Project in Visual Computing II

Project Report

Dominik Wolf

August 2024

Supervisor: Assistant Prof. Dr.techn. Manuela Waldner, MSc

1 Introduction

With the growing abundance of data, the task of exploratory data analysis is becoming increasingly complex. While identifying anticipated patterns remains straightforward, uncovering unexpected insights is becoming more challenging. However, both aspects are crucial to the exploration process. The main challenge lies in the frequently unstructured form of the data, rendering traditional data exploration methods impractical. Machine learning (ML) assists in analyzing vast unstructured data. However, experts in various research domains often seek highly specific properties and characteristics, which pre-existing ML models may not address. Additionally, the objective may be ambiguous or non-existent initially and can evolve over time.

The Joint Human-Machine Data Exploration (JDE) project introduces a novel approach to address this challenge. The goal of the JDE project is to facilitate efficient visual exploratory data analysis of large-scale unstructured data while allowing for dynamic user guidance and interaction throughout the process. The underlying architecture of the JDE framework defines several entities, visually depicted in Figure 1:

- $\mathcal{D}$: Represents the high-dimensional feature space of the input data in vector form.
- $\mathcal{K}$: The knowledge model, depicted as a graph that illustrates concepts and their relations to each other. This model depicts a representation of the user's knowledge and may include hierarchical structures, which can be represented within the graph.
- $\mathcal{F}$: The frame model, which depicts the mapping between the input data and the knowledge model, therefore representing the framing of the data through the user's knowledge.
- Ω: A controller updating the knowledge model and/or the frame model, interacting with either of these models.
- V: The visualization that displays the current state of any of these models.

The primary challenge for the JDE framework is effectively conveying how well the data aligns with the user's expectations and understanding of $\mathcal{D}$. To address this, a triple-view visualization approach has been proposed.

The first view, $V_{\mathcal{D}}$, provides a general overview of (potentially a subset of) the data $\mathcal{D}$. This can range from a simple grid of source data elements (e.g., images) to a more complex

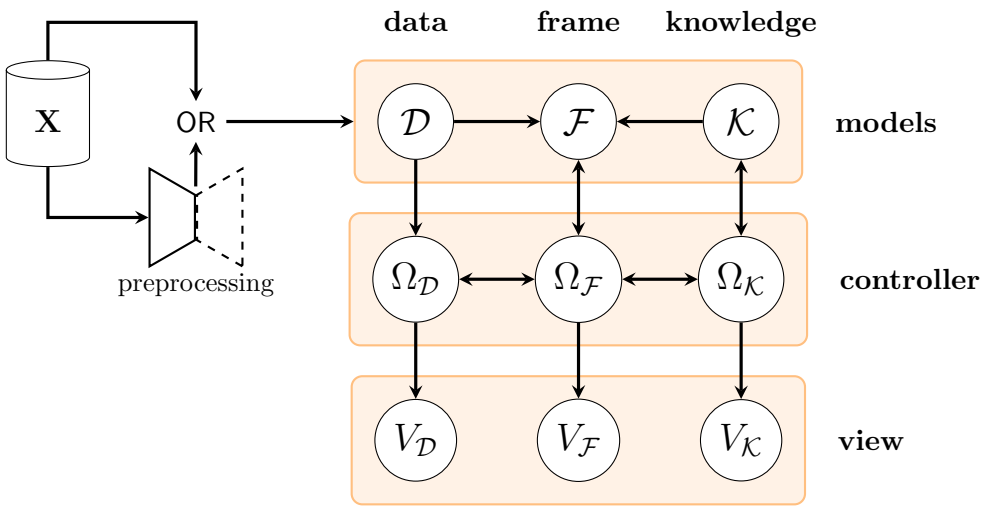


Figure 1: Architecture of the proposed JDE framework.

visualization, such as a 2D plot of the high-dimensional feature space using dimensionality reduction techniques like t-SNE or UMAP.

The second view, $V_{\mathcal{F}}$, offers insights into the alignment between the knowledge space $\mathcal{K}$ and the feature space $\mathcal{D}$. This alignment is visualized on a 2D plane, with the actual representation being interchangeable.

The third view, $V_{\mathcal{K}}$, visualizes the knowledge space $\mathcal{K}$ as an interactive graph. This graph comprises nodes and edges representing concepts and their associations. Users can adjust the graph's elements, refining the frame model space.

Users can easily toggle each view on or off within the visualization.

The objective of this project was to develop a software framework for an interactive prototype that incorporates the concepts of the JDE framework, tailored to the effective exploration and sense-making of large unstructured data.

2 Implementation

The implementation of the JDE framework involved several interface components designed to manage and process the data, track user interactions, and handle events that drive the system's dynamic updates. This section outlines the core components, interface areas, and the technology stack used to build the interactive prototype. The main interface is depicted in Figure 2.

2.1 Technology Stack

The JDE framework was implemented as a server-client application, comprising a front end and a back end. The back end utilizes Python as programming language and the FastAPI web framework for providing a REST API to the client. Additionally, the back end integrated several libraries for machine learning and data processing, including PyTorch for deep learning tasks and scikit-learn for a wide range of machine learning algorithms. On the front end, the framework was built using TypeScript, which transpiles to JavaScript for execution in the user's web browser, and React as component framework with D3.js for dynamic and interactive visualizations.

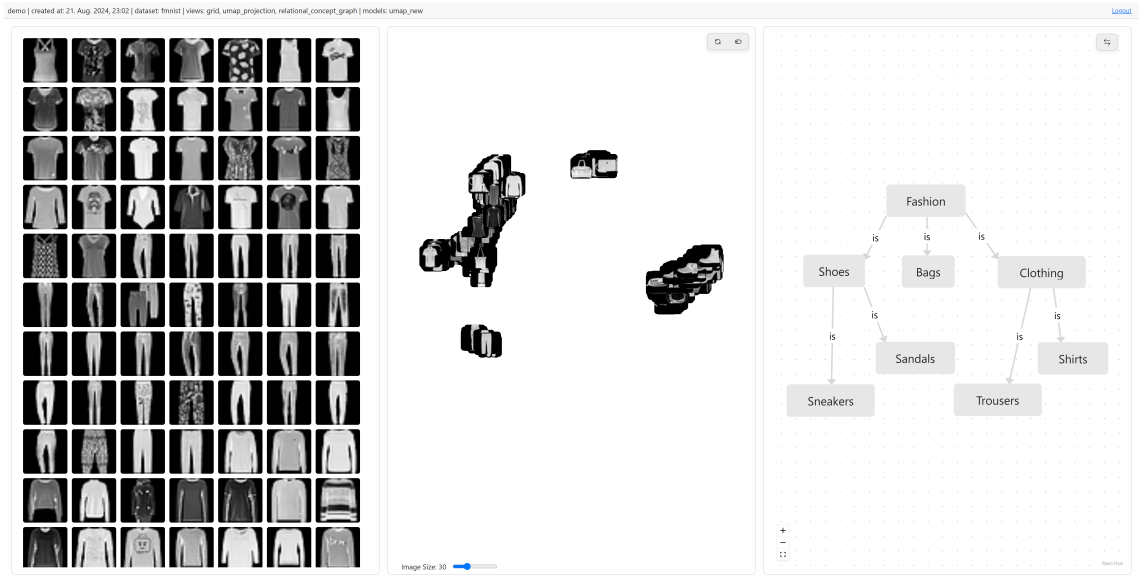


Figure 2: The main interface of the JDE framework.

2.2 Components

This section covers several core concepts and component designs which have been implemented in the JDE framework. Figure 3 provides an overview of how these components interact.

2.2.1 Datasets

In the JDE framework, datasets are designed as base classes to provide a flexible and extensible structure for data management. Each dataset class includes various abstract methods for accessing the items within the dataset, ensuring that the framework can handle diverse types of data consistently. This object-oriented approach allows the framework to support various forms of data, such as images, videos, and text, by treating them as distinct item types, which are also represented as implementations of an item base class.

For the initial implementation of the JDE framework, a concrete dataset implementation, the file dataset, was developed, which manages datasets stored locally on the file system. The dataset design allows the JDE framework to be easily extended to support other data sources. For example, additional dataset classes could be implemented to fetch data from external locations, such as via a web API or a cloud storage service.

2.2.2 Configurations

Configurations in the JDE framework are distinct, named combinations that define the relationship between datasets, views, and models. Each configuration specifies which models will process the data on the back end and which views will be used to display the data on the front end.

These configurations are stored in a JSON file, which is loaded when the back end is initialized. New views or models can be easily integrated by implementing the necessary components and referencing them within the JSON configuration file. A sample configuration block depicting the format of the JSON file is provided in Listing 1.

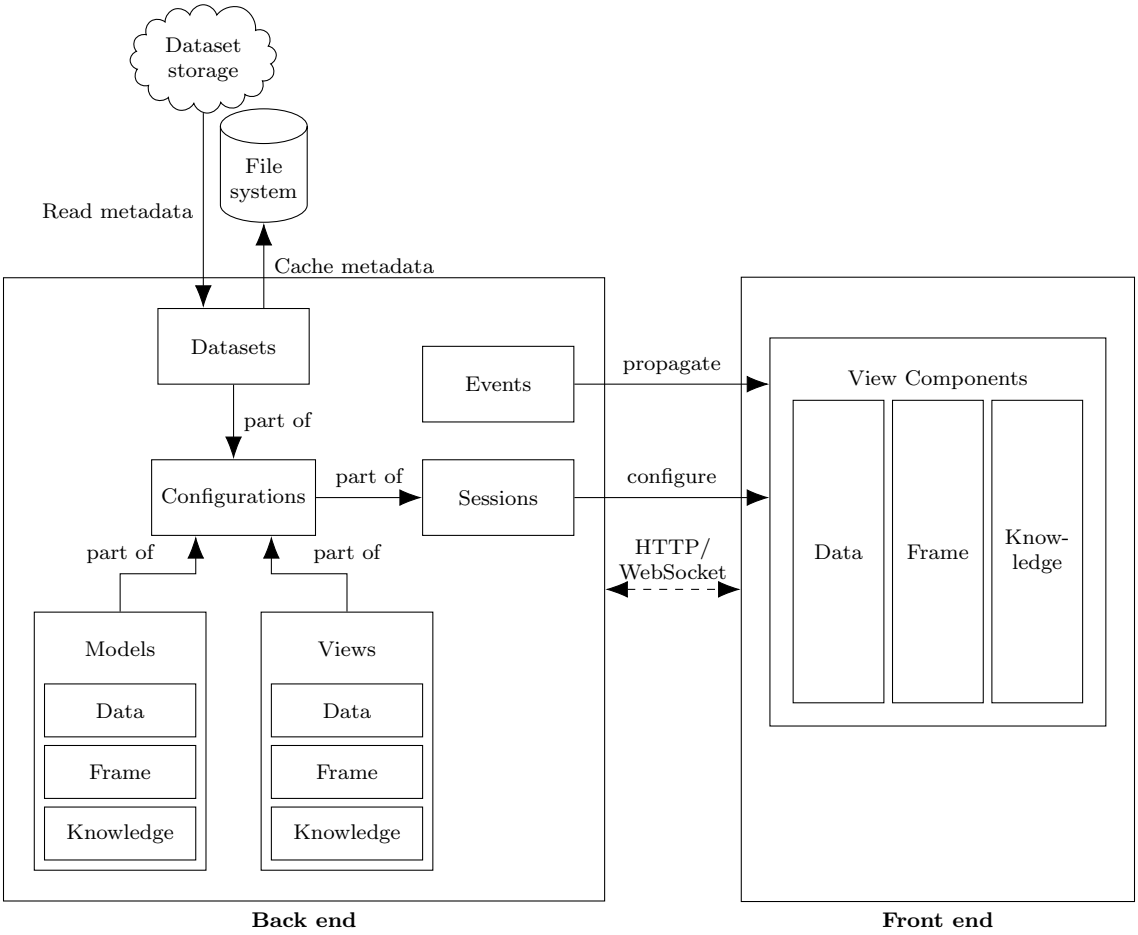


Figure 3: Overview of the key components and their interactions within the JDE framework.

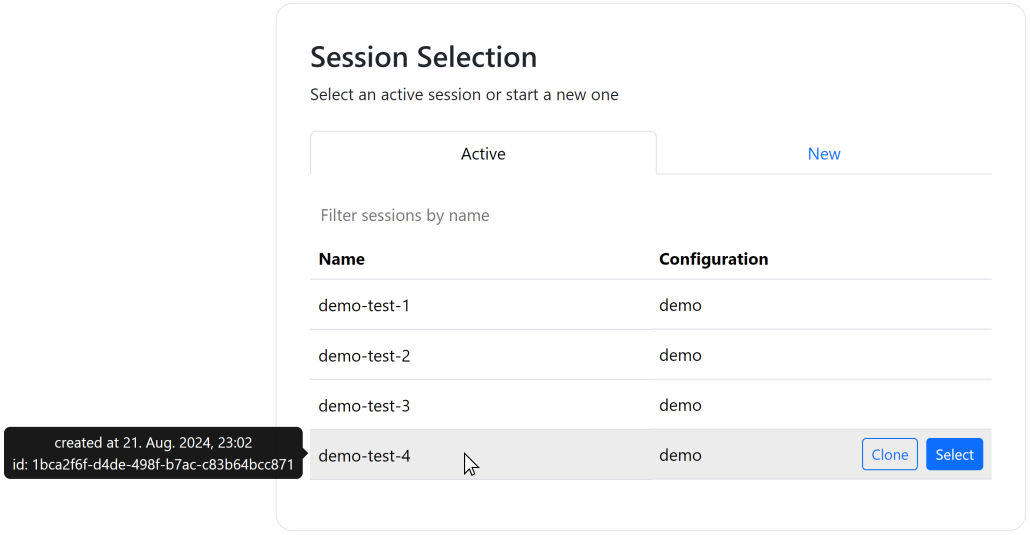


Figure 4: The session selection screen of the JDE framework. Hovering over an existing session entry reveals details such as the session's creation time and ID, along with buttons to either select or clone the session.

```
{
  "demo": {
    "dataset": "fmnist",
    "views": {
      "data": "grid",
      "frame": "umap_projection",
      "knowledge": "relational_concept_graph"
    },
    "models": {
      "data": "umap",
      "knowledge": null
    }
  },
  ...
}
```

Listing 1: Excerpt of the JSON file containing the configurations for the JDE framework.

2.2.3 Sessions

Sessions in the JDE framework are used to store and manage the state of the application. Each session is associated with a specific configuration, referenced by its name, and is also assigned a unique ID and a session title given by the user. This structure allows users to easily manage multiple sessions, each with its own configuration and state. Additionally, sessions can be cloned, enabling users to create duplicates with the same state for further exploration into other directions.

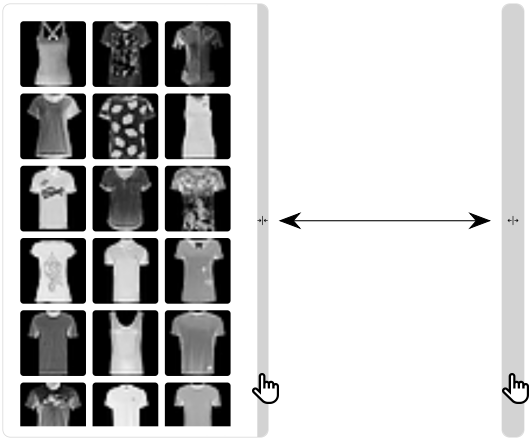
The session object includes a data property where models can store various pieces of information related to the session. This data must be serializable to ensure it can be saved and retrieved correctly. To maintain data integrity, the back end enforces concurrency control by locking the data property during access. This ensures that only one code section can modify the session's data at any given time.

When the data object is released after access, the framework verifies whether any changes have occurred by comparing the hash of the previous data state with the hash of the current state. If a change is detected (i.e., the hashes differ), the framework updates a timestamp that records the last modification.

A dedicated session saver thread is initiated when the back end starts. This thread regularly checks all sessions to determine if their data has been modified by comparing the timestamp of the last modification with the timestamp of the last save. If a difference is detected, indicating that the session's data has been updated, the session is serialized and saved to its own JSON file.

2.2.4 Events

The JDE framework includes an event system which utilizes WebSockets to push server-side events to the client, facilitating real-time communication for asynchronous tasks. The WebSocket connection is established when the page loads and is bound to the user's session. The WebSocket automatically reconnects if the connection is lost, ensuring a continuous communication between the server and the client. Events in this system are defined by a type and a payload. Front end components can subscribe to specific event types and have to implement the logic to handle the associated payload when an event is received.



Expanded view Collapsed view

Figure 5: Comparison between an expanded and a collapsed view. The cursor indicates the user's mouse click interaction that transitions from the expanded state to the collapsed state and vice versa.

2.3 Interface Areas/Views

The interface is organized into three distinct areas: the data view, the frame view, and the knowledge view, each arranged in a separate column. These areas are designed to display specific front end interface components based on the active configuration, as outlined in Section 2.2.2. Users can collapse one or two of the columns to maximize the space available for the remaining views, as illustrated in Figure 5. The following sections provide more detailed descriptions for each of these views.

2.3.1 Data View

The data view corresponds to $V_{\mathcal{D}}$ as described in Section 1. For the initial version of the framework, this view is implemented as a very basic grid of images. The grid displays all images from the dataset associated with the user's session, providing a visual overview of the data being explored.

2.3.2 Frame View

The frame view corresponds to $V_{\mathcal{F}}$ as described in Section 1. This view shows a 2D scatterplot which visualizes image features utilizing dimensionality reduction via UMAP. The scatterplot is built with D3.js and offers interactive functionality, allowing users to toggle between circle and thumbnail representations of the data points.

Users can pan, zoom in and out, and use lasso selection to highlight specific elements within the scatterplot.

2.3.3 Knowledge View

The knowledge view corresponds to $V_{\mathcal{K}}$ as outlined in Section 1. This view presents a hierarchical, interactive directed graph where users can dynamically add nodes and edges to model relationships. Nodes can be labeled with text, while edges can be assigned relation types, which users can create and customize.

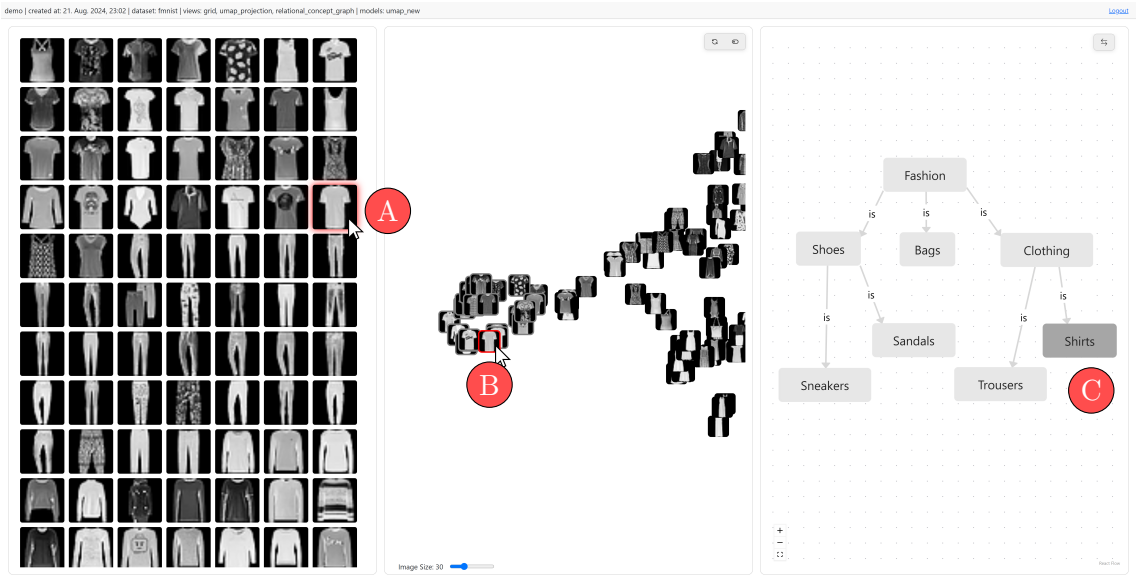


Figure 6: Illustration of the interaction between different views: Hovering over an item in the data view (A) or the frame view (B) highlights the corresponding item in the other view, highlights the nearest neighbors in the frame view, and, if applicable, highlights the associated label in the knowledge view (C).

The directed graph allows users to model semantic hierarchies by linking multiple nodes to a parent node labeled with an umbrella term. This functionality facilitates the organization and exploration of concepts within the data. The graph interface has been built using React Flow[1], a React component for building node-based interfaces, and has been heavily customized, including the ability to edit text labels directly within the nodes and define relationships between nodes through customizable edge types.

2.4 View Linking & Interaction

The various interface views provided by the front end, as described in Section 2.3, are linked together to provide user feedback during data exploration. When a user hovers over a data item in the data view, the corresponding item is highlighted in the frame view, and vice versa, as illustrated in Figure 6. Additionally, if the hovered item has an assigned label, this label is highlighted in the knowledge view. Conversely, hovering over a node in the knowledge view highlights all associated items in both the data view and frame view, as shown in Figure 7.

Labeling items is straightforward: users can select data items in the frame view using the lasso tool, which also filters the corresponding items in the data view, as depicted in Figure 8. They can then right-click on a node in the knowledge view and choose the "Assign Label" option from the context menu. Assigning a label triggers a k-nearest neighbors (k-NN) classification algorithm on the back end, which automatically reassigns labels to all data items that are either unlabeled or previously labeled by a k-NN classification. The reassignment process updates the label based on the most frequent label among the k closest data points in the 2D space of the frame view.

[1] `https://reactflow.dev/`

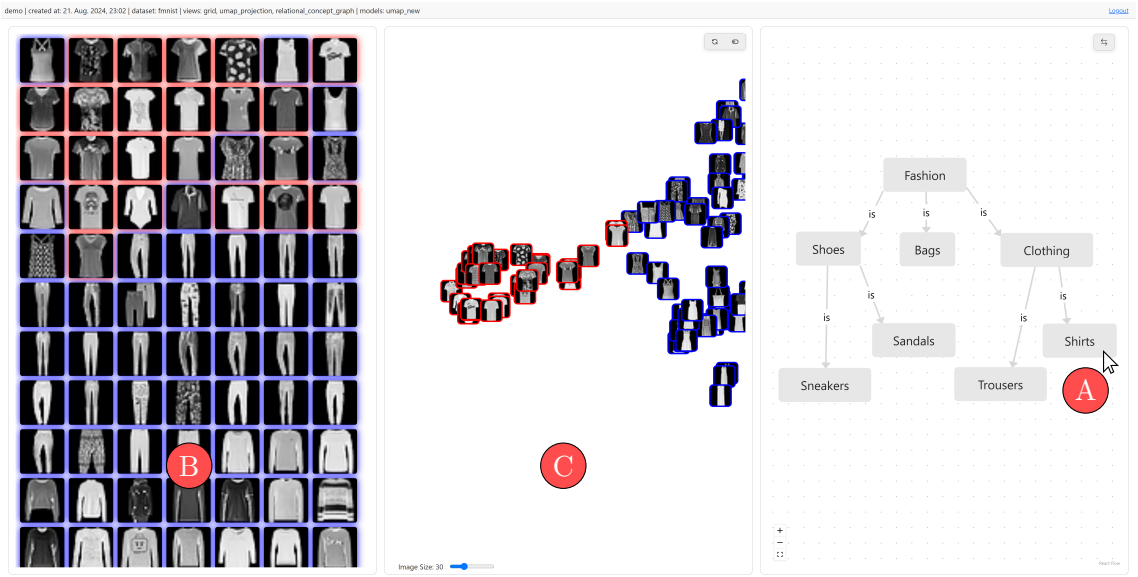


Figure 7: Illustration of the interaction between different views: Hovering over a node in the knowledge view (A) highlights the items associated with the respective label in both the data view (B) and the frame view (C). User-assigned labels are outlined in red, while labels assigned by the k-NN classifier are outlined in blue. Note that many k-NN-assigned items may still have incorrect labels.

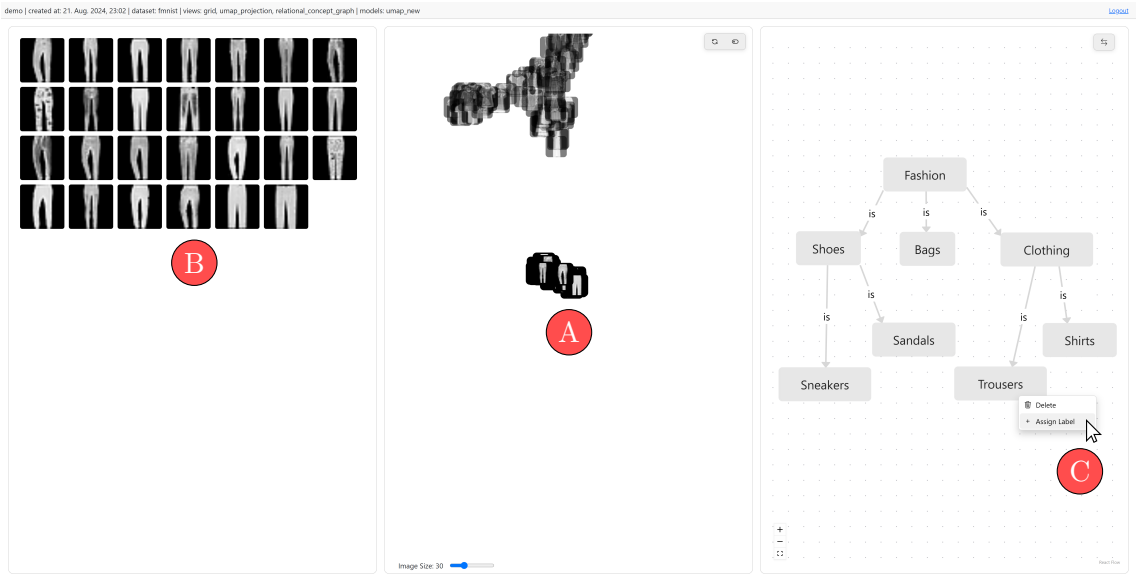


Figure 8: Labeling interaction workflow: The user begins by selecting the items to be labeled in the frame view (A), which also filters the corresponding items in the data view (B). Next, the user right-clicks on the desired label node in the knowledge view and selects the "Assign Label" option from the context menu (C) to complete the label assignment.

3 Conclusion and Outlook

The JDE framework offers a practical solution for tackling the challenges of exploratory data analysis in large-scale unstructured datasets. By integrating user interaction with machine learning techniques, the framework facilitates the discovery of both expected and unexpected patterns within the data. The triple-view visualization approach—comprising the data view, frame view, and knowledge view—provides users with a structured and intuitive interface for exploring data and aligning it with evolving models and concepts.

The initial implementation demonstrates that the framework can effectively support flexible and adaptable data exploration, especially for researchers and domain experts who require specialized analysis tools. The modular architecture, with configurable datasets, views, and models, allows for easy customization to meet specific needs across various applications. However, implementing new views or models currently still demands programming expertise and deeper understanding of the framework, which may limit its accessibility to a broader audience.

Looking ahead, there are several opportunities to enhance the framework's capabilities. Future development could focus on creating a more flexible approach for integrating new datasets and establishing a standardized interface for incorporating additional machine learning models to handle more complex data wrangling scenarios. Enhancing the user interface with advanced visualizations, improved interaction techniques, and refining existing elements could further improve the user experience.

In retrospect, some design choices, such as using files to store configurations or user data, may not have been optimal. Utilizing a database management system (DBMS) and providing a minimal user interface for manual data editing—initially a key reason for choosing file storage—could offer more efficient data handling and inherently support atomicity and consistency in transactions.

In summary, the JDE framework offers a solid foundation for exploratory data analysis in unstructured datasets. With further development, it could serve as a useful tool for researchers and data scientists, combining human insight with machine learning to uncover meaningful patterns.