data sets. This Term Frequency Model could easily be replaced with an LDA Model that converts these raw text documents to LDA

topics instead of term frequency vectors.

Different visual encodings can just as easily be studied, as the pipeline has no sense of what encodings are used for the data it provides. The radar and 3D visualizations demonstrate how the same pipeline can be used to visualize the same data in different ways. This allows researchers to focus solely on the interface and encoding the information without having to be concerned with the data processing. The same can hold true for interactions. While some general interactions are defined within the Models, how these interactions get triggered are left to the visualization. For example, users currently have to modify the relevance of a document by typing in a new value. This could easily be changed to allow users to resize nodes to indicate increased or decreased relevance, and this would not require any changes in the Models.

These generalized types of interactions also simplify implementing non-traditional interfaces. Large, high resolution displays allow users more space to manage their thought process [2, 3]. Likewise, 3D and immersive interfaces, can bring many new facets to visual analytics [4, 7, 29, 30]. For example, node selection could be done through means of hand tracking or through a peripheral device in such environments, but these methods just need to be mapped to the same interaction understood by the Model. This separation between capturing interactions and interpreting interactions enables better collaboration between multiple bodies of research. An expert in immersive technologies could easily create their own visual analytics tools by creating an interface for a pipeline created by a data analysis expert. This ability to collaborate will further hasten research into visual analytics tools.

While the examples previously presented show independent methods for visualizing data, one might consider combining multiple different views or perspectives for examining a single data set. One possibility is by creating two separate views that use the same outputs of the pipeline. For example, a keyboard and mouse based 3D visualization could be used simultaneously with a client supporting virtual reality and peripheral devices for interacting in 3D. Since the inputs and outputs of these two views is the same, they could safely interact with the same pipeline without requiring any new functionality. Alternatively, one might consider multiple views within a single visualization, such as multiple plots that use different sets of weights for the dimensionality reduction, to test multiple hypotheses concurrently. While this would require the creation of new Models to handle multiple dimensionality reductions, there is nothing in the framework preventing such behavior.

5.3 Pipeline Improvements

One feature common in most analytics tools not present in our pipeline is undo functionality. It is a capability present in almost any computer application a typical analytics tool user has interacted with, and provides many benefits when analyzing data. If you do not like an interaction or to something accidentally, undoing can immediately revert that change. The downside of the highly modular framework is the complexity in enabling such a feature. It would require the pipeline itself keeping some kind of state over each of the modules, and the modules themselves having some logic regarding undoing its most recent parameter changes.

To make matters worse, some of the algorithms used in our pipelines, such as the forward and inverse MDS algorithms, are non-deterministic, so maintaining state of previous sets of attribute weights would not enable proper undo functionality. Each projection of the data would need to be stored to return them to their exact previous projections.

This modularity could also be further built upon. While we believe the current framework can be extremely useful for studying most aspects of analytics tools, it does not scale as well as it possibly could. The entire pipeline exists within a single Python instance, limiting the resources that may be available to individual pieces within the pipeline. These different modules can create their background threads to improve throughput, but allowing each module to exist on different machines within their own processes could improve things even further. Enabling this would be fairly straightforward, as the Models would be created in the exact same fashion where they accept some JSON data blob and update it. The main change would be within the pipeline itself coordinating communication between each piece. Handling different modules on different machines would likely be a headache for researchers, so this modification would only be useful for studying certain algorithms that are optimized for specific types of machines.

Chapter 6

Conclusion

We have developed a bidirectional pipeline framework for creating visual analytics tools that use semantic interaction and V2PI to aid the sensemaking process. Three key pieces make up this framework. A Data Controller defines what type of data is being visualized and how it is accessed. A series of Models transform the data into a form suitable to be visualized, as well as interpret interactions from the visualization. Finally, a Connector controls how the visualization communicates with the pipeline. Each of these pieces can easily be replaced to quickly prototype and experiment with different types of data, mathematical models, interaction techniques, and visual encodings.

We demonstrate this ability by developing several prototypes that exemplify how to research each of these aspects. We hope that this new pipeline will replace the traditional visualization pipeline to emphasize its bidirectional nature and the role of inverse algorithms to interpret

interactions. Additionally, the modularity well help multiple developers work together to create visual analytics tools.

By enabling rapid prototyping of such tools, researchers will be able to quickly conduct user studies on many of these alternative methods of semantic interaction. We intend to continue expanding on these prototypes and conduct our own user studies. These studies will reveal how the user perceives these different visual encodings and interactions, which methods best support the user's sensemaking process, and how to develop better visual analytics tools in the future.

Bibliography

[1] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proceedings of the 8th International Conference on Database Theory*, ICDT '01, pages 420–434, London, UK, UK, 2001. Springer-Verlag.

[2] C. Andrews, A. Endert, and C. North. Space to think: Large high-resolution displays for sensemaking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 55–64, New York, NY, USA, 2010. ACM.

[3] C. Andrews and C. North. Analyst's workspace: An embodied sensemaking environment for large, high-resolution displays. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 123–131, Oct 2012.

[4] R. Ball, C. North, and D. A. Bowman. Move to improve: Promoting physical navigation to increase user performance with large displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 191–200, New York, NY, USA, 2007. ACM.

[5] J. Behr, P. Eschler, Y. Jung, and M. Zöllner. X3dom: A dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, Web3D '09, pages 127–135, New York, NY, USA, 2009. ACM.

[6] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[7] D. A. Bowman, E. Kruijff, J. J. LaViola, and I. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[8] L. Bradel, C. North, L. House, and S. Leman. Multi-model semantic interaction for text analytics. In *Visual Analytics Science and Technology (VAST), 2014 IEEE Conference on*, pages 163–172, Oct 2014.

[9] L. Bradel, N. Wycoff, L. House, and C. North. Big text visual analytics in sensemaking. In *IEEE International Symposium on Big Data Visual Analytics*, page 8 pages, 09/2015 2015.

[10] S. Cheng and K. Mueller. The data context map: Fusing data and attributes into a unified display. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):121–130, Jan 2016.

[11] G. Chin, M. Singhal, G. Nakamura, V. Gurumoorthi, and N. Freeman-Cadoret. Visual analysis of dynamic data streams. *Information Visualization*, 8(3):212–229, June 2009.

[12] D3. `https://d3js.org/`, 2016. Accessed: 2015-11-05.

[13] A. Endert, L. Bradel, and C. North. Beyond control panels: Direct manipulation for visual analytics. *IEEE Comput. Graph. Appl.*, 33(4):6–13, July 2013.

[14] A. Endert, P. Fiaux, and C. North. Semantic interaction for sensemaking: Inferring analytical reasoning for model steering. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2879–2888, Dec 2012.

[15] A. Endert, P. Fiaux, and C. North. Semantic interaction for sensemaking: Inferring analytical reasoning for model steering. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2879–2888, Dec 2012.

[16] A. Endert, P. Fiaux, and C. North. Semantic interaction for visual text analytics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 473–482, New York, NY, USA, 2012. ACM.

[17] A. Endert, C. Han, D. Maiti, L. House, S. Leman, and C. North. Observation-level interaction with statistical models for visual analytics. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pages 121–130, Oct 2011.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[19] C. Gormley and Z. Tong. *Elasticsearch: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2015.

[20] M. Goto, T. Ishida, and S. Hirasawa. Statistical evaluation of measure and distance on document classification problems in text mining. In *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, pages 674–673, Oct 2007.

[21] J. Hendler. Web 3.0 emerging. *Computer*, 42(1):111–113, Jan 2009.

[22] L. House, S. Leman, and C. Han. Bayesian visual analytics: Bava. *Statistical Analysis and Data Mining*, 8(1):1–13, 2015.

[23] X. Hu, L. Bradel, D. Maiti, L. House, C. North, and S. Leman. Semantics of directly manipulating spatializations. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2052–2059, Dec 2013.

[24] M. Krstaji and D. A. Keim. Visualization of streaming data: Observing change and context in information visualization techniques. In *Big Data, 2013 IEEE International Conference on*, pages 41–47, Oct 2013.

[25] S. C. Leman, L. House, D. Maiti, A. Endert, and C. North. Visual to parametric interaction (v2pi). *PLoS ONE*, 8(3):1–12, 03 2013.

[26] Node.js. `https://nodejs.org`, 2016. Accessed: 2015-10-24.

[27] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. *Proceedings of International Conference on Intelligence Analysis*, pages 2–4, 2005.

[28] N. F. Polys, B. Knapp, M. Bock, C. Lidwin, D. Webster, N. Waggoner, and I. Bukvic. Fusality: An open framework for cross-platform mirror world installations. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D '15, pages 171–179, New York, NY, USA, 2015. ACM.

[29] N. F. Polys, A. Mohammed, J. Iyer, P. J. Radics, F. Abidi, L. Arsenault, and S. Rajamohan. Immersive analytics: Crossing the gulf with high-performance visualization. In *2016 IEEE VR Workshop on Immersive Analytics (IA)*. IEEE, March 2016. (to appear).

[30] G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. van Dantzich. Data mountain: Using spatial memory for document management. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, UIST '98, pages 153–162, New York, NY, USA, 1998. ACM.

[31] T. Ruotsalo, J. Peltonen, M. Eugster, D. Glowacka, K. Konyushkova, K. Athukorala, I. Kosunen, A. Reijonen, P. Myllymäki, G. Jacucci, and S. Kaski. Directing exploratory search with interactive intent modeling. In *Proceedings of the 22nd ACM international conference on Conference on information and knowledge management*, CIKM '13, pages 1759–1764, New York, NY, USA, 2013. ACM.

[32] J. Z. Self, L. House, S. Leman, and C. North. Andromeda: Observation-level and parametric interaction for exploratory data analysis. Technical report, Department of Computer Science, Virginia Tech, Blacksburg, Virginia, 2015.

[33] J. Z. Self, X. Hu, L. House, S. Leman, and C. North. Designing for interactive dimension reduction visual analytics tools to explore high-dimensional data. Technical report, Department of Computer Science, Virginia Tech, Blacksburg, Virginia, 2015.

[34] A. Taivalsaari and T. Mikkonen. The web as an application platform: The saga continues. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 170–174, Aug 2011.

[35] Tweepy. `http://www.tweepy.org/`, 2016. Accessed: 2016-06-15.

[36] Websockets. `https://www.w3.org/TR/websockets/`, 2016. Accessed: 2015-11-05.

[37] Zeromq. `http://zeromq.org`, 2016. Accessed: 2016-02-09.

[38] zerorpc. `http://zerorpc.io`, 2016. Accessed: 2016-02-09.

Appendix A

Creating a Pipeline

Creating a new pipeline is a simple task with the framework, and this chapter steps through all the necessary pieces. Three sections make up a fully functional pipeline: a Data Controller, a list of Models, and a Connector. Each of these has a base class contained in the *nebula.pipeline* module. These classes contain all necessary functions, each of which can be overridden in a subclass to define its behavior. The *Pipeline* class mediates communication between each of these pieces. Here we explain exactly what the functions are in each module that makes up a pipeline and when they are called, further explore the communication, and finally demonstrate how to put the pieces together.

A.1 Data Controller

The purpose of the Data Controller is to be the main translator of the data to be visualized to a form the rest of the pipeline can understand. This data may be locally stored or in some remote database. Different Data Controllers can easily be created for different types or locations of data. A Data Controller has four main functions which can be overridden: *setup*, *get*, *run*, and *reset*.

The *setup* function is executed on the initial startup of the pipeline, and whenever the pipeline is reset. It allows for data to be passed down to the *setup* function of the Models within the pipeline. The argument to this function is a data blob in the form of a dictionary object which can be modified in place to send data to the Models. This can allow the pipeline to start off with some initial set of data or random sampling, rather than waiting for the first user interaction to pull in data. *setup* should not returning anything.

The *get* function is unique to the Data Controller. Its purpose is to send raw data and metadata directly to the visualization, rather than passing through the whole pipeline. This function will be called by the pipeline, so the Data Controller merely needs to respond to requests made through its dictionary argument. The return value should be a dictionary containing the results of the query.

run is the workhorse of the Data Controller. Upon each iteration of the pipeline initiated by a user interaction, the *run* function gets called after all the inverse functions of the Models are executed. The argument to this function is a dictionary object that has passed through all

of the Models' inverse functions, allowing the Models to pass data up to the Data Controller to retrieve specific types of data. The return value of this function is important, as whatever is returned becomes the data blob which is passed down the forward pipeline. As such, it must be a dictionary object.

Finally, the *reset* function behaves exactly as expected. Whenever the pipeline is reset, the reset function of the Data Controller and each Model is executed, allowing any parameters to be reset to their initial values. It takes no arguments and doesn't return anything.

A.2 Models

Models contain very similar functionality to Data Controllers. There are four key functions that can be overridden to provide their functionality: *setup*, *forward*, *inverse*, and *reset*. Additionally, the *forward_input_reqs*, *forward_output*, *inverse_input_reqs*, and *inverse_output* functions are used to specify the inputs and outputs of each Model to ensure a valid and properly constructed pipeline, and are further explored in Section A.4 below. The *setup* is called on the initial pipeline startup, as well as after a reset. The argument passed to the *setup* function is a dictionary that has been passed through all Models closer to the Data Controller, as well as the Data Controller itself. Any changes made to this argument will be available to the remaining Models' *setup* functions as well. The *reset* function behaves exactly the same as the Data Controller.

The *forward* function should contain the main logic for transforming the data to a form the

visualization can use. Its argument is the dictionary object that contains the data from the Data Controller's *run* function return value, as well as all modifications made by previous Models' *forward* functions. Any changes made to this argument will be available to the remaining Models' *forward* functions. There should be no return value.

inverse interprets user interaction to modify parameters within the Model. Like the others, a single dictionary argument is provided with details of the interaction from the visualization as well as any modifications made by Models' *inverse* functions closer to the Connector. Returning a value from the *inverse* function signals the pipeline to short circuit itself. The return value then becomes the data blob immediately passed into the current Model's *forward* function.

A.3 Connector

The Connector defines a method for a visualization to communicate with the pipeline. This method may be an RPC mechanism, raw sockets, or any other form of communication with an external process. Connectors have three methods that define their behavior: *set_callbacks*, *start*, and *push_update*. When a Connector is added to a pipeline, the pipeline will call the *set_callbacks* function with callbacks for an *update*, *get*, and *reset* function. Using whatever communication method is chosen, these callbacks must be called when the appropriate message is received. The Connector should send the return value from the *update* and *get* callbacks to the pipeline. The *reset* function does not return any value.

start simply lets the Connector know that is should start listening for incoming messages from the visualization. For example, if using a socket based approach, the socket would start listening for incoming messages in the *start* function. The *push_update* function allows updates to be asynchronously pushed to the visualization. In order to support this, a Connector must have a way of sending a message to the visualization directly. RPC approaches may not support this, as they can only respond when called by the visualization. This function must only be defined if the Connector wishes to support push functionality.

A.4 Communication

Communication within the pipeline is key to understand before developing new modules. This communication is controlled by the pipeline in the form of data blobs that are passed to the various functions within each module. These data blobs exist as Python dictionaries within the pipeline framework and are typically transformed to JSON objects outside of the framework. Modules communicate with each other by modifying or adding elements to these data blobs which are passed in as an argument to the various core functions.

There are several points where a data blob is created. The first is for the series of setup calls, and is illustrated in Figure A.1. An empty blob is created and passed to the Data Controller's *setup* function, followed by each Models' *setup* function. Second, a data blob can be created from the visualization to initiate an iteration of the pipeline. This data blob only can make it as far as the Data Controller, which would then create a new data blog for

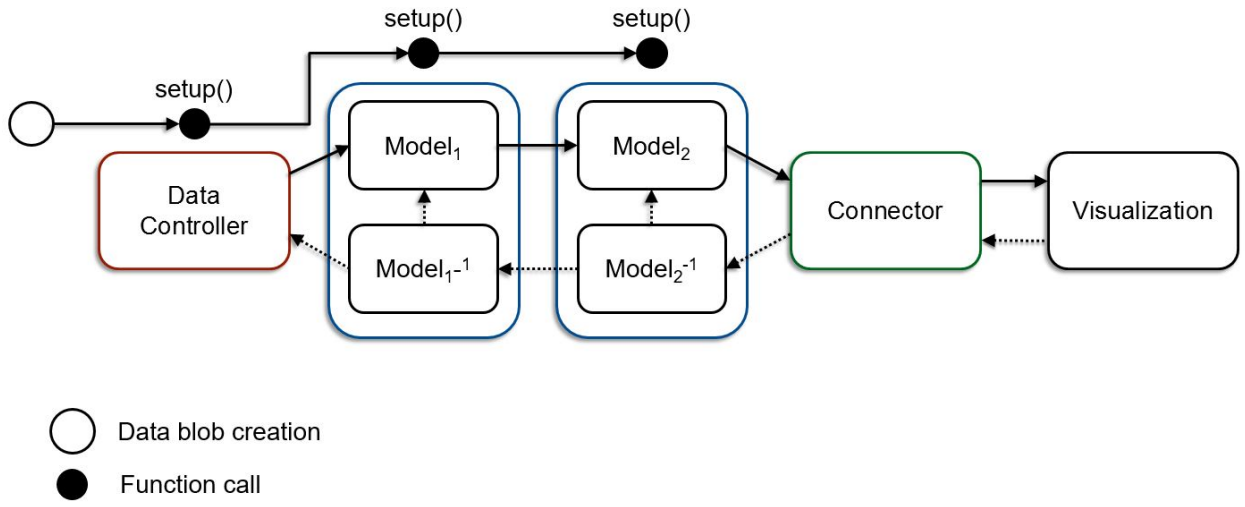

Figure A.1: A data blob is created by the pipeline and passed through the setup up function of the Data Controller and each Model.

the forward pipeline. This case is shown in Figure A.2. Alternatively, a Model can choose to short circuit itself by creating a new data blob in its inverse function. This then becomes the data blob for the forward pipeline beginning with that Model, and is demonstrated in Figure A.3. Finally, a data blob can be created in a background thread of a Data Controller or Model through the push functionality, shown in Figure A.4 and Figure A.5, respectively. These data blobs are deleted at the end of their path.

When designing new Data Controllers and Models, it is key to understand what the input requirements and outputs of each module are. For example, forward functions of Models are designed to enrich the data in some way, but to do this they must have some data to enrich and know where to find it. As previously stated, this is done through acting on certain elements in the data blob based on a common key. To help aid the development process,

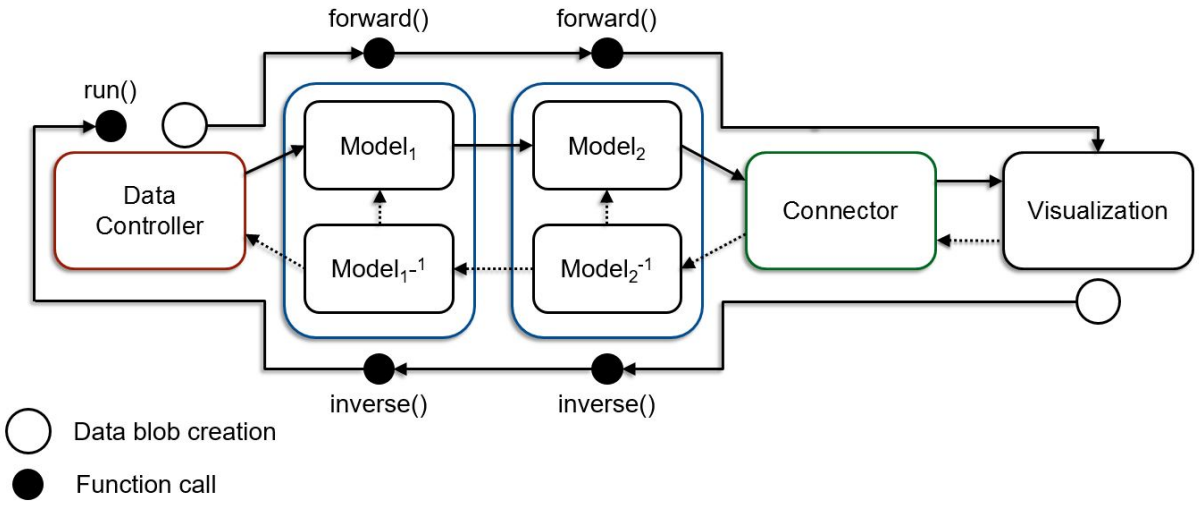

Figure A.2: A data blob is created by the visualization representing an interaction. The Data Controller creates a new blob for the forward pipeline.

Data Controllers and Models are required to list out their required elements in the data blob for their forward and inverse functions, as well as the elements that they may add. The inputs for each Model are then compared against the output of all previous Models. These keys do not have to exist in all data blobs, but should exist some of the time for a Model to provide any beneficial behavior.

These inputs and outputs are specified through the *forward_input_reqs*, *forward_output*, *inverse_input_reqs*, and *inverse_output* functions within a Model, and the *input_reqs* and *output* functions of a Data Controller. These functions simply return a list of strings representing keys within the data blob. At the startup of a pipeline, all of these inputs and outputs are compared against each other to ensure that the pipeline has been properly constructed. If any Model is missing an input, an error will be reported and the pipeline will not start.

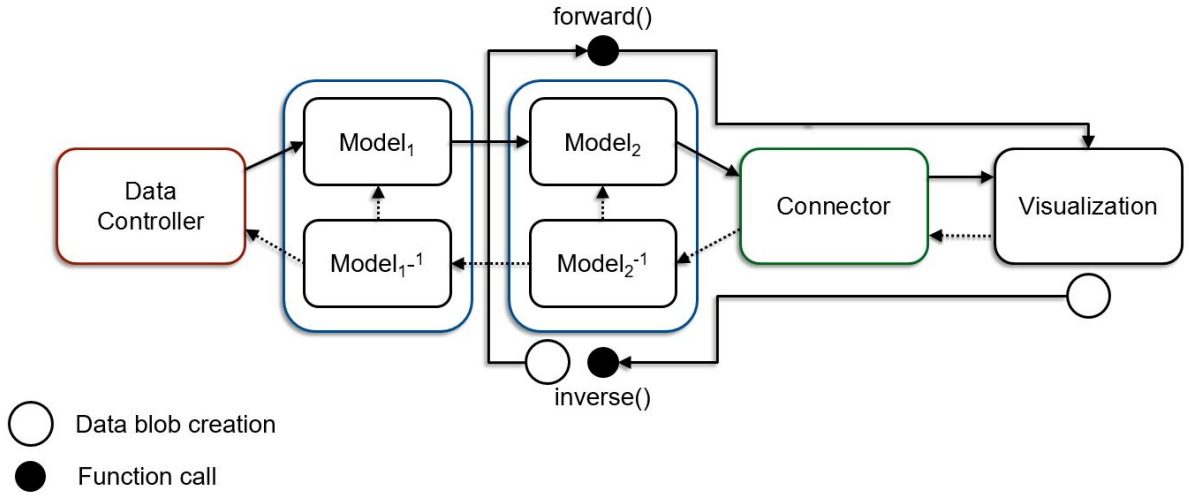

Figure A.3: Inverse functions trigger a short circuit by returning a new data blob.

The format of each of these elements do not have any strict definition, however. If two Models or a Model and Data Controller operate on the same element within a data blob, it is assumed that they have a common understanding of how that element is formatted. Because of this, it is important to provide clear definitions of what each key within the data blob represents. The element mapped to a specific key should never have different formats. Should a new format be required, a new element at a new key should be specified. The elements used in the existing Models and Data Controllers use a common set of key definitions within the *pipeline* module. Each of these keys has a brief description of what the element is meant to represent. New modules are not limited to using these keys, but if new ones are created there should be a strict definition of what element exists at that key.

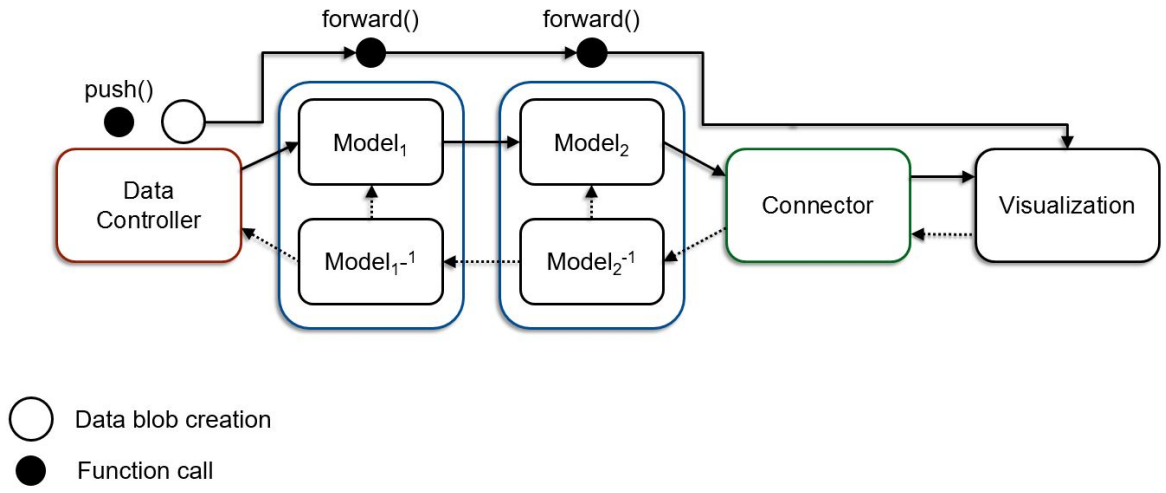

Figure A.4: The Data Controller asynchronously pushes a new data blob down the pipeline.

A.5 Putting it Together

Each of these pieces come together in a single Pipeline object. The Pipeline class is not meant to be overridden, and provides all the logic to mediate communication between each different piece. Models can be added to the Pipeline using the *insert_model* or *append_model* functions, and the Data Controller and Connector can be set using the *set_data_controller* and *set_connector* functions, respectively. Once all the pieces are in place, a simple call to the *start* function of the Pipeline object will execute the *setup* function of the Data Controller and all Models and start the Connector. The Pipeline then listens for any messages from the visualization and responds accordingly.

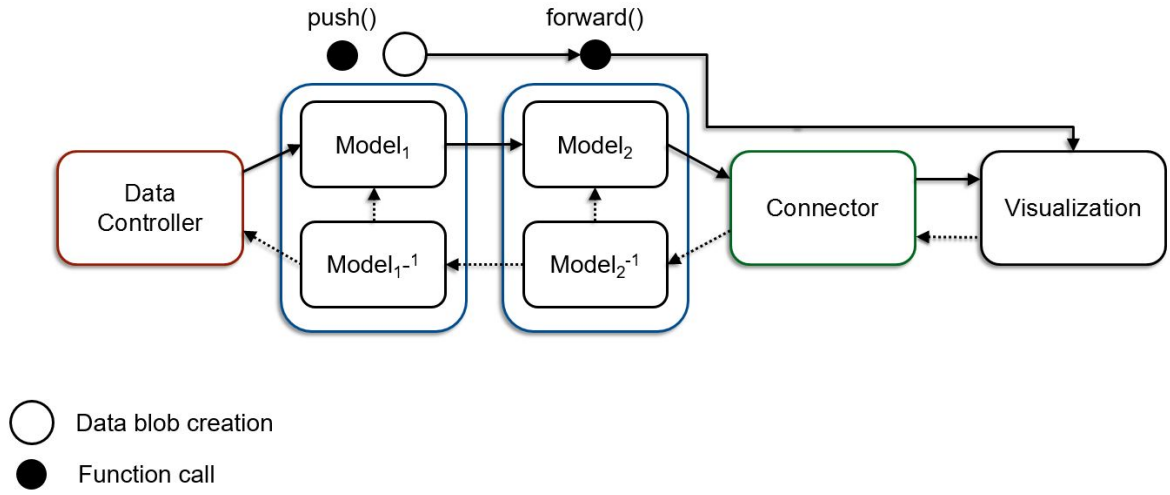

Figure A.5: A Model asynchronously pushes a new data blob down the pipeline.