

Meta-Storyboard Using Feature-Action Patterns: Concept, Techniques, and Case Studies

S. Khan^{1†}  S. Jones²  B. Bach³  J. Cha¹  M. Chen²  J. Meikle²  J. C. Roberts⁴  J. Thiagalingam¹ 
J. Wood⁵  and P. D. Ritsos⁴ 

¹Science and Technology Facilities Council (STFC), UK Research and Innovation (UKRI), UK

²University of Oxford, UK

³University of Edinburgh, UK

⁴Bangor University, UK

⁵City University, UK

Abstract

This paper introduces the concepts of meta-storyboard (MSB) and meta-storyboard authoring for data-driven storytelling. Traditionally, in data-driven storytelling, a storyboard is predefined for a given data set and the information an author wants to convey. However, such a static process does not account for dynamic data and does not scale up to the need to produce visual stories for different individual viewers and contexts. In a meta-storyboard, an author anticipates potential features that may appear in dynamically arriving or selected data and consequently defines visualization actions to convey explanations about these features. An MSB thus includes feature-action patterns as reusable design patterns for anticipated data features, viewers' requests, and application contexts. Through two case studies, we first developed the concept of MSB as a standalone software for communicating time series data in the context of COVID-19; then, we broadened the concept as an API to tell stories about machine learning workflows. MSB is complementary to traditional methods for authoring storytelling visualizations, while providing an efficient means to construct data-dependent storyboards for different data streams of similar contexts.

CCS Concepts

• *Human-centered computing* → *Visualization systems and tools; Information visualization;*

1. Introduction

Visualization provides a powerful means for telling stories about data [KM13, GP01, TRB*18, LRIC15]. This paper is aimed at authors and developers of data-driven storytelling visualizations who must support large, fast-changing, or highly heterogeneous data — settings where building a bespoke storyboard for every dataset, region, or viewer is no longer practical.

In most existing workflows, authors and developers of storytelling visualizations are typically given a complete dataset that is not expected to change once the storyboard is constructed (e.g., [SXS*21, OBCT24]). Fig. 1(a) illustrates such a workflow. This assumption breaks down as data volumes grow. In many applications, it is necessary to deliver data-driven storytelling visualization in a cost-effective manner. A storyboard should be able to efficiently handle numerous data items, especially since the author may not have the time to examine each one individually. Additionally, the storyboard should respond automatically to patterns

in dynamic data that the author cannot preview. It must also scale up to support various combinations of data selections requested by individual viewers. Viewers often pay more attention to stories relevant to them, such as events near their location, information that influences their decisions, or historical facts of personal interest.

These requirements became strikingly evident during the COVID-19 pandemic. Institutions, such as governments and news outlets, were able to transfer data at the national level to stories effectively. Meanwhile, many members of the public found it difficult to access information of more immediate interest, such as finding out what happened in their location, checking recent data related to their planned journeys, or comparing historical data between regions of interest. It would not be scalable to author many different storyboards for every individual region or every pair of regions. Nor would it be scalable to change storyboards manually, whenever a dynamic dataset is updated with newly-arrived data. Maintaining and repeatedly revising multiple storyboards would also create a substantial burden on developers implementing these storyboards.

Motivated by this gap, we argue that it is highly desirable for storytelling authors to use a new type of storyboard workflow, which

† Corresponding author: saiful.khan@stfc.ac.uk

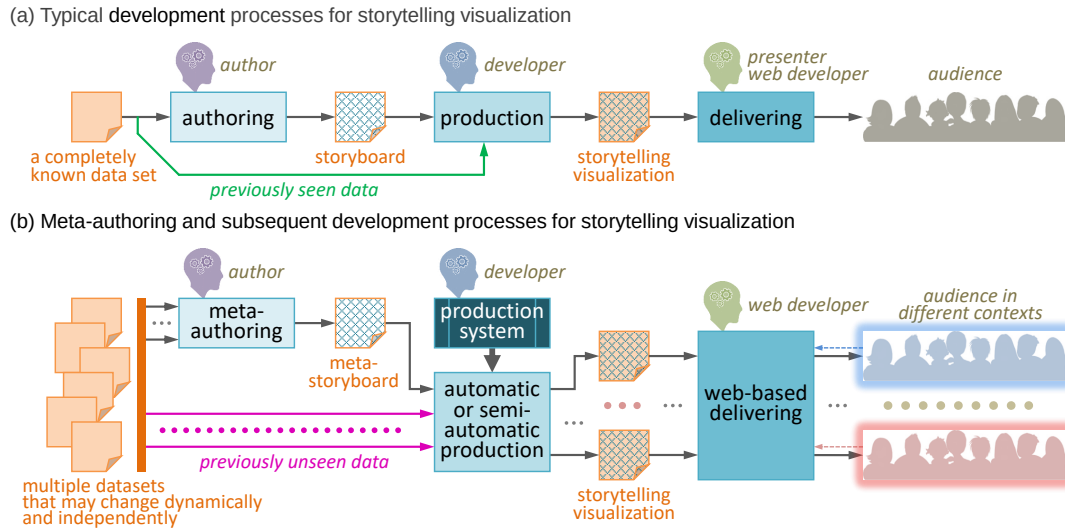


Figure 1: (a) In a typical workflow for creating storytelling visualization, an author defines a storyboard for a known dataset, which is then developed as a web-based visualization, usually for a specific target audience. (b) With our approach, the author creates a meta-storyboard that works with multiple, dynamic, and often not-yet-inspected datasets. The storyboard is converted by a developer, using the MSB API and following rules that facilitate the automatic or semi-automatic depiction of user-driven stories, for different target audiences.

lets them anticipate potential features that may appear in dynamically arrived or user-selected time series data, and define visualization actions for telling stories about those features once they occur. We refer to such a flexible storyboard as a *meta-storyboard* (MSB) and the process of creating it as *meta-authoring*.

In this paper, we outline the concept of MSB and describe the process of meta-authoring in Fig. 2. The core technique for realizing the MSB concept was first developed during the COVID-19 pandemic. As part of the RAMPVIS project [CARA*22, DARA*22] for providing a variety of visualization capabilities to support epidemiological modeling, a small team focused on devising a novel technique for storytelling visualization, which enables a meta-authoring workflow as shown in Fig. 1(b). In such an application, there were numerous dynamic time series data and individual viewers could request for a story about different pieces of said data entailing numerous combinations.

Although difficulties may be caused by the huge variation in selected sub-datasets (referred to as *data streams*), many characteristic patterns in these data streams are expected to be similar. Similar data patterns can be depicted using similar visual patterns. The key to developing an efficient mechanism for combining an MSB with individual data streams is to define, implement, and apply design patterns for storytelling visualization [BSB*18]. To avoid overusing the word “pattern”, we use the term “data feature” in place of “data pattern”, and “visualization action” in place of “visual pattern”. Hence, the design patterns in storytelling visualization are patterns of mapping feature to action, i.e., “feature-action” design patterns. Therefore, our contributions in this work are:

- the introduction of the concept of MSB, which employs feature-action patterns for the creation of stories from dynamic time-series data streams,

- an algorithmic solution and an implementation for realizing MSBs,
- two use-cases demonstrating the use of MSBs, one realizing stories for the COVID-19 pandemic, and one visualizing machine learning (ML) workflows, both followed by corresponding user-based evaluations.

The structure of this paper is as follows. In Section 2 we provide an overview of prior work in personal information visualization, storytelling, and automation in storytelling. In Section 3, we describe the system architecture and algorithm solution for realizing an MSB (Section 4). Technically, it is a novel addition to the emerging set of methods for automated storytelling (e.g., [WSZ*20,SKH*22]) and offers a scalable solution to produce storytelling visualization for numerous pieces of dynamic time series data. The storytelling visualization of COVID-19 data, from the RAMPVIS project provides an important case study to demonstrate the feasibility of the MSB technique. In Section 5, we report several example storyboards defined using the MSB and summarize feedback from various stakeholders. The detailed report of the user-centered evaluation can be found in Appendix E.

Following the first case study and its evaluation (Section 5 and Appendix E), we repurposed the software to provide storytelling visualization of machine learning (ML) workflows. In conjunction with the development of this second case study, we advance the MSB technique further by designing and implementing an open-source API (i.e., Meta-Storyboard (MSB) API, <https://www.npmjs.com/package/meta-storyboard>) to facilitate long-term development and broad application of the MSB concept and technique. We briefly describe the MSB API in Section 6, while providing further technical details of the MSB API in Appendices B, C, and D. To demonstrate the broad applicability

of the MSB and usability of the MSB API, we report the ML case study and its evaluation in Section 7.

2. Related Work

Storytelling and visualization have a long history [TRB*18], helping to reveal information in ways that are intuitive and compelling [GP01]. Employed techniques are wide ranging from sketching [LKS13], slide shows [HDR*13], comics [WRC*21] to investigating specific tasks in storytelling, such as linking [ZOM19], collaboration [LRIC15], immersion [WFRR20] and learning methods [TLW*21].

2.1. Personalizing Information Visualization

As many of our daily activities are mediated by some form of interactive technology, recording, sharing and utilizing data and related information about said activities has become more prevalent. In particular, data-driven information that can be personal, i.e., concern us either as individuals or some identifiable collective, is already driving our decision-making. For instance, Yousuf and Conlan explore personal visual-learning narratives [YC18]. During the COVID-19 pandemic, the degree of technological mediation increased, either as we started working and socializing more online, or as we tried to follow the pandemic's progression through visualizations on public media. Looking at daily, national, or local infection rates became a daily activity which often determined our daily or monthly routine and choices.

Nevertheless, most of the visualizations that we had access to provided either none or limited functionality when it came to enabling personalized points of view. For instance, dashboards such as the John Hopkins COVID-19 map [Cen22] provided only a world and national overview, whereas Governmental portals, such as the UK's Coronavirus dashboard [UK 22] allowed the selection of locations (via post code) but did not provide any form of contextualization, or a story. This is also true for visualization interfaces that present data on world events, concerning our day-to-day lives (e.g., world crises impact on food pricing) and often remain on national level averages. Even when some forms of limited personalization were offered (e.g., location selection), the storyboard for one time series could not be applied easily to similar time-series data streams from other sources.

Inevitably, providing more personalized storytelling requires:

- (a) the availability of data for a more personal point of view, e.g., specific to a location, person, communities, and
- (b) mechanisms to involve the users, and allow them to tailor their queries, explore the resulting visual depictions, and involve and engage them on a suitably personal level.

Our work takes steps towards this direction.

2.2. Storytelling Concepts and Temporal Data Storytelling

Data-driven storytelling aims to provide a curated lens on evidence in data. It does so through contextual information, a curated set of messages, compelling narrative devices [SH14], and

often a sequential ordering of information. It employs narrative patterns [BSB*18], such as *gradual reveal*, or *juxtaposition*, and communicates through a set of genres [SH10] such as videos [ARL*16], Data GIFs [SWT*20], slideshows [HDR*13, WLF*19], or data comics [BWF*18]. Visual narratives help to explain ideas [RBSN22], which are especially useful in education settings [YC18]. While studying data comics, Wang et al. [WWF*19] found that by breaking down the complexity of information into individual steps and presenting them sequentially (i.e., comics), readers could follow the story better, compared to infographics or illustrated texts. Feature-action data patterns create such sequences that a reader can navigate forwards and backwards.

A key point in storytelling is the relation between the story and the audience. In author-driven stories, narration, sequence, and content are defined by the author of the story, with little to no agency from the audience. In reader-driven storytelling, readers gain agency over the story and consequently can personalize it. Agency is usually achieved through interaction, e.g., by navigating within the story, data selection or free exploration [WRC*21]. However, one of the drawbacks of interactive storytelling is the discoverability of interaction affordance [Cox11, BEDF15], risking to impede personalized storytelling [MRL*17]. With our approach, we reduce the need for interaction and customization by creating prescribed stories for a given set of input data. As shown in Section 5, we keep interaction to a minimum and instead provide stories at different levels of granularity. While the sequence, narration, and contents is fixed, said granularity, which is informed by our algorithm (Section 4), defines how many messages the story has.

Storytelling of temporal data has a long tradition as demonstrated through two comprehensive surveys [RG13, BLB*17] which feature many historic examples of *time-lines*. Through their sequential nature, temporal data lends itself particularly well to sequential and author-driven storytelling [WaCP*15]. However, these bespoke examples make generalization to other datasets hard. Commercial tools for creating visual timelines include TimelineJS [Nor18] and most recently Timeline Storyteller [BLB*17]. Likewise, Timeline Curator supports the creation of visual timelines from text, extracting data from the latter, and enabling the curation of the visual timeline [FBM15].

2.3. Automation in Storytelling

A good number of systems allow authors to create storytelling visualizations interactively, e.g., [OBCT24, MBS24]. Meanwhile automation for storytelling aims at increasing the access to information by lowering the burden for humans to create bespoke stories. Some systems aim more at supporting analysts gaining an overview over their data and personalizing their interest through interaction [SXS*21], whereas other systems aim to support the creation of effective and compelling *communication material* [WSZ*20]. Other systems deal with fact extraction, as by Law et al. [LES20, SyzZW24]. Facts can be selected based on statistics and ordered based on a ranking score representing subjective relevance, interesting-ness, and importance [SXS*21, WSZ*20]. User interaction can enable dynamic data selection [WGH*24]. Our method works similarly in that it provides a specific routine for determining specific features, while ranking is defined by a Gaussian

function. To the best of our knowledge, all these systems explicitly focus on tabular data, whereas we focus on temporal data.

Storytelling systems may group visual information into panels based on category [DHPP17], create dashboards [WWZ*21], or generate infographic-like fact sheets of visualizations around topics, featuring textual explanations [WSZ*20]. To organize data facts into sequences, graph-based approaches create a similarity graph from all facts and consequently select a sequence of visualizations (a path through the graph) based on minimizing edge weight [YLRC10, KWHH17]. Data facts can also be ordered using a logic-oriented Monte Carlo tree search algorithm [SXS*21].

Closest to us, Parry et al. [PLC*11] presented an automated method to extract important frames from a video, for composing a sports game summary storyboard. They used a Gaussian mixture model to estimate the importance of individual frames. We adapted their technique for time series data. With our approach, data facts are extracted from the time series through analysis as well as complemented through public information on policy decisions. A Gaussian distribution defines which facts are shown for each given timeline and user-defined granularity. Consequently, timelines are built up step-by-step, progressing through identified breakpoints, while contextual information is displayed in textual labels. For our data, the narrative sequence follows the temporal order of breakpoints.

3. Meta-Storyboard: Concept

As illustrated in Fig. 1, a meta-storyboard (MSB) must support storytelling visualizations for data that the author has not previously seen. This poses non-trivial challenges to

- the authoring process,
- the underlying algorithmic pipeline for transforming an MSB and one or more data streams to storytelling visualization, and
- creating reusable software for the broad application of the MSB concept and technique.

In this section, we focus on (a) from the perspective of authors of MSBs. We will describe (b) in Section 4 and (c) in Section 6, together with further technical details in several appendices.

The authoring process of an MSB is referred to as *meta-authoring*, in which story authors (i.e., a *meta-author*) define narratives that can be applied in many similar, dynamic, and often not-yet-inspected data streams. As outlined in Fig. 2, a meta-author needs to (1) explore the data; (2) turn specific story items into a list of generalized data features, $\{F_i, i = 1, 2, \dots\}$, and (3) connect each feature, F_i , to one or a few actions, $\{A_{i1}, A_{i2}, \dots\}$. A developer then uses the Meta-Storyboard API (Section 6 and Appendix B) to transform the MSB into a piece of storytelling visualization software, which can process an unseen data stream automatically to detect any feature in the feature list, and react to each encountered feature, F_i , by activating the set of actions $\{A_{i1}, A_{i2}, \dots\}$ defined by the meta-author. Each mapping ($F_i \rightarrow \{A_{i1}, A_{i2}, \dots\}$) is a feature-action pattern that can be reused on many data streams, including new data arrived dynamically or data streams selected by users interactively.

To define the feature-action patterns, firstly, a meta-author inspects some known data and ascertains key features F anticipated.

One challenge that meta-authors face is to cope with the quantity, range of, and precise values of the available data streams. In contrast to 'traditional' storytelling visualization – where authors and developers would have direct access to the data – in meta-authoring, a meta-author may not have (or does not need to have) examined all data in detail. Therefore, it is essential for the meta-author to anticipate potential features that the data may or may not contain and to design a story specification that functions effectively in either case. While a meta-author may specify features and actions using plain language, the MSB translates them into a feature-action table that the MSB API can interpret (Section 4 and Appendix C). Then, the storyteller needs to express features F in a descriptive and generalizable form, facilitating the detection of said features without requiring knowledge of when (or whether) they occur in each data stream. An example would be to “find if the data in each month features a rising slope above 15°”.

Finally, the meta-author needs to consider what actions A occur for each encountered feature. Each action may highlight the feature visually and display a narrative (e.g., showing values and predefined text strings). A few examples of implemented actions can be seen in Fig. 3.

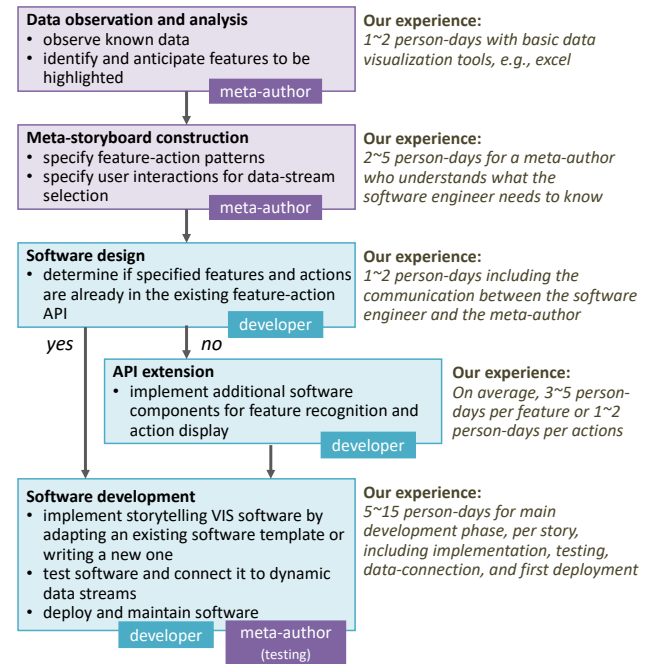


Figure 2: Overview of the proposed meta-authoring process, involving a story meta-author and a developer. The resulting storytelling visualization software can be applied to many similar, dynamic, and often not-yet-inspected data streams.

4. Algorithmic Pipeline

In this section, we describe the underlying algorithmic pipeline of our method. This section is supported by two appendices in the supplementary materials, both of which detail the programming details from the perspective of a software developer in implementing

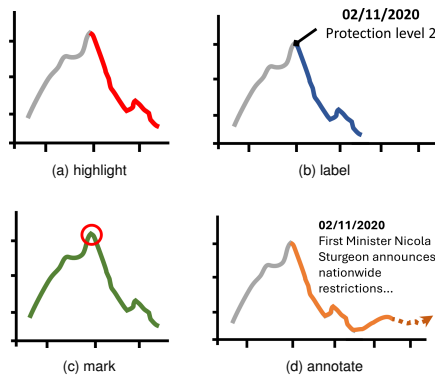


Figure 3: Actions in our implementation with their parameters: (a) change the color of a section of the time series line for highlighting. (b) annotate the graph with a line attached at a particular point. (c) draw a circle at a datapoint. (d) While animating a time series segment, place a text description on the opposite half of the graph. It is possible to only animate a time segment or animating a segment and annotating a point.

a meta-storyboard. The application programming interface, i.e., the MSB API, for supporting the method is described in Section 6 and further detailed in Appendix D.

Figure 4 provides an overview of the workflow of our method. The orange boxes in the top row show the human input, of the story viewer (data selection by virtue of story and location selection, fetched from the data repository) and the storyteller (remaining inputs). The light green boxes along the thick brown line are the data interface that translates the viewer's and storyteller's inputs to inputs algorithms can process. The remaining boxes, below the light brown line, illustrate the data flow and algorithmic process in the flow. We consider two categories of time series data:

Numerical time series (NTS) – This category includes commonly-encountered time series during the COVID-19 pandemic, such as daily, accumulated, normalized, and k -day moving average data, with semantics such as number of cases, hospitalizations, fatalities, vaccinations. Fig. 5(a) shows an example of a time series of the infection rate in Aberdeen City between March 2020 and October 2021. A feature detection algorithm that searches for peaks is applied to the data. Each detected peak is segmented and highlighted by a different color and its apex indicated by a black dot.

Categorical time series (CTS) – This category extends the notion of NTS by considering each data point at time t can have a categorical or nominal value. Assuming that “null” or “no value” is a valid categorical value, every data point in the time series has a value in the same way as NTS. Fig. 5(b) shows an example of a categorical time series, represented as a timeline featuring major healthcare policy changes, such as “lockdown”. The example details events between the period of March 2020 to October 2021. Each blue dot on the timeline represents a different semantic event.

4.1. Feature Detection

Feature detection is a widely used data analysis process in many applications, such as signal, speech, and image processing. In a sto-

ryboard about time series, there are some common features, such as “peaks”, “valleys”, “steep climb/fall”, data milestones, smoothness of data. Nevertheless, the design of a feature detection algorithm may not always be straightforward. For example, a “peak” may be defined as a local maximum point in a time series – there are usually a lot of them. Alternatively, it may be defined as a global maximum point – there is often just one such point, or if there are a few, they are unlikely to be distributed sparsely in different periods. On the other hand, a story author typically requires a more complex feature definition, e.g., a local maximum that is at some distance from other peaks. Furthermore, the way a human may identify a peak is different from how a computer mathematically defines one.

In our case, the feature detection algorithms required several rounds of adjustment to align with the goals of the story designers. An example of the results of our peak detection algorithm can be seen in Fig. 5(a). The algorithm selects possible peaks from the set of all maximal points in the data. In decreasing height order, we traverse down the slopes to the left and right of each maxima until a minimum point is reached. In doing so, the bounds of the peak are found and any maxima along the way, considered part of the same peak, are removed from the selection pool. The data is segmented into different peak regions, as can be seen in Fig. 5(a), with different colors representing different peaks. Algorithms for detecting different features in an NTS assign categorical values to the data points at different time points. These categorical values collectively define a CTS.

4.2. Importance Ranking

Given a CTS with many different categorical values, a story author will often place different levels of emphasis on different data points according to their categorical values. For example, a “full lockdown” event may be considered more important than a minor policy change. Similarly, given an NTS, following feature detection processes, the detected features can also be associated with different levels of importance. As shown in Fig. 4, we conveniently refer to the importance ranking for a CTS as $\text{feature}_{\text{semantic}}$ ranking, and that for an NTS as $\text{feature}_{\text{data}}$ ranking. In our system, importance ranking processes allows real-value ranking with a maximum value ($r_{\text{max}} > 0$) that must be consistently defined within each story. For all our case studies, we defined $r_{\text{max}} = 10$. The black bars in Fig. 5(c) indicate the importance ranking values assigned to each peak feature detected in Fig. 5(a), and the ranking takes into account the height of the peak.

Because the importance ranking is based on categorical values of events and features, the ranking is somehow discrete and the interactions between different ranking values (e.g., the black bars in Fig. 5(c)) are difficult to compute numerically. We thus assign a *Gaussian component distribution* (commonly referred to as a *Gaussian* for short) to each ranking value as Gaussians indicate an interaction between two features/events and the level of interaction can be numerically computed. In many cases, there could be many interactions, such as Fig. 5(d), where each Gaussian corresponds to each categorical values (except null) in Fig. 5(b).

In storytelling visualization, Gaussian mixture models were previously applied to video data [PLC*11]. As Gaussians are continuous curves in $[-\infty, \infty]$, we can use Gaussian mixture models to

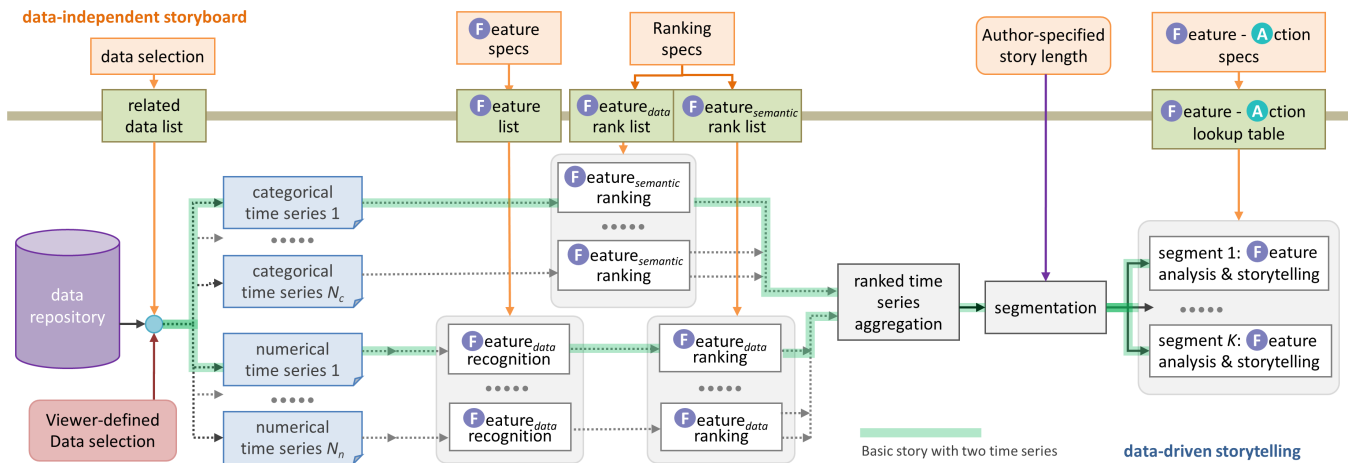


Figure 4: Overview of our system's workflow. The author builds a story by choosing data that depends on a particular dataset from the repository. They then define features, determine the ranking (importance) of these features and map detected features $\mathbb{F}$ to specific actions $\mathbb{A}$. They also define the effective length of the story ($\square$). These are converted by the developer in lookup tables and lists used by the algorithm (steps denoted in $\square$). The lists are then used to process the different categorical and numerical time series (steps denoted in $\square$), by detecting and ranking the defined features (steps denoted in $\square$ stages). The time-series are then aggregated, and segmented based on the author-specified story length. Once the user selects the location(s) of the story (step denoted in $\square$) the story commences, and actions are invoked when any corresponding features are detected. The algorithmic pathway for the simple story, described in Section 5, is highlighted in green. Further technical details, from the perspective of a software developer, can be found in Appendices C and B in the supplementary materials. The MSB API, which provides various software components at the system level (i.e., below the thick brown line) is described in Section 6 and further detailed in Appendix D. The MSB API is open-source software (<https://github.com/saifulkhan/meta-storyboard>).

obtain a continuous importance curve by combining all Gaussians within a time series and an overall importance curve by combining the importance curves of all NTS and CTS related to a storyboard. For example, in Fig. 5(e), the light red time series is the mixed importance curve of the Gaussians in Fig. 5(c), while the light blue time series is that of the Gaussians in Fig. 5(d). A $\max()$ Gaussian mixture function is used in both cases. These two mixed importance curves are then mixed using a $\text{mean}()$ Gaussian mixture function, yielding the purple overall importance curve that encodes the importance features in Fig. 5(a) and events in Fig. 5(b).

4.3. Timeline Segmentation

Once we have obtained the overall importance curve for a storyboard, we can start the process of generating a storytelling visualization. Similar to telling a story in speech or writing, we would like to select relatively more important features to be “mentioned visually” in a story according to the time or other resources available to the story. In this work, we consider two types of resources, the number of button pressing actions in interactive story progression, and the total animation time in fully automated story progression. If there were too many button pressing actions or the animation were too long, the viewers would become weary or impatient. If there were too few interactions or the animation were too short, the viewers may find the information provided inadequate. Technically, we would like to divide the timeline of a storyboard into k sections. In interactive story progression, k sections require k button pressing actions (including the start). In automated story progression, k determines the total animation time as $k \times \text{“unit section time”}$. Fig. 5(f)

shows two possible segmentation results with $k = 3$ and $k = 5$ respectively.

We developed an algorithm to segment the timeline of a storyboard by considering the peaks in the overall importance curve of the storyboard. The algorithm selects the top $k - 1$ peaks with a maximal gap $\Delta_{\max}$ between neighbouring peaks. In this way, we ensure that the most important temporal points are selected and located at the section boundaries, where we can insert more visualization actions. With interactive story progression, users can spend more time to view the current visualization before pressing the button for progression. In automated story progression, we can slow the animation at each boundary between the two sections.

4.4. Feature-Action Profiling

As aforementioned, the output of the meta-authoring process is a feature-action specification expressed by the story author in different ways (e.g., textual descriptions, pseudocode and sketches). A developer translates the specifications to feature-action data patterns in a lookup table, where features are categorical labels of all features that might be detected in the NTS and CTS selected by story viewers, and actions are calls to appropriate functions of the software components, for displaying various visual artefacts (Fig. 3) as well as exhibiting the desired software behaviors in terms of interaction, graphics, and animation. How long the story is depends on the number of segments of the Gaussian curve and the features present in the lookup table. A story designer can extend the time a story takes to complete, by increasing the number of features in the table to pick up on. This timing can also be controlled using

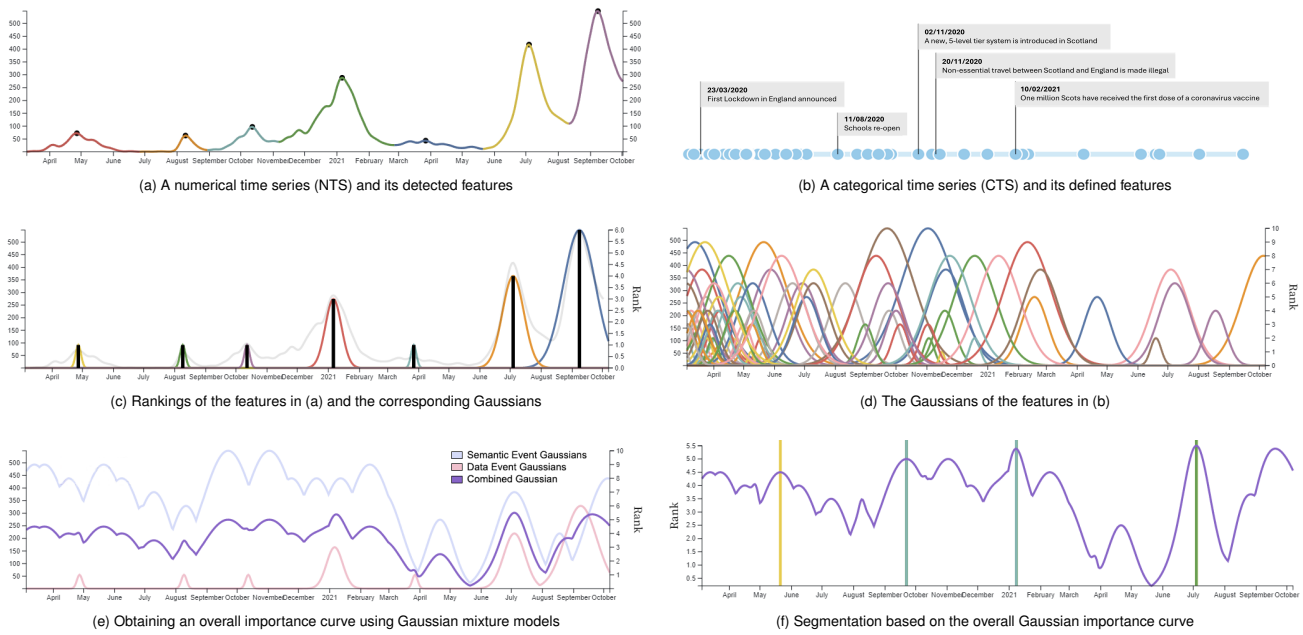


Figure 5: Graphical depiction of the algorithmic pipeline for automated storytelling. (a) Numerical time series (NTS): Detection of peaks, rising, and falling segments. (b) Categorical time series (CTS): Predefined, ranked event features. (c) Each detected data feature is given a rank (illustrated as a black bar). Each ranking value is converted to a Gaussian curve. (d) Likewise, ranked semantic events are converted individually to Gaussian curves. (e) Gaussian curves are combined using Gaussian mixture models. Here, a max-model is used within a time series and a mean-model is used between time series. (f) The story is divided in segments according to the combined importance curve. The number of segments is defined by a meta-author. In this example, the three green lines show the results of the segmentation algorithm, and four yellow lines indicate the segmentation results if five segments are required (three yellow lines happen to coincide with the green lines).

the feature rankings. For instance, to decrease the time to complete a story one could restrict the algorithm to showing only the most important events, say the top three events in a segment or maybe just those with a ranking of 8 or greater. The algorithm for feature-action profiling visits each segment one-by-one. Within segments each feature is visited in chronological order. Each time, it refers to the feature-action lookup table to check if the current feature matches any entry. If so the visual action is initiated.

4.5. Generation of Storytelling Visualization

After all sections have been profiled, the “play” button becomes available to users. The progression mode, whether interactive or automated, is specified by the meta-author during the construction of the MSB and it enables a user-configurable setting in the visualization interface. In the mode of interactive story progression, a viewer presses the button to activate the storytelling visualization of a section. In response, the underlying algorithm visits the NTS and CTS of user-selected region in the order of time. For any feature (or event) that is marked as being included in the story, the algorithm invokes the corresponding action. When the end of the section is encountered, the algorithm waits until the viewer presses the “play” button, to proceed to the next section. In the mode of automated story progression, instead of waiting for a viewer’s interaction, the algorithm waits for a short time span, and continues to the succeeding section automatically.

5. Case Study One and Evaluation

In order to demonstrate and evaluate our approach, we created three use-cases using MSBs, intended for the RAMPVIS project. The purpose of these stories is twofold: a) to demonstrate how MSBs were used to create stories for the pandemic, as examples, and b) as focal points of a preliminary evaluation of our approach, from said stories’ creators, described in this section. Our three stories, detailed in this section, are: (1) a story from single location with one time-series (Section 5.1), (2) a comparison story (Section 5.2), and (3) a scrollable time-line story (Section 5.3).

We requested written feedback on these stories from different stakeholders of our method, using a shared online document. We collected their comments as evaluation and reflection, similar to a retrospective verbal protocol analysis. The stakeholders involved were: (i) story authors who created said use-cases (about 1400 words), (ii) software developers who were directly involved in developing the pipeline in Fig. 4, various feature detection algorithms, and visualization components (about 1200 words), and (iii) public members and one public engagement expert (about 900 words). The first two roles are the intended users of our approach, whereas the latter were chosen to provide insight on the viewer-centric nature of our method’s outcomes, as part of the RAMPVIS project activities. While we provided five questions for each group as prompts, we explicitly directed participants to “please feel free to add any comment.” The full text of the prompts and feedback

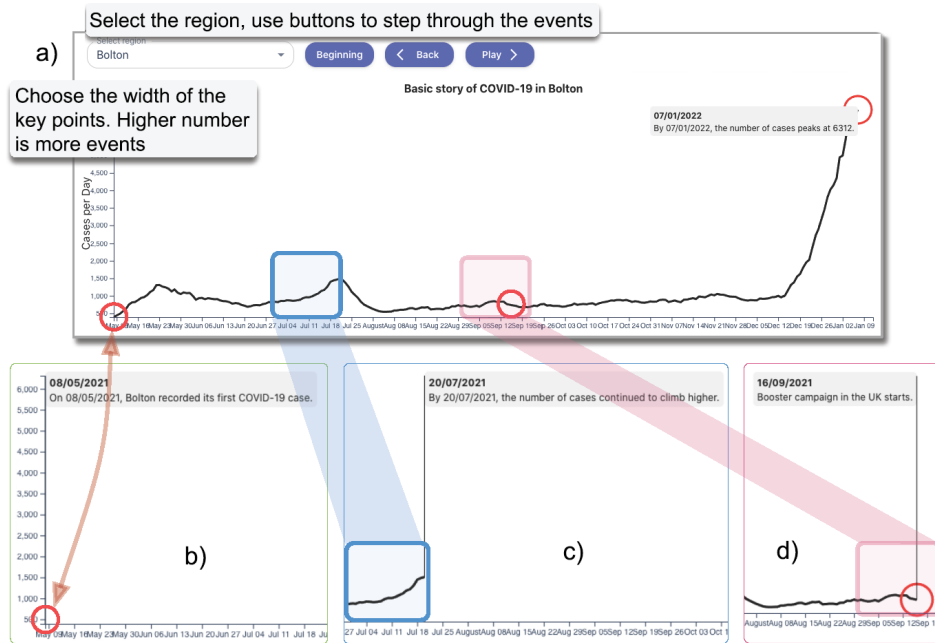


Figure 6: Demonstration of the single-location story. Programmed features are located. When the Play button is pressed the delivery system actions to progress to the next feature, the appropriate text is concatenated with real data, and placed in a suitable position (if there is space it will be shown to the right of the vertical line, otherwise to the left). (a) The full interface; (b) story start, (c) highest deaths per day, (d) booster vaccination program starts.

is given in Appendix C in the supplementary materials. The main feedback points are summarized in Section 5.4, grouped in themes, with direct extractions from stakeholders' input in *italics*.

5.1. Single Location and Time Series

The first story (Fig. 6) focuses on one location, the number of positive COVID-19 cases, and major events such as lockdowns and vaccination progress. The story involves two time series: a) COVID-19 case data, which is continuous and spread over time, and b) related events, categorized in terms of importance.

In this case, the viewer can progress/rewind the story via buttons. An importance ranking value determines which events will be displayed, i.e., whether all or more significant events are displayed. The ranking of these events is decided by the author, whereas the ranking mechanism is explained in Section 4.2. As features are encountered and actions initiated, text descriptions are concatenated from meta-author defined text strings, with location and data values from the data enriching the story text. This is shown in a labelled text-box linked directly to the specified feature (Fig. 6(c)).

This story's meta-author used a spreadsheet to organize and map features and actions, which can be a suitable approach for several reasons. Exemplar features can be added to the spreadsheet and then edited into general ones. The order of the features can be readily re-arranged, whereas different priorities that help to identify the importance and order of the story elements can also be added to the spreadsheet. In addition, it is simple to map features alongside actions, as different cells. Furthermore, code can be translated and aligned closely to the spreadsheet content, or even parse it.

5.2. Comparison Storyboard

The comparison story (Fig. 7) allows viewers to compare two different locations. In our example, a viewer selects Bedford (100,000 inhabitants, 50 miles from London) and Bradford (350,000 inhabitants, 150 miles North of Bedford), both in England, under the same COVID-19 restrictions and reasonably close; yet on different rail routes to London and distant enough to be separate ecosystems. This could indicate when the pandemic was moving towards or away from London.

Similar to the previous example, each line-graph corresponds to a location. Because there are two time series, the comparison between them is the key narrative element. The meta-author defined the same set of features to be detected for both locations. When the detected features were visualized, it appeared to alternate between features in the two line-graphs. Such features are relevant to the characteristics of COVID-19 waves; e.g., cases increased in two weeks. The meta-author defined actions according to features. Messaging for this story has two main goals: (a) to highlight features on each time series, in a similar manner to the previous story, and (b) to highlight comparisons between the time series [GAW*11].

This example explicates several key facets. First, the story starts with the region that has the first significant event; in this case, it starts with Bradford as shown in Fig. 7(b). Second the animation automatically alternates between regions and significant events. The meta-author designed the story of each city around its major COVID-19 waves, which are identified by matching peak-related features in the data. The meta-author instructed that events for both cities were to be grouped around the same wave and to be played

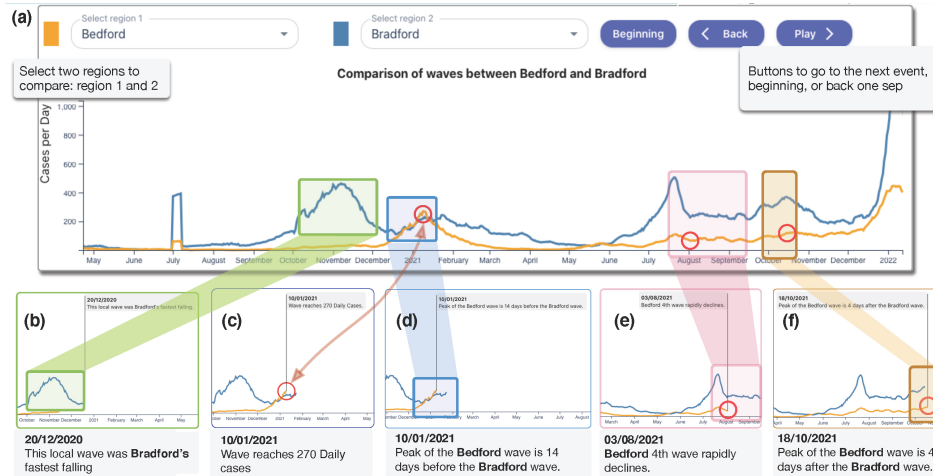


Figure 7: Comparison story demonstration, where (a) depicts the final frame. The story is shown in stages, moving key ‘features’, and alternating ‘actions’ between region 1 and 2. The insets (b-f) depict several key event features, which are incrementally shown as the story progresses; (b) a single feature and action about Bradford (region 2); (c) story action focusing on peaks, with data specific to the local site; (d and f) comparison feature showing differences in terms of days; (e) feature comparison based on calculated data.

in the order of waves. With such grouping, comparisons can be made between the two cities. For example, in Fig. 7(b) the story focuses on Bradford, and then in Fig. 7(c) the other region (Bedford) catches up. Later, the developer implemented this instruction by processing of features in two time series concurrently. The meta-author also wished that the highest ranked features were shown and highlighted. Our example depicts a few events from one region, before moving to events from the other. Regions with higher ranking, or higher data values, will play first. Third, we compare events; whether they are different, same or larger. We look at significant rises, declines and peaks; see Fig. 7 (d), (e) and (f), respectively. For instance, Fig. 7(d) shows how the text explicitly compares two values (saying that Bedford was 14 days before Bradford), which hints that the wave is moving from the smaller city (Bedford) to the larger (Bradford). The point of interest is indicated with red circle and a vertical text label (generated from the data).

5.3. Scrollable Storyboard

Our third demonstration provides a scrolling story of the events in England and includes three additional facets. First we display data as a line-graph, which provides an overview of all events in the storyline. Second, we present text descriptions for the current event and elided versions for the previous and the forthcoming event. The text scrolls as the viewer plays the story. In this manner, the current story text is shown in the context of what has happened, plus a hint of what is to come (Fig. 8).

The scrollable story was designed with two abstractions in mind: (a) the detail of individual, and (b) the overview of all, features (cases and deaths time series). The design was captured using the FDS [RHR16] method. Features were determined in relation to either cases, deaths, or both, and an overview is depicted in the event-line. Each highly-ranked feature is highlighted by a circle, with the

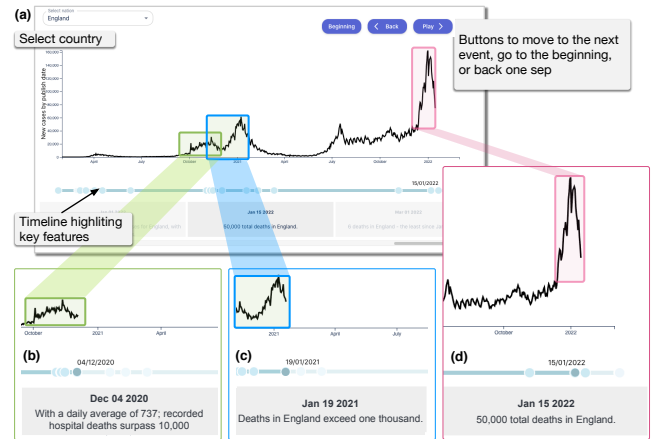


Figure 8: The country story (in this example focusing on England). (a) Shows a screenshot near the end of the story. Insets (b-d) highlight three features. As the reader plays the story, the line-graph animates to reveal the next part of the story, along with the text (sliding to reveal the next text). The text narrative shows the current text bracketed by the previous and next story text.

current one darker. Current, previous and next event text descriptions are updated for each event.

Important features, such as “first recorded case”, number of deaths or hospitalization cases in tens of thousands, rise in cases by 20% and so on, can be automatically detected respectively. By defining features with notable pre-defined values it is possible to signify interesting events from the time-series data as well. For example, people will be interested in the first case, when cases (or deaths) reach 100, 1000, and so on. Finally, events such as lockdown start/end dates, vaccination start dates, and quotes from parliamentarians can be ranked as in the previous examples.

5.4. Feedback on Authoring MSBs

Meta-authors indicated that crafting meta-storyboards can be more challenging than telling a story about a single dataset, as they needed to *think about the story holistically*, and how different data, say from different regions, may *support key messages*. They argued that *generalizing the story is definitely a gain from using this approach*, and that this was *a process of building-up, e.g., creating a simple story with one time series, then adding more time series into consideration*. They added that creating feature-action patterns can be *somewhat challenging*, as *some key features were obvious, such as a peak, rise and fall yet not all were easy to define*.

Meta-authors mentioned that this approach made them consider *the progression of the story in a more “mathematical” way*, and they felt that *using meta instructions enables core elements of a story to be exposed that might not be created by hand*. On the other hand, story authors mentioned that they were conscious that *information needed to be abstracted or generalized into instructions for software developers*.

5.5. Feedback on Transforming MSBs to Visualizations

In terms of how an MSB can be converted to a story-telling visualization, developers indicated that it was not difficult to support the story authors. Given a storyboard and visualization guidance from the meta-authors, *a programmer can create the storytelling visualization easily. The notion of feature-action design patterns was accepted by both meta-authors and developers*. The developers were able to *translate meta-authors’ qualitative specification of design patterns to the actual implementation in a look-up table relatively easily*.

Developers commented that *as the visualization components (e.g., multi-line time series plots, annotations, timelines, animation) are implemented using d3.js [BOH11] as reusable classes and functions [BOH11], developing new software components [...] had been quick and easy*. They also expected that *gradually, the time for implementing new software components for a new story can be reduced*. (cf. Section 6).

In terms of how long does it take to develop the necessary feature detection algorithms, this *varied from taking a few hours to days*, whereas for bespoke software components that implement actions, while it is not hard to implement them individually, *it can take up to a week to put them together for a story*. Among our stories, *the longest development, for an entire story took roughly four days*. Both developers and meta-authors commented that Observable was *very good for “drafting”*, however, placing many dynamic data streams on Observable was not feasible and a dedicated storytelling visualization infrastructure, that keeps the dynamic data streams, had to be developed and deployed (cf. Section 6).

Finally, both story authors and developers acknowledged that *there are opportunities for further automation of the current workflow*, whereas story authors commented that the approach of having *software engineer in-the-loop* may work for *large organizations*, but independent meta-authors or those in *small organizations* can benefit from meta-authoring tools that would allow them *to create these feature-action design patterns. One could envision an inter-*

face where storytellers can pick and match story segments and combine them sequentially or hierarchically (such as the one described in Section 6). Feature-action abstraction can be exploited to create a more modular approach for meta-authors.

Developers and meta-authors have suggested a number of features that could be incorporated in a larger infrastructure that supports meta-storyboarding, including (i) *visual programming environment*, (ii) *object-oriented/template-based programming environment*, (iii) *scripting language*, (iv) *markup language (e.g., XML)*, (v) *declarative data interchange format (e.g., JSON, YAML)*, and (vi) *form-like webpage*. Further requirement analysis and technology development will be needed, while the algorithmic pipeline, the algorithms, and *the software components developed in this project can provide a good basis for future developments*.

5.6. Feedback on Storytelling Visualizations Created with MSBs

The public expert in our evaluators pool highlighted that the stories produced to serve multiple data streams with a single meta-storyboard had added *human interest*, because *the stories are told linearly and from a local area perspective* and that *the user can situate themselves both in place and time in relation to the events*. They commented that this was contrary to many time series we saw during the COVID-19 pandemic that did not provide sufficient context, or easily digestible segments which would allow the viewer to *look at the situation leading up to an event and following after it, and to understand/reflect on each segment before moving on to the next*. The expert added that *the stories are likely to appeal to members of the public who are regular news viewers, who listen to scientific podcasts, and who take an interest in the local or regional context of the pandemic and how policies or actions affect its progress*. Nevertheless, they also added that *as this is a very visual technique it may hinder the engagement with some groups of the general public – and the format will not appeal to all*.

The non-expert members of the public indicated that if this form of storytelling was *available some time in 2020 (and perhaps the early part of 2021)*, it would get used a lot. They also added that *because it takes only one selection button for a region and then the play button to start, it is actually slightly easier than many search interfaces*.

Both public members and the engagement expert indicated that *this form of location-dependent, data-driven storytelling would be useful and of interest to members of the public*, and could be used to visualize *housing, energy and fuel prices, school performance, crime rate, and transport statistics*.

6. Meta-Storyboard (MSB) API

The concepts of meta-storyboard (MSB) and meta-authoring (Section 3), along with the algorithmic solution (Section 4), enable the reuse of feature-action design patterns across different data streams within the same context, even for data streams that have not arrived or been seen by the meta-author. It is necessary to make a further technical advancement to enable the concept and algorithmic solution reusable across different applications.

The algorithmic pipeline described in Section 4 was first developed using Observable. It was re-implemented as part of the RAM-PVIS infrastructure [KNA*22], using TypeScript, d3.js, and React.js. During the evaluation session reported in Section 5 and Appendix E, the developers indicated that the software support could be better (Section 5.5). In particular, we would like to consolidate the reusable software components for supporting meta-authoring in different applications. This motivated us to reorganize the existing feature and action components into a better structured API, while adding new feature and action functionality. This effort led to the Meta-Storyboard (MSB) API, which provides open-source and reusable software to the public while having much improved functionality and documentation in comparison with the previous two versions of the software.

The MSB API was written in TypeScript. The visualization functions were developed using d3.js. The API was implemented using object-oriented principles, where the abstract base classes serve two purposes: defining high-level API abstractions and enabling extensibility. The MSB API is divided into components, each implementing a set of distinct functionality. It provides story developers with:

- MSB Features — functions to specifying features to be recognized (e.g., peaks in time series);
- MSB Actions — activating drawing actions (e.g., circles, nodes, or text boxes);
- MSB Plots — drawing plots (e.g., line chart, parallel coordinate plot), and activating animated actions;

Likewise, the components for internal automation are:

- MSB Search — controlling the search for specified features;
- MSB Gaussian — mixing different importance curves using Gaussian mixture models.

All components of the MSB were implemented as TypeScript classes. Table 1 summarises the classes and their functionalities. For further implementation details, including source code and documentation, please refer to the repository at <https://github.com/saifulkhan/meta-storyboard>.

In Appendix B, we provide beginners' guidance for installing and programming with the MSB API. A simple MSB program allows a new MSB developer to familiarize the main workflow for implementing a simple MSB, which can serve as a template that can be modified to build more complex MSB programs.

One way to increase the complexity of storytelling visualization is to introduce more feature-action patterns in an MSB. In Appendix C, we provide further details about feature-action tables.

For readers who are interested in various technical components of the MSB API, Appendix D provides a quick tour of these components. The installation and maintenance of the MSB API is supported by the Node Package Manager (NPM) <https://www.npmjs.com/package/meta-storyboard>.

In the next section, we report another case study where the meta-storyboard was implemented using the MSB API, demonstrating the capability and flexibility of the API in supporting different applications.

Table 1: Overview of the main TypeScript Classes in the MSB API and summary of their functionalities.

Classes	Summary
Search, Utils	Class for detecting features and utilities for processing, manipulating, segmenting, etc., time series data (discussed in Sec. 4.1).
Feature, NumericalFeature, CategoricalFeature, Current, Fall, First, Last, Max, Min, Peak, Raise, Slope	Detected feature representation classes (discussed in Sec. 4.1).
Gaussian	Class for fitting Gaussian curves and Gaussian mixture model (discussed in Sec. 4.2 and 4.3).
Action, Circle, Dot, TextBox, Connector, ActionGroup	Classes for drawing and animating visual actions and annotations (discussed in Sec. 4.4).
Plot, LinePlot, MirroredBarChart, ParallelCoordinatePlot	Drawing plots of the stories (Sec. 5 and 7).
PlayPauseController, SyncPlotsController	For controlling and synchronizing animation behaviors in visualizations.
ActionFactory, FeatureFactory, FeatureActionFactory	The factory pattern to translate to action, or visualization objects based on the feature-action table.

7. Case Study Two and Evaluation

While the aforementioned meta-authoring method for creating storytelling visualization was developed during the COVID-19 pandemic, we have always been keen to apply the method to other domains, and demonstrate its generalizability. An opportunity arose when one of our visualization (VIS hereafter) researcher joined a ML team in the Science and Technology Facilities Council (STFC) in the UK. The ML team has been providing physics, nuclear, space, and astronomy scientists with ML models for processing a variety of data captured using different sensory modalities (e.g., [HBJT20]). One R&D strand of the team has been to develop and analyze ML benchmarks as the basis for providing guidelines and best practices in AI for Science [TSFH22]. In particular, analyzing these ML benchmarks required handling large-scale multivariate temporal data, motivating us to extend our MSB framework to support such analyses.

7.1. Requirements Analysis

Two authors from the VIS research team met the ML team leader and identified general requirements for using automated visualization tools in ML workflows, as well as several specific requirements for using storytelling visualization. As the team always has a few

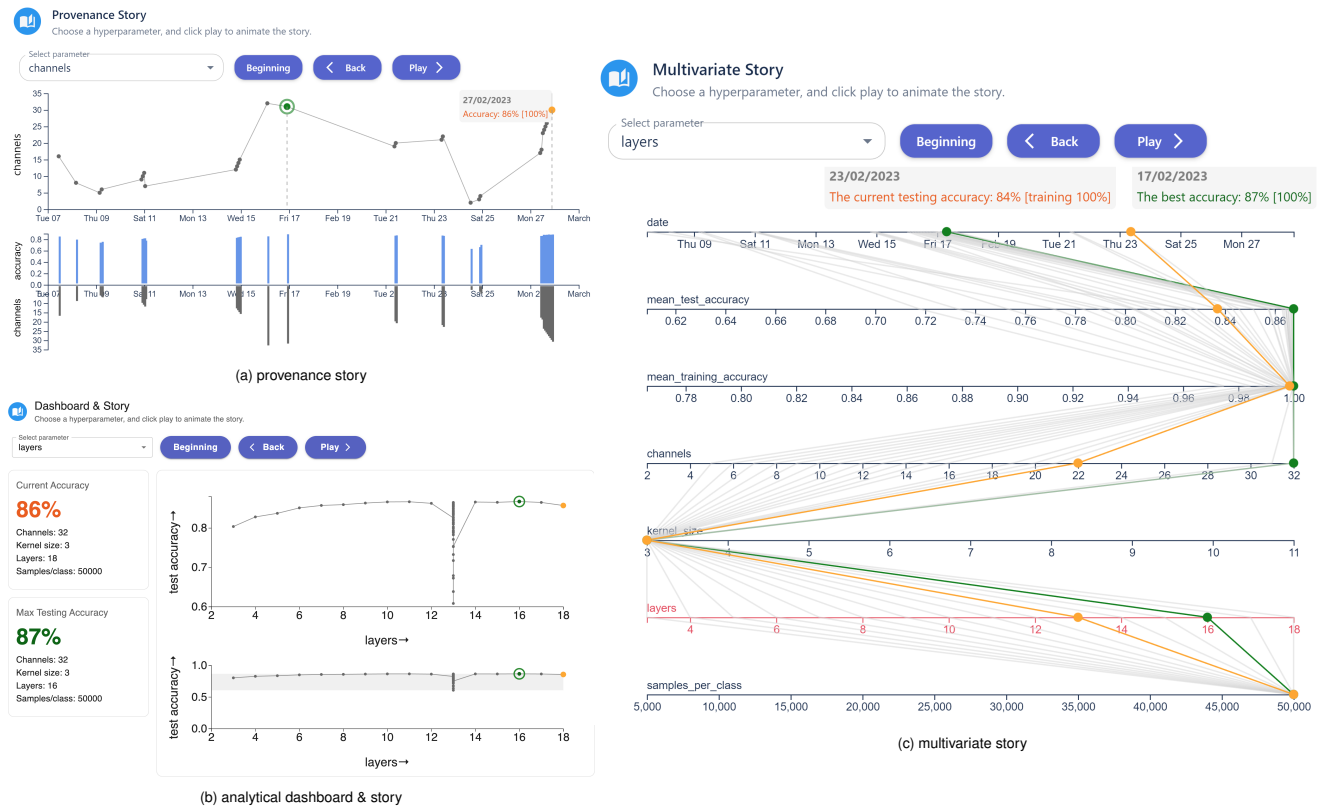


Figure 9: Screenshots of three stories implemented for supporting ML workflows developed using our approach: a) a provenance story focused on a hyperparameter, b) an analytical dashboard and story allowing ML researchers to visualize logged events, and c) a multivariate story about multiple hyperparameters visualized in a parallel-coordinate plot.

ongoing ML workflows concurrently and each may last up to 18 months, they need to observe their progress and provenance frequently, preferably through dynamically updated stories. In other words, after every training-testing iteration in an ML workflow, a visualization tool can automatically create a few stories about the results of the iteration, and provide an ML developer with useful information as if there is a helpful visualization assistant who is able to give an overview, highlight key information, and pause for interaction. Meanwhile, developing such tools for a number of ML workflows can present opportunities for testing and evaluating the MSB API described in Section 6.

Following this meeting, one VIS researcher collaborated closely with an ML researcher working on ML benchmarks. The VIS researchers designed and implemented two storyboards initially as shown in Fig. 9(a, c). During one evaluation meeting, we identified the need for an analytical dashboard with storytelling on demand. This became a new storyboard, shown in Fig. 9(b).

7.2. Meta-Storyboards

The STFC ML team has an established process for storing log data for ML training and testing. When a model is tested, each logged event consists of time series data such as date, time, hyperparameters, and accuracy measures. To repurpose the meta-authoring soft-

ware developed in the context of COVID-19, we focused on log data featuring time series.

A **Provenance Story** (Fig. 9(a)) was designed around the provenance and progress of an ML workflow. A viewer can select a story to be told from the perspective of a specific hyperparameter. The animated visualization shows the logged chronological events and focuses on those related to the selected hyperparameter. Interesting data features include, for instance, when the accuracy has a significant improvement or reaches a peak point. Visual actions (e.g., text message, color highlighting) are triggered by such features.

An **Analytical Dashboard & Story** (Fig. 9(b)) enables individual ML researchers to visualize logged events in both static and storytelling manner. Whenever a model-testing process is completed, the ML researcher concerned needs to observe the new logged event. To prevent the storytelling mechanism from slowing down the routine VIS tasks, the storyboard makes static visualization the default, with a visual design that focuses on the ordered values of a selected hyperparameter rather than event date and time. Drawing on our experience of working on COVID-19 data [KNAR*22, BFAR*23, KNA*22], we introduced elements of dashboard design to improve the efficiency of routine VIS tasks. The storytelling mechanism can be activated when ML researchers wish to be reminded of the provenance or be told stories about individual hyperparameters.

Finally, a **Multivariate Story** (Fig. 9(c)) tells stories about multiple hyperparameters through a parallel coordinates plot (PCP). Viewers can select a specific hyperparameter to inform the importance ranking of features (see Section 4.2). The animated visualization shows the logged events chronologically, but unlike Fig. 9(a), viewers can see all hyperparameters in the same visualization. As the visualization is more complex, the feature-action patterns involve more frequent pause-and-continue actions.

7.3. Stakeholder Evaluation and Improvement

In the STFC ML team, the VIS researcher was in contact with ML colleagues regularly, discussing requirements and design options while conducting frequent bite-sized evaluations. In addition, there were two formal evaluation meetings. In the first meeting, ML researchers were shown a first-draft working version of the Provenance and Multivariate Dashboard stories (Fig. 9(a,c)), which prompted them to make a number of revision suggestions. We also noticed that different team members suggested a revision on the requirements of the Provenance Story. Moreover, the team wanted to observe the progress and provenance of all ML workflows in a single storytelling visualization. Finally, individual ML developers wanted to observe new results quickly whenever a trained model has been tested and observe the provenance data on demand. We decided to create a separate storyboard for these, i.e., the Analytical Dashboard & Story (Fig. 9(b)).

Although that meeting was the first time when the ML researchers worked with a PCP they found the Multivariate Story particularly useful for observing the combined impact caused by different hyperparameters, without much explanation by VIS researchers. This was somewhat unexpected, as many novice visualization users often encounter difficulties when working with PCPs. We hypothesize that as storytelling shows data-events line-by-line, it makes the concept of PCPs more intuitive. Animation may also reduce the confusion caused by cluttering, while enabling viewers to focus on interesting events without much visual searching effort.

In the second evaluation meeting, we focused on the Analytical Dashboard & Story (Fig. 9(b)). The ML experts were enthusiastic about the dashboard and discussed the ideal data to be shown in the information boxes, and the interesting features in the plot, so that the storytelling mechanism would highlight them automatically.

With the MSB API, the specification and implementation of each storyboard shown in Fig. 9 took 5~15 person-days. In comparison with the implementation time of 6~12 person-weeks for each COVID-19 pandemic meta-story (Section 5, and Figs. 6, 7, and 8), the MSB API significantly reduces the development costs. The ML case study also enabled us to test and evaluate the MSB API, with several aspects of the MSB API getting improved.

Developing stories for new datasets usually takes several hours, assuming that all necessary feature detection, actions, and story designs are already available within the MSB API library. However, if custom designs or new features need to be implemented, the library will require extensions. Consequently, this may extend the development timeline, taking approximately 5~15 person-days to complete the development of the additional functionalities, integrate them with the existing API, and deliver the story.

7.4. Independent Expert Evaluation

In addition to the two formal evaluation meetings, we collected qualitative feedback from four ML experts to further assess the usefulness of MSB in ML workflows. The questions asked included their impressions of computer-generated storytelling animation; the merits and demerits of such animation in comparison to static visualization; automatic generation of animations using APIs; and potential usefulness of such visualization to ML team leaders, researchers, and developers. The individual evaluators' comments are provided in Appendix F.

The experts found that the computer-generated storytelling animation is useful for keeping track of and understanding long-running ML workflows, as the animated stories can replay logged events in temporal order, draw attention to important events, and reduce the effort required to interpret a large number of logged events. Several experts confirmed our observation made during formal meetings that the multivariate parallel coordinate plot was easy to follow. One expert related it to grid search over hyperparameters, enabling the identification of the best combination as well as the observation of suboptimal alternatives in a single view.

Experts' feedback emphasized that storytelling and static visualization serve complementary purposes. Animation was seen as beneficial for guided exploration, understanding temporal evolution, and communicating the progression of ML experiment. However, experts also noted that animation can be slower to consume and can make it more difficult to compare distant events or inspect an individual component. In that they indicated that static visualizations remain better suited for these specific tasks.

Automatic generation of stories was viewed as an important advantage of the approach. Experts consistently indicated that they would not routinely create an animation manually for every experiment, whereas an API-based mechanism could make storytelling scalable to a large number of experiments and workflows encountered in practice. They also identified the ability of MSB to generate stories for previously unseen data as an important benefit of the approach.

The experts offered a number of suggestions for future development, importantly more ML-specific feature detection, and integration of MSB with commonly used ML logging and monitoring tools. These suggestions show that the experts have looked ahead to the potential applications of the MSB API.

8. Discussions and Conclusion

In this work, we presented a new method for creating storytelling visualizations, which is supported by the MSB API (open-source software) for supporting workflows of meta-authoring with feature-action design patterns. We have demonstrated the feasibility of this new method through three storyboards developed in the COVID-19 visualization context, followed by three storyboards describing ML workflows. Our work addresses the challenge of creating generic storyboards that can be applied to different yet similar, often partially unknown data streams and can respond to different (data) features automatically using different (visualization) actions. As summarized in Sections 5 and 7, the method has been welcomed by

different stakeholders, who have already looked ahead to the potential applications in the future as well as more advanced technologies that can provide system-level support to meta-authoring workflows.

In addition to the novel method for authoring meta-storyboard for dealing with numerous dynamic data streams, we provide an algorithmic solution as well as a system solution to alleviate the serious scalability issue if one were to implement different software for different applications. While the development of the MSB API (Section 6) was built on the software written for the COVID-19 application (Sections 5 and 4), and it was used to implement meta-storyboards in an ML application (Section 7), which in turn, was used to test and evaluate the MSB API.

The MSB concept and technique enable design patterns in storytelling visualization to be reusable across different data streams in the same context. The MSB API enables the MSB concept and algorithmic solution to be reusable across different applications. Naturally, features that are not explicitly defined in our API are not automatically detected. Our modular API design facilitates the straightforward addition of new feature-action patterns, allowing easy extension of the API to address the needs for adding new data characteristics in complex application domains. Therefore, an important aspect of future work is to introduce more features and actions as well as more plot types, in order to support more design patterns and more applications. From a scalability perspective, the MSB pipeline scales linearly in relation to the number of data streams. Because our feature-action patterns operate locally on individual data streams rather than relying on global system updates, the system can efficiently process increasing numbers of independent streams in real-time, effectively decoupling story logic from data volume. While interaction defined in MSB storyboards provides different viewers with the flexibility of personalization, each specific story in the form of sequential, narrative-driven animation may not align with the preference of every viewer. Addressing diverse needs of different viewers remains an important avenue of future research. Finally, our evaluation is preliminary, which relied largely on collaborative feedback and varying case study conditions. Future work will allow us to engage more evaluators and result in more case studies, facilitating the evaluation of general applicability and efficacy. Building on this work, we hope to continue to improve techniques for storytelling visualization, broadening application contexts and extending the MSB API.

Acknowledgments

The authors would like to express their gratitude to the SCRC management team for its leadership, all SCRC members for their selfless efforts, and the UK Science and Technology Facilities Council (STFC) for providing and managing the RAMPVIS hardware platform. In particular, we thank all other VIS volunteers for their contributions to various RAMPVIS activities since May 2020. We would also like to thank four ML experts, including Dr. Afrah Farea, Jiachuan Peng, Chenhao Xue, and Zelin Ye, University of Oxford, who evaluated the MSB application to ML workflows. This work was supported in part by the UKRI/EPSCRC grants EP/V054236/1 and EP/X029557/1.

Ethics Statement

N/A

References

- [ARL*16] AMINI F., RICHE N. H., LEE B., MONROY-HERNANDEZ A., IRANI P.: Authoring data-driven videos with dataclips. *IEEE Trans. Visual Comput. Graphics* 23, 1 (2016), 501–510. doi:10.1109/TVCG.2016.2598647. 3
- [BEDF15] BOY J., EVEILLARD L., DETIENNE F., FEKETE J.-D.: Suggested interactivity: Seeking perceived affordances for information visualization. *IEEE Trans. Visual Comput. Graphics* 22, 1 (2015), 639–648. doi:10.1109/TVCG.2015.2467201. 3
- [BFAR*23] BACH B., FREEMAN E., ABDUL-RAHMAN A., TURKAY C., KHAN S., FAN Y., CHEN M.: Dashboard design patterns. *IEEE Trans. Visual Comput. Graphics* 23, 1 (2023), 342–352. doi:10.1109/TVCG.2022.3209448. 12
- [BLB*17] BREHMER M., LEE B., BACH B., RICHE N. H., MUNZNER T.: Timelines Revisited: A Design Space and Considerations for Expressive Storytelling. *IEEE Trans. Visual Comput. Graphics* 23, 9 (2017), 2151–2164. doi:10.1109/TVCG.2016.2614803. 3
- [BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D³ Data-Driven Documents. *IEEE Trans. Visual Comput. Graphics* 17, 12 (2011), 2301–2309. doi:10.1109/TVCG.2011.185. 10
- [BSB*18] BACH B., STEFANER M., BOY J., DRUCKER S., BARTRAM L., WOOD J., CIUCCARELLI P., ENGELHARDT Y., KOEPPE U., TVERSKY B.: Narrative design patterns for data-driven storytelling. In *Data-driven storytelling*. AK Peters/CRC Press, 2018, pp. 107–133. 2, 3
- [BWF*18] BACH B., WANG Z., FARINELLA M., MURRAY-RUST D., HENRY RICHE N.: Design Patterns for Data Comics. In *Procs. of the ACM CHI* (New York, NY, USA, 2018), CHI '18, ACM, p. 1–12. doi:10.1145/3173574.3173612. 3
- [CARA*22] CHEN M., ABDUL-RAHMAN A., ARCHAMBAULT D., DYKES J., SLINGSBY A., RITSOS P. D., TORSNEY-WEIR T., TURKAY C., BACH B., BRETT A., FANG H., JIANU R., KHAN S., LARAMEE R. S., MATTHEWS L., NGUYEN P. H., REEVE R., ROBERTS J. C., VIDAL F., WANG Q., WOOD J., XU K.: RAMPVIS: Answering the challenges of building visualisation capabilities for large-scale emergency responses. *Epidemics* 39 (2022), 100569. doi:10.1016/j.epidem.2022.100569. 2, 20
- [Cen22] CENTER FOR SYSTEMS SCIENCE AND ENGINEERING (CSSE) AT JOHNS HOPKINS UNIVERSITY (JHU): *COVID-19 Dashboard*. [Online]. Available <https://coronavirus.jhu.edu/map.html>, 2022. Accessed: 25/03/2022. 3
- [Cox11] COX A.: How editing and design changes news graphics. In *IEEE Conference on Visual Analytics Science and Technology (VAST)* (2011), pp. xiii–xiii. doi:10.1109/VAST.2011.6102432. 3
- [DARA*22] DYKES J., ABDUL-RAHMAN A., ARCHAMBAULT D., BACH B., BORGO R., CHEN M., ENRIGHT J., FANG H., FIRAT E. E., FREEMAN E., GÖNEN T., HARRIS C., JIANU R., JOHN N. W., KHAN S., LAHIF A., LARAMEE R. S., MATTHEWS L., MOHR S., NGUYEN P. H., RAHAT A. A. M., REEVE R., RITSOS P. D., ROBERTS J. C., SLINGSBY A., SWALLOW B., TORSNEY-WEIR T., TURKAY C., TURNER R., VIDAL F. P., WANG Q., WOOD J., XU K.: Visualization for epidemiological modelling: Challenges, solutions, reflections & recommendations. *Philosophical Transactions of the Royal Society A* 380 (2022), 2233. doi:10.1098/rsta.2021.0299. 2
- [DHPP17] DEMIRALP Ç., HAAS P. J., PARTHASARATHY S., PEDAPATI T.: Foresight. In *Procs of the VLDB Endowment* (Aug. 2017), vol. 10, VLDB Endowment, pp. 1937–1940. URL: <https://doi.org/10.14778/3137765.3137813>, doi:10.14778/3137765.3137813. 4
- [FBM15] FULDA J., BREHMER M., MUNZNER T.: TimeLineCura-tor: Interactive authoring of visual timelines from unstructured text.

- IEEE Trans. Visual Comput. Graphics* 22, 1 (2015), 300–309. doi:10.1109/TVCG.2015.2467531. 3
- [GAW*11] GLEICHER M., ALBERS D., WALKER R., JUSUFI I., HANSEN C. D., ROBERTS J. C.: Visual comparison for information visualization. *Information Visualization* 10, 4 (2011), 289–309. doi:10.1177/1473871611416549. 8
- [GP01] GERSHON N., PAGE W.: What Storytelling Can Do for Information Visualization. *Communication of the ACM* 44, 8 (aug 2001), 31–37. doi:10.1145/381641.381653. 1, 3
- [HBJT20] HEY T., BUTLER K., JACKSON S., THIYAGALINGAM J.: Machine learning and big scientific data. *Philosophical Transactions of the Royal Society A* 378 (2020), 20190054. 11
- [HDR*13] HULLMAN J., DRUCKER S., RICHE N. H., LEE B., FISHER D., ADAR E.: A deeper understanding of sequence in narrative visualization. *IEEE Trans. Visual Comput. Graphics* 19, 12 (2013), 2406–2415. doi:10.1109/TVCG.2013.119. 3
- [KM13] KOSARA R., MACKINLAY J.: Storytelling: The next step for visualization. *Computer* 46, 5 (2013), 44–50. doi:10.1109/MC.2013.36. 1
- [KNA*22] KHAN S., NGUYEN P., ABDUL-RAHMAN A., FREEMAN E., TURKAY C., CHEN M.: Rapid Development of a Data Visualization Service in an Emergency Response. *IEEE Transactions on Services Computing* 15, 3 (2022), 1251 – 1264. doi:10.1109/TSC.2022.3164146. 11, 12, 20
- [KNAR*22] KHAN S., NGUYEN P. H., ABDUL-RAHMAN A., BACH B., CHEN M., FREEMAN E., TURKAY C.: Propagating Visual Designs to Numerous Plots and Dashboards. *IEEE Trans. Visual Comput. Graphics* 28, 1 (2022), 86–95. doi:10.1109/TVCG.2021.3114828. 12
- [KWHH17] KIM Y., WONGSUPHASAWAT K., HULLMAN J., HEER J.: GraphScape: A Model for Automated Reasoning about Visualization Similarity and Sequencing. In *Proc. of the ACM CHI* (New York, NY, USA, May 2017), ACM, pp. 2628–2638. URL: <https://doi.org/10.1145/3025453.3025866>, doi:10.1145/3025453.3025866. 4
- [LES20] LAW P., ENDERT A., STASKO J. T.: Characterizing automated data insights. In *Proc. of the IEEE Visualization Conference - Short Papers* (2020), IEEE, pp. 171–175. URL: <https://doi.org/gk8brf>, doi:10.1109/VIS47514.2020.00041. 3
- [LKS13] LEE B., KAZI R. H., SMITH G.: SketchStory: Telling More Engaging Stories with Data through Freeform Sketching. *IEEE Trans. Visual Comput. Graphics* 19, 12 (2013), 2416–2425. doi:10.1109/TVCG.2013.191. 3
- [LRIC15] LEE B., RICHE N. H., ISENBERG P., CARPENDALE S.: More Than Telling a Story: Transforming Data into Visually Shared Stories. *IEEE Comput. Graphics Appl.* 35, 5 (2015), 84–90. doi:10.1109/MCG.2015.99. 1, 3
- [MBS24] MOERTH E., BRUCKNER S., SMIT N. N.: Scrollyvis: Interactive visual authoring of guided dynamic narratives for scientific scrollytelling. *IEEE Trans. Visual Comput. Graphics* 30, 1 (2024). 3
- [MRL*17] MCKENNA S., RICHE N. H., LEE B., BOY J., MEYER M.: Visual Narrative Flow: Exploring Factors Shaping Data Visualization Story Reading Experiences. *Comput. Graphics Forum* 36, 3 (2017), 377–387. doi:10.1111/cgfm.13195. 3
- [Nor18] NORTHWESTERN UNIVERSITY KNIGHT LAB: *Timeline.js*. [Online]. Available <https://timeline.knightlab.com/>, 2018. Accessed: 30/03/2022. 3
- [OBCT24] OFFENWANGER A., BREHMER M., CHEVALIER F., TSANDILAS T.: Timesplines: Sketch-based authoring of flexible and idiosyncratic timelines. *IEEE Trans. Visual Comput. Graphics* 30, 1 (2024). 1, 3
- [PLC*11] PARRY M. L., LEGG P. A., CHUNG D. H., GRIFFITHS I. W., CHEN M.: Hierarchical Event Selection for Video Storyboards with a Case Study on Snooker Video Visualization. *IEEE Trans. Visual Comput. Graphics* 17, 12 (2011), 1747–1756. doi:10.1109/TVCG.2011.208. 4, 5
- [RBSN22] ROBERTS J. C., BUTCHER P., SHERLOCK A., NASON S.: Explanatory Journeys: Visualising to Understand and Explain Administrative Justice Paths of Redress. *IEEE Trans. Visual Comput. Graphics* 28, 1 (2022), 518–528. doi:10.1109/TVCG.2021.3114818. 3
- [RG13] ROSENBERG D., GRAFTON A.: *Cartographies of time: A history of the timeline*. Princeton Architectural Press, 2013. 3
- [RHR16] ROBERTS J. C., HEADLEAND C., RITSOS P. D.: Sketching Designs Using the Five Design-Sheet Methodology. *IEEE Trans. Visual Comput. Graphics* 22, 1 (2016), 419–428. doi:10.1109/TVCG.2015.2467271. 9
- [SH10] SEGEL E., HEER J.: Narrative visualization: Telling stories with data. *IEEE Trans. Visual Comput. Graphics* 16, 6 (2010), 1139–1148. doi:10.1109/TVCG.2010.179. 3
- [SH14] SATYANARAYAN A., HEER J.: Authoring narrative visualizations with Ellipsis. *Comput. Graphics Forum* 33, 3 (2014), 361–370. doi:<https://doi.org/10.1111/cgfm.12392>. 3
- [SKH*22] SHIN M., KIM J., HAN Y., XIE L., WHITELAW M., KWON B. C., KO S., ELMQVIST N.: Roslingifier: Semi-automated storytelling for animated scatterplots. *IEEE Trans. Visual Comput. Graphics* (2022), 1–1. doi:10.1109/TVCG.2022.3146329. 2
- [SWT*20] SHU X., WU A., TANG J., BACH B., WU Y., QU H.: What makes a Data-GIF Understandable? *IEEE Trans. Visual Comput. Graphics* 27, 2 (2020), 1492–1502. doi:10.1109/TVCG.2020.3030396. 3
- [SXS*21] SHI D., XU X., SUN F., SHI Y., CAO N.: Calliope: Automatic Visual Data Story Generation from a Spreadsheet. *IEEE Trans. Visual Comput. Graphics* 27, 2 (2021), 453–463. URL: <https://doi.org/10.1109/TVCG.2020.3030403>, doi:10.1109/TVCG.2020.3030403. 1, 3, 4
- [SyzZW24] SHEN L., YIZHI ZHANG, ZHANG H., WANG Y.: Data player: Automatic generation of data videos with narration-animation interplay. *IEEE Trans. Visual Comput. Graphics* 30, 1 (2024). 3
- [TLW*21] TANG T., LI R., WU X., LIU S., KNITTEL J., KOCH S., ERTL T., YU L., REN P., WU Y.: PlotThread: Creating Expressive Storyline Visualizations using Reinforcement Learning. *IEEE Trans. Visual Comput. Graphics* 27, 2 (2021), 294–303. doi:10.1109/TVCG.2020.3030467. 3
- [TRB*18] TONG C., ROBERTS R. C., BORGIO R., WALTON S. P., LARAMEE R. S., WEGBA K., LU A., WANG Y., QU H., LUO Q., MA X.: Storytelling and Visualization: An Extended Survey. *Information* 9 (2018), 65. doi:10.3390/info9030065. 1, 3
- [TSFH22] THIYAGALINGAM J., SHANKAR M., FOX G., HEY T.: Scientific machine learning benchmarks. *Nature Reviews Physics* 4 (2022), 413–420. 11
- [UK 22] UK HEALTH SECURITY AGENCY: *GOV.UK Coronavirus (COVID-19) in the UK*. [Online]. Available <https://coronavirus.data.gov.uk/>, 2022. Accessed: 25/03/2022. 3
- [WaCP*15] WALKER R., AP CENYDD L., POP S., MILES H. C., HUGHES C. J., TEAHAN W. J., ROBERTS J. C.: Storyboarding for visual analytics. *Information Visualization* 14, 1 (2015), 27–50. doi:10.1177/1473871613487089. 3
- [WFRR20] WILLIAMS R. L., FARMER D., ROBERTS J. C., RITSOS P. D.: Immersive visualisation of COVID-19 UK travel and US happiness data. In *Posters of the IEEE Conference on Visualization* (Oct. 2020). 3
- [WGH*24] WU G., GUO S., HOFFSWELL J., CHAN G. Y.-Y., ROSSI R., KOH E.: Socrates: Data story generation via adaptive machine-guided elicitation of user feedback. *IEEE Trans. Visual Comput. Graphics* 30, 1 (2024). 3
- [WLF*19] WANG Q., LI Z., FU S., CUI W., QU H.: Narvis: Authoring Narrative Slideshows for Introducing Data Visualization Designs. *IEEE Trans. Visual Comput. Graphics* 25, 1 (2019), 779–788. doi:10.1109/TVCG.2018.2865232. 3

- [WRC*21] WANG Z., ROMAT H., CHEVALIER F., RICHE N. H., MURRAY-RUST D., BACH B.: Interactive Data Comics. *IEEE Trans. Visual Comput. Graphics* 28, 1 (2021), 944–954. doi:10.1109/TVCG.2021.3114849. 3
- [WSZ*20] WANG Y., SUN Z., ZHANG H., CUI W., XU K., MA X., ZHANG D.: DataShot: Automatic Generation of Fact Sheets from Tabular Data. *IEEE Trans. Visual Comput. Graphics* 26, 1 (2020), 895–905. URL: <https://doi.org/10.1109/TVCG.2019.2934398>, doi:10.1109/TVCG.2019.2934398. 2, 3, 4
- [WWF*19] WANG Z., WANG S., FARINELLA M., MURRAY-RUST D., HENRY RICHE N., BACH B.: Comparing effectiveness and engagement of data comics and infographics. In *Procs. of the ACM CHI* (New York, NY, USA, 2019), ACM, p. 1–12. doi:10.1145/3290605.3300483. 3
- [WWZ*21] WU A., WANG Y., ZHOU M., HE X., ZHANG H., QU H., ZHANG D.: MultiVision: Designing Analytical Dashboards with Deep Learning Based Recommendation. *IEEE Trans. Visual Comput. Graphics* 28, 1 (2021), 162–172. doi:10.1109/TVCG.2021.3114826. 4
- [YC18] YOUSUF B., CONLAN O.: Supporting Student Engagement Through Explorable Visual Narratives. *IEEE Transactions on Learning Technologies* 11, 3 (2018), 307–320. doi:10.1109/TLT.2017.2722416. 3
- [YLRC10] YU L., LU A., RIBARSKY W., CHEN W.: Automatic Animation for Time-Varying Data Visualization. *Comput. Graphics Forum* 29, 7 (2010), 2271–2280. doi:10.1111/j.1467-8659.2010.01816.x. 4
- [ZOM19] ZHI Q., OTTLEY A., METOYER R.: Linking and layout: Exploring the integration of text and visualization in storytelling. *Comput. Graphics Forum* 38, 3 (2019), 675–685. doi:10.1111/cgfm.13719. 3

Appendices

Meta-Storyboard Using Feature-Action Patterns: Concept, Techniques, and Case Studies

S. Khan, S. Jones, B. Bach, J. Cha, M. Chen, J. Meikle, J. C. Roberts, J. Thiagalingam, J. Wood, and P. D. Ritsos

Appendix A: Introduction to Appendices

This set of four appendices provides further details about the technical aspects of this work and the evaluation of the meta-storyboard (MSB) approach. These include:

- Appendix B: This appendix provides the essential information for a new MSB developer to start using the MSB API. It includes instructions on installing the MSB API and a template program that can be modified to implement more complex MSBs.
- Appendix C: This appendix provides an overview of the general structure of a feature-action table, which is the key component that an MSB author must specify (typically in plain texts) and an MSB developer must translate the specification into a feature-action table that the MSB API can understand. In the context of the system architecture shown in Fig. 4, the feature-action specification and feature-action table are illustrated at the top-right corner.
- Appendix D: This appendix provides a summary description of the MSB API, including its class structure and a sequence diagram that implements the algorithmic workflow presented in the lower half of 4.
- Appendix E: It provides the original texts collected in the evaluation process. They are summarized in Section 5.
- Appendix F: It presents the original texts collected from ML experts during the evaluation process.

Note that the source code of the Meta-Storyboard (MSB) API, comprehensive technical documentation, and examples can be found at:

- source code: <https://github.com/saifulkhan/meta-storyboard>,
- published npm package: <https://www.npmjs.com/package/meta-storyboard>,
- example created using React.js: <https://github.com/saifulkhan/meta-storyboard-examples>, and
- live examples: <https://saifulkhan.github.io/meta-storyboard-examples>.

In addition, there are further supplementary materials:

- a 4-minute video for showing six demo video clips <https://youtu.be/IT1Xl-VycJs>,
- a 3-minute video <https://vis4pandemres.github.io/materials/1014-file-i36.mp4> as overview of the development of the meta-authoring concept in the context of pandemic responses. This video was shown as a video-bulletin in the VIS 2023 Workshop on Visualization for Pandemic and Emergency Responses (<https://vis4pandemres.github.io>).

Appendix B: Meta-Storyboard: Getting Started

Workflow of a Meta-Storyboard Program

A meta-storyboard (MSB) program typically consists of the following four steps:

STEP 1: Data Selection and Preprocessing

- Based on a viewer's input, the program selects or filters the data. For example, with COVID-19 data, a viewer may select a region, and the program assigns the time series for the selected region to TS1, which is the ID used in the feature-action table.
- For a more complicated storyboard, this step may include processes for creating derived data, e.g., computing the difference between two time series selected by a viewer.

STEP 2: Processing Feature-Action Table

- For each row in the feature-action table, the program
 - detects the specified feature,
 - registers the specified action against the feature, and
 - assigns the specified rank to the action.
- In addition to the properties for features and actions, there is a data buffer shared by the feature search or detection function `search()` and the action registration or creation function `create()`. The buffer allows `create()` to access information such as the current data point, the previous data point before the feature search (excluding the dummy feature CURRENT), and so on.
- When there are multiple time series specified in the feature-action table, this step processes all time series concurrently.
- If there are multiple time series, the program creates an integrated time series with combined ranks and all registered actions. The ranks are combined using a Gaussian mixture model.
- The program determines the number of time segments according to the length of the animation to be created. The default number is 1 segment, i.e., no segmentation is required.
- The program applies a segmentation algorithm to divide the integrated time series according to the specified number of segments.
- For each time series, the program selects the important feature-action pairs (with higher ranks) according to the time allowed for the animation of this segment.

STEP 3: Initialize Animation Controller and Visualization

- Animation control components manage play, pause, timing, and user interactions. The management of animation states is accomplished through state and context mechanisms, which include the logic needed to synchronize animation states, current frame, and animation parameters across various components (e.g., user interface, actions, and visualization).
- We centralized animation state to ensure consistent updates and smooth transitions during both user interactions and automated playback.
- Visualization components utilize the animation state to trigger transitions and interpolate visual properties in response to changes in that state.

STEP 4: Create Animated Visualization

- The program displays the selected actions as an animated sequence.

Installing the MSB API (Library)

The MSB API is released as a standard JavaScript/TypeScript library that can be accessed via NPM at <https://www.npmjs.com/package/meta-storyboard>. To install the MSB library in a Node.js development environment, one may execute the following command:

```
npm install meta-storyboard
```

Subsequently, one can import the library into a project using the following command:

```
import * as msb from 'meta-storyboard';
```

The following simple MSB program is a “hello” program for a very simple story. It can be modified gradually to develop more complex stories. Hence, it is a template program, where there are four essential major steps.

```
1. Select data and feature action tables
const data = <time series data>
const numericalFATable = <feature-action table>

2. Process the feature-action table
const timelineActions =
new msb.FeatureActionFactory()
    .setProps(<properties>)
    .setData(data)
    .setNumericalFeatures(numericalFATable)
    .create();

3. Initialize visualization components
const plot = useRef(new msb.LinePlot()).current;
const [controller, isPlaying] =
    useControllerWithState(msb.
        PlayPauseController, [plot]);

4. Configure and render the visualization
plot
    .setData(data)
    .setName(<name>)
    .setPlotProps(<plot properties>)
    .setCanvas(<svg element>)
    .setActions(timelineActions)
    .animate();

onClick={controller.togglePlayPause};
```

For applications requiring the integration of both numerical and categorical time series with automated segmentation using Gaussian mixture models, the `msb.FeatureActionFactory` provides two additional methods `setCategoricalFeatures` and `segment`. The step 2 of the above template can be modified as:

```
2. Process the feature-action table
const timelineActions =
new msb.FeatureActionFactory()
    .setProps(<>)
    .setData(data)
```

```
.setNumericalFeatures(numericalFeatures)
.setCategoricalFeatures(<events>, <FA table>)
.segment(<number of segments>, <method>)
.create()
```

The template program generates storytelling visualization using React-based web development framework, together with the built-in `useRef` hook and a custom `useControllerWithState` hook for animation control (communication and state management between play/pause button and animated plots). Nevertheless, the core MSB API is framework-agnostic. The MSB library can be used with other frameworks such as Angular, Vue, or vanilla JavaScript/HTML implementations (with minor adaptations to the animation state management).

Example Stories

The GitHub site for the MSB API also includes four different meta-storyboards, each including instructions from the meta-authors, application data, target users, feature-action tables, and visual representations. As shown in Table 2, the first two stories focused on telling stories about COVID-19 data. Three meta-authors collaborated to create the storyboards. One developer initially wrote programs to prototype these meta-storyboards, while a second developer adapted and ported the programs to the RAMPVIS server. Later, the second developer re-implemented them after the MSB API was developed.

The other two stories were authored and implemented by the second developer after he joined a machine learning (ML) team. Both of them were implemented using the MSB API directly.

Creating a new story can easily be accomplished by adapting an existing MSB program or modifying the template given in the previous section. The best starting point for developing a story is to read the example stories in Table 2. These examples are also discussed in our case studies in the main body of the paper.

For further details, please refer to the README file at the GitHub site for the MSB library, <https://github.com/saifulkhan/meta-storyboard/blob/main/README.md>.

Appendix C: Feature Action Tables

As shown in Fig. 2 (the 2nd box) and Fig. 4 (top-right), a meta-author develops a specification for a set of feature-action patterns. This specification can be documented in various formats, such as Word files, spreadsheets, or even handwritten and hand-drawn notes. The meta-author then discusses the specification with a developer, who converts it into a feature-action table in JSON format, which is easily readable by both humans and the meta-storyboard program.

Each feature-action table has the following columns or attributes:

1. **Feature:** `<name>` of a feature extraction function in the Meta-Storyboard API (MSB Features).
2. **Feature Properties:** a list of properties for the feature concerned in the form of `<property name>:<property value>`, `<property name>:<property value>`,

Table 2: Four example stories available at the GitHub site <https://github.com/saifulkhan/meta-storyboard-examples/tree/main/react/src/pages/example>.

(a)	story-covid19.tsx
	An MSB for telling a story about COVID-19 data at one location that is selected by a viewer from many optional locations. The MSB involves features defined on a single time series.
(b)	story-covid19-gaussian.tsx
	Similar to (a), include additional segmentation using a Gaussian mixture model.
(c)	story-ml-pcp.tsx
	An MSB uses a parallel-coordinate plot for telling a story about ML testing results. In an ML workflow, an ML model is developed through many iterations. In each iteration, testing generates multivariate data. The MSB enable automatic visualization of testing data obtained in the latest and past iterations.
(d)	story-ml-mirrorred-bar.tsx
	An MSB uses line plot and a mirrored bar chart to tell stories about testing results obtained in the latest and past iterations.

3. **Rank:** a rank value $\in [1, 10]$ assigned to the action for the detected feature.
4. **Action:** `<name>` of an action function in the Meta-Storyboard API (MSB Actions).
5. **Action Properties:** a list of properties for the action concerned in the form of `<property name>:<property value>`, `<property name>:<property value>`, ...

A meta-authoring storyboard may need to deal with multiple time series, e.g., a categorical time series for dates and events, two numerical time series for the number of cases in two different regions, and the “difference time series” derived from the numerical time series. Features can be defined for each time series independently. The current version of the software assumes that complex features for characterizing interactions among two or more time series are detected in two ways: (a) through a derived time series resulting from a subroutine that processes two or more time series, such as a difference function, (b) through the Gaussian mixture process following the feature extraction step.

Meta-Storyboard Features in the MSB API

Conceptually, features are defined as enumerators and are implemented using functions in the MSB API. Developers can include undefined feature names or property names in a feature-action table when communicating with meta-authors, with the understanding that they will later add these features and/or properties to the MSB API.

In order to implement and manage similar features in an object-oriented manner, we group similar feature functions using a high-

level feature construct. Table 3 presents a list of high-level feature constructs. Each construct is defined in conjunction with a list of properties, such as (`lte:1000`, `gte:2000`) for a value range [1000, 2000]. This allows high-level constructs to be decomposed into low-level constructs. For example, the PEAK feature has low-level constructs for detecting, e.g., the k -th peak or a peak with additional slope criteria. For each construct, there is a predefined feature search function, like `search()`, which analyzes the given time series according to the feature properties specified in the feature-action table.

Table 3: The main features implemented in the MSB API for specifying time series features.

Feature Name	Description
<i>numerical features</i>	
FIRST	encountering the first data point
CURRENT	setting action location at the current data point
LAST	encountering the last data point
MIN	encountering the MIN value
MAX	encountering the MAX value
PEAK	encountering a peak (not necessarily MAX)
RISE	encountering a rising segment
FALL	encountering a falling segment
SLOPE	encountering a segment with specified slope range
<i>categorical features</i>	
EVENT	encountering an event with a specific event label

Meta-Storyboard Actions in the MSB API

Similar to features, actions are defined as enumerators and are implemented using functions in the MSB API, and the function names and their property names are used as predefined constants in feature-action tables. Similar action functions are also grouped together using a high-level feature construct. Table 4 shows a list of high-level action constructs. Action properties facilitate the decomposition of a high-level construct into low-level constructs.

When a feature is detected, the corresponding action is not activated immediately in order for all feature-action patterns to be further processed (e.g., Gaussian mixture, segmentation, and rank-based event selection). Hence, when a feature is detected, a function `create()` creates or registers an action at the data point where the feature is encountered.

Example Feature-Action Tables

We present two example feature-action tables, Tables 5 and 6. They are for illustration, and they are simpler than the meta-storyboards that were developed in our case studies. Fig. 10 shows a screenshot of the user interface for the feature-action table editor. This feature is currently being developed to replace the existing method of defining the feature-action table using JSON.

Appendix D: Meta-Storyboard (MSB) API

In this appendix, we provide the high-level system documentation of the Meta-Storyboard (MSB) API. The development of the soft-

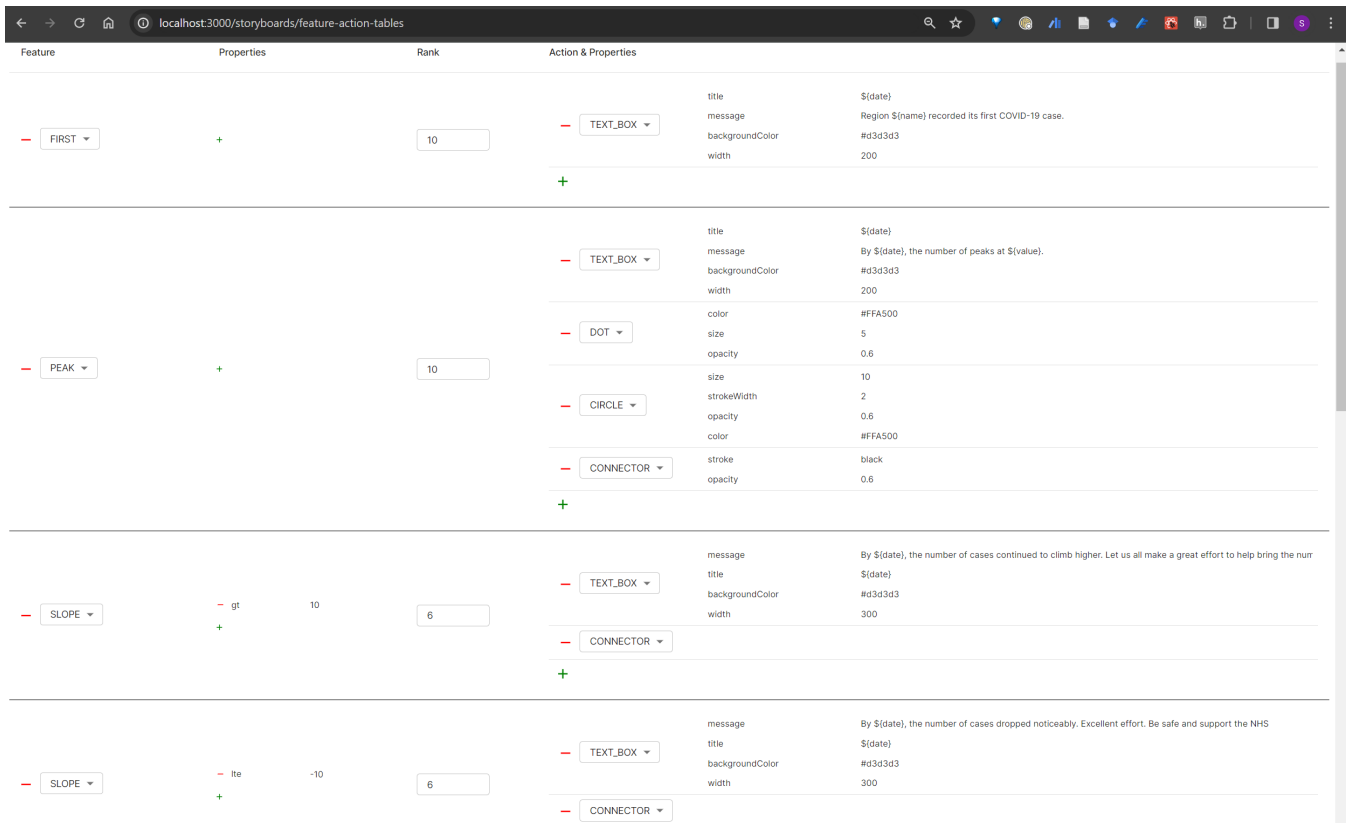


Figure 10: Screenshot of the user interface for feature action table 5 developed for writing meta-stories using feature action design pattern. Unspecified properties use default values.

Table 4: The main actions implemented in the MSB API for specifying visualization actions.

Action Name	Description
DOT	highlight or dehighlight a graph node
CIRCLE	draw or remove a highlighting circle
TEXT_BOX	display a text string in a predefined message box
CONNECTOR	highlight or dehighlight a graph edge (connector)

ware for MSB started in the summer of 2021 as part of the RAM-PVIS project [CARA*22]. The first version of the software was implemented using Observable. In January, the software was re-implemented as part of the RAMPVIS infrastructure [KNA*22]. While these early versions feature an API architecture, they were not well documented due to the need to focus our efforts on other, more urgent tasks during the pandemic response. The API architecture was redesigned after STFC took over the software in the summer of 2023. Since then, the API has evolved gradually, and documentation has become part of the software engineering process. The API has recently been released as a standard JavaScript library on NPM at <https://www.npmjs.com/package/meta-storyboard>.

Naming Conventions

Table 7 shows the standard naming conventions used within the MSB API. These conventions cover MSB actions, MSB features, and their properties. Additionally, it outlines the naming styles for classes, methods, constants, and other coding practices.

Feature Action Table Data Structures

Feature action tables can be written in CSV or JSON files. In our latest version of the software, we implemented user interfaces for meta-story authors to create and update feature action tables. The data structures of the tables are shown below in JSON style format. For more details, please browse the source code.

Numerical Feature Action:

```
{
  "feature": "<feature_name>",
  "properties": {
    "...": "..."
  },
  "rank": "...",
  "actions": [
    {
      "action": "...",
```

Table 5: An example of a feature action table used to create a single-region meta-storyboard with COVID-19 data. The visual results are similar to the screens shown in Figures 6 and 8. Empty cells are interpreted as default properties or null values.

Feature	Feature Properties	Rank	Action	Action Properties	Comment
FIRST		10	TEXT_BOX	message: Region \${name} recorded its first COVID-19 case.	
PEAK		10	TEXT_BOX	message: By \${date}, the number of peaks at \${value}.	
			DOT		default properties
			CIRCLE		default properties
			CONNECTOR		default properties
SLOPE	gte:10	6	TEXT_BOX	message:By \${date}, the number of cases continued to climb higher. Let us all make a great effort to help bring the number down. Be safe and support the NHS.	case A
SLOPE	lte:-10	6	TEXT_BOX	message:By \${date}, the number of cases dropped noticeably. Excellent effort. Be safe and support the NHS.	case B
SLOPE	gt:-10, lt:10	3	TEXT_BOX	message:By \${date}, the number of cases remained low. We should continue to be vigilant.	case C
RISE	gt:15	5	TEXT_BOX	message:The number of cases grew.	
FALL	lte:-15	5	TEXT_BOX	message:By \${date}, the number of cases came down noticeably. We should continue to be vigilant.	
...	...	...	...	...	...

Table 6: An example feature action table for the ML provenance story as shown in Figure 9(a).

Feature	Feature Properties	Rank	Action	Action Properties	Comment
FIRST		10	DOT	size:5, color:#FFA500, opacity:0.6	
			CIRCLE	size:10, color:#FFA500, opacity:0.6, strokeWidth:3	
			TEXT_BOX	message: A newly-trained model achieved testing accuracy of \${test}% and training accuracy of \${train}%, denoted \${test}% [\${train}%].	
			CONNECTOR		default properties
CURRENT		9	DOT	size:10, color:#FFA500, opacity:0.6	
			CONNECTOR		default properties
MAX		10	DOT	size:5, color:#008000, opacity:0.6	
			CIRCLE	size:10, color:#008000, opacity:0.6, strokeWidth:3	
			TEXT_BOX	message: On \${date}, a model achieved the best testing accuracy \${test}% [\${train}%].	
			CONNECTOR		default properties
			PAUSE	duration:5	

Table 7: Standard names used in the API.

Name	Description
Terms in Code	
Class	CamelCase with the initial letter capitalized, e.g., TextBox
Method	camelCase with the initial letter lower-cased, e.g., setProperties
Constants or enumerators	Uppercase letters with underscores, e.g., TEXT_BOX
Setter methods	Uses set as prefix, e.g., setRank
Getter methods	Uses get as prefix, e.g., getRank
<>	Placeholder for actual value
...	Omitted to save space
Other Terms	
Properties	Parameters of features or actions

```

    "properties": {
      "...": "...",
    }
  },

```

```

    "...",
  ],
  "comment": "...",
}

```

Categorical Feature Action:

```

{
  "feature": "<event_name>",
  "properties": {
    "...": "...",
  },
  "rank": "...",
  "actions": [
    {
      "action": "...",
      "properties": {
        "...": "...",
      }
    },
  ],
  "...",
},
"date": "...",
"description": "...",

```

```
}
```

The examples of feature-action tables can be found at <https://github.com/saifulkhan/meta-storyboard-examples/tree/main/react/src/assets>.

MSB Feature Action Factory: The `FeatureActionFactory` translates the meta-stories defined as feature action tables to MSB feature objects and MSB action objects. To do so, it internally uses the `FeatureFactory` and `ActionFactory` classes, which will be discussed later. Instantiate this class as follows:

```
new FeatureActionFactory(...)
  .setProps(...)
  .setNumericalFeatures(...)
  .setCategoricalFeatures(...)
  .setData(...)
  .segment()
  .create()
```

Further details regarding the factory classes can be found at <https://github.com/saifulkhan/meta-storyboard/tree/main/src/factory>.

MSB Features API

The `Feature` abstract class, in Fig. 11, is the core of MSB features API, encapsulating the numerical and semantic dimensions of time series data via subclasses `NumericalFeature` and `CategoricalFeature`. These base classes are the foundation for concrete classes such as specific feature implementations like `Peak`, `Min`, etc. It offers a systematic method for defining time series data features, allowing developers to create new features by extending these base classes.

MSB Feature Names: The MSB feature names are defined as enumerator, e.g.,

```
enum FeatureName {
  FIRST,
  CURRENT,
  LAST,
  MAX,
  MIN,
  PEAK,
  FALL,
  RAISE,
  SLOPE,
  EVENT,
  ...
}
```

All the TypeScript enumerators and type definitions can be found at <https://github.com/saifulkhan/meta-storyboard/tree/main/src/types>.

Creating MSB Feature: To instantiate a feature, use the constructor of a concrete feature class. Set properties using method chaining. For example, when a feature `PEAK` is detected, it creates a `Peak` object as follows. The default feature properties are used unless defined in the feature-action table. Use the 'getter' methods to retrieve the feature properties.

```
new Peak()
  .setDate(<>)
  .setHeight(<>)
  .setNormWidth(<>)
  .setNormHeight(<>)
  .setRank(<>)
  .setMetric(<>)
  .setStart(<>)
  .setEnd(<>)
  .setDataIndex(<>)
```

The implemented features can be found at <https://github.com/saifulkhan/meta-storyboard/tree/main/src/feature>.

MSB Feature Factory: The `FeatureFactory` class implements a factory design pattern for streamlined feature creation, utilizing search functions to dynamically generate feature instances based on input feature action table and time series data. Instantiate this class as follows:

```
new FeatureFactory()
  .setProps()
  .setData()
  .searchNumericalFeature(<feature name>, <
    properties>, ...)
```

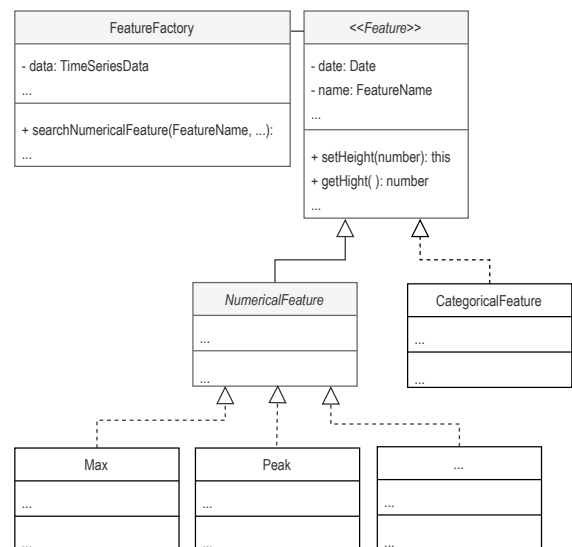


Figure 11: The class diagram of the meta-storyboard feature classes. Please see the source code for all classes, as well as their methods and attributes.

All factory classes are available at <https://github.com/saifulkhan/meta-storyboard/tree/main/src/factory>.

MSB Actions API

The `Action` abstract class, as in Fig. 12, is the abstraction of MSB action API. This base class serves as a blueprint for defining atomic actions, e.g., circles, dots or nodes, etc., represented by concrete classes `Circle` and `Dot` (or 'node' is a JavaScript reserved keyword and thus cannot be defined in program) respectively.

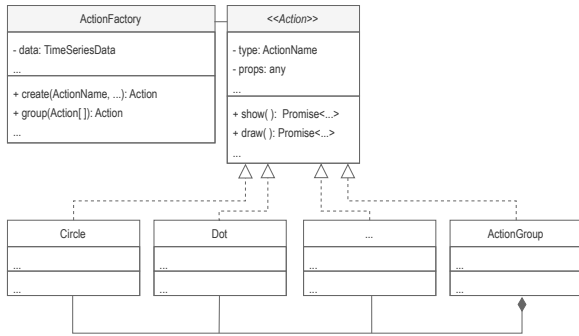


Figure 12: MSB action classes.

MSB Action Names: The MSB action names are defined as enumerators, e.g.,

```
enum ActionName {
    DOT,
    CIRCLE,
    TEXT_BOX,
    CONNECTOR,
    ...
}
```

Creating MSB Actions: Instantiate a `Circle` action object and display it within an SVG canvas. Unless defined in the feature-action table, the default action properties are used.

```
new Circle()
    .setProps(<properties>)
    .setCanvas(<svg>)
    .setCoordinate(<coordinates>)
    .show()
    ...
```

Animated movement for objects, e.g., `TextBox`, can be achieved by:

```
new TextBox()
    .setProps(<properties>)
    .setCanvas(<svg>)
    .setCoordinate(<x, y coordinate>)
    .show()
    .move(<coordinates>, ...)
    ...
```

Group MSB Actions: The `ActionGroup` class employs a composite design pattern to create API for grouping multiple actions representing one feature, as shown in an example below:

```
new ActionGroup()
    .group(<array of actions>)
    .setCanvas(<svg>)
    .setCoordinate(<x, y coordinate>)
    .show()
```

All action classes can be found at <https://github.com/saifulkhan/meta-storyboard/tree/main/src/components/actions>.

MSB Action Factory: The `ActionFactory` uses the factory design pattern to simplify creation of MSB action objects, streamlining action generation from feature action tables.

```
new ActionFactory()
    .create(<name>, <properties>)
```

Feature Search and Gaussian API

The feature search or detection API and the API for calculating the Gaussian distributions of numerical and categorical time series are designed as pure functions.

Feature Detection: The `Search` class implements various feature detection algorithms. The following is an example API for peak detection. Similarly, additional feature detection APIs have also been implemented.

```
Search.searchPeaks(<time series>, ...)
```

Gaussian: The `Gaussian` class implements various Gaussian algorithms. The following is an API for the Gaussian Mixture Model (GMM) to compute combined Gaussians from numerical time series and categorical time series.

```
Gaussian.gmm(<nts>, <cts>)
```

The search and Gaussin classes are available at <https://github.com/saifulkhan/meta-storyboard/tree/main/src/processing>.

Animated Plots API

`Plot` is an abstract class, as in Fig. 13, defines the API of all animated plots. The API includes methods for drawing and animating plots, setting their properties, and animating actions. Concrete plots, such as `LinePlot`, `ParallelCoordinatePlot`, `MirroredBarChart`, etc. define their own drawing logic.

Creating a Plot: Create a line plot and animate actions,

```
new LinePlot()
    .setData(<time series>)
    .setName(<data stream name>)
    .setLineProps(<line properties>)
    .setPlotProps(<plot properties>)
    .setCanvas(<svg>)
    .setActions(<timeline actions>)
    .plot()
    .animate()
```

All implemented plots, such as line plot, mirrored bar chart, and parallel coordinate plot, can be found at <https://github.com/saifulkhan/meta-storyboard/tree/main/src/components/plots>.

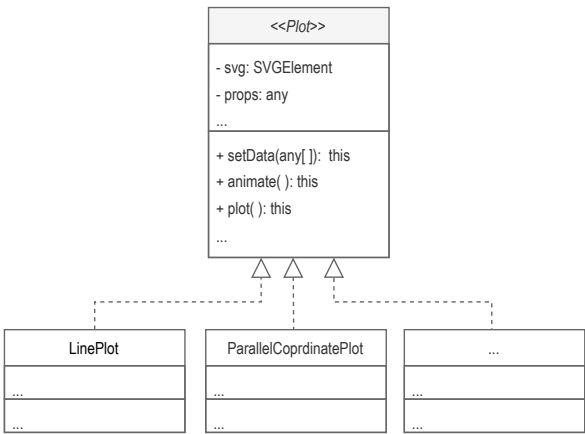


Figure 13: Plot classes.

Detailed Sequence Diagram

The sequence diagram in Fig. 14 shows the steps and API involved in the story creation process, as discussed in appendix B. This diagram shows the flow of a story creation process and the interactions between different API.

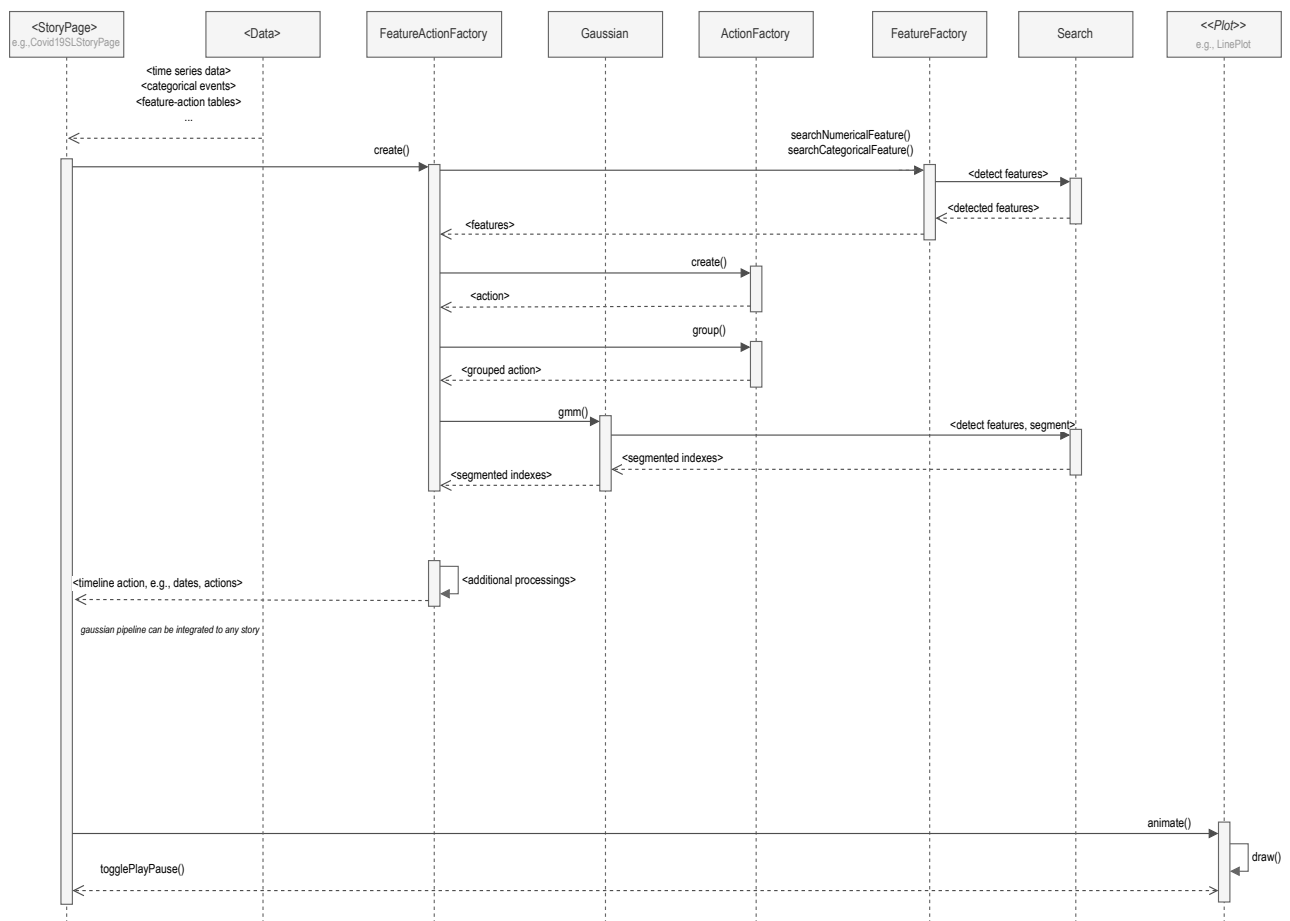


Figure 14: Sequence diagram illustrating the process of creating a story at a high level. This diagram outlines the key classes and methods that are involved in this process.

Appendix E: Evaluation Records – COVID-19Storyboards

This Google doc is for collecting feedback and reflection from different stakeholders of the Storytelling visualization project. We would like to collect feedback, comments, and reflection from three perspectives, namely story design, technical development, and public engagement (including ordinary users) in the following three sections. Please feel free to add any comment, while the questions listed are just prompts. One does not need to answer all questions and one can add additional questions and answers. In this version of the document, the stakeholders' initials have been replaced by [A], [B], etc.

From the Perspective of Story Design

Please feel free to add any comment from the perspective of a story designer. The questions below are just prompts.

Q1: Comments on the challenges in Meta-storytelling. What is the difference between designing a storyboard for a single piece of data and that for multiple pieces of data?

[A]

Designing the meta-stories needs you to think differently. Two things need to be achieved. First, we need to think about the story holistically. What are the important story elements? How does the story progress? What are the key moments of interest? Second, the information needs to be abstracted (generalized) into instructions that can be coded, and recorded. In the case of the covid stories, we needed to think about what drives the main moments. We can extract key moments such as: rise of cases; fall of cases; peak cases; and key moments such as vaccination start, local lockdowns, and so on. But it is more than this we need to encode these into mathematical generalizations. Rise $\pm$ 20% over two weeks; or peak within two weeks.

From a design point of view, the interface needs to enable the user to move between events, and understand the story in a sensible way. Using play, rewind and so on, means that it is a familiar interface. Exposing the timeline (in story 3) also provides an overview of the information, and a hint at where some of these story elements exist.

[B]

Thinking about the stories in this 'meta' way is certainly a challenge, however it forces one to think about the important messages, the progression of the story and how in a sense, get to the point. Thinking about what each timeline depicts and how that translates into key messages, or messages of comparison does force the storyteller to consider whether the underlying data does indeed support the argument that they want to put across. It is similar to answering a questionnaire that has a mechanism to compensate for response acquiescence - it forces you to consider the story in detail.

More importantly though, it makes you consider alternative scenarios in a more computational way, rather than a method that is more creative. In this case you have to think about aspects such as what to do when a feature is not present, or a comparison does not result into a key message - in a way, similar to catching exceptions in programming. In that regard, it is a more technical way to approach what-if questions.

[C]

Forcing authors to think about the underlying data in different regions may help instigate a good practice in creating storytelling visualization. Creating a story based on the observation of the data in one region may lead to overgeneralization in some messages and conclusions. Since storytelling visualization can potentially engage users in the public more than visualization for domain experts, the biased message based on anecdotal evidence (e.g., one region) can be an issue in storytelling visualization as users may not have the same knowledge as domain experts to spot the biases or overgeneralization.

Q2: What does a story designer need to anticipate? To design a storyboard for visualizing a type of time series e.g., the number of cases in a region over a period, how many pieces of time series one may need to see before one can formulate a "generic" and "intelligent" storyboard that should work for many regions?

[A]

The story designer needs to be able to visualize (in their minds eye) the whole story. Often, especially in the case of the covid example, it is possible to explore the story in one location, and then abstract the key elements, and then generalize the story to several locations. In other words, creating a draft storyline (on some specific data) helped to think about how to generalize and abstract the information into the meta-instructions.

[B]

Generalizing the story is definitely a gain from this approach. In some cases the expected data (or resulting features) may not be there, so the story has to progress, or compensate for that absence - after all, absence of data may be data itself (especially in the Covid-19 pandemic). In the scenarios we explored, it is impossible to know for sure whether the pandemic unfolded in a particular way in a particular location.

Granted, there is a general expectation, but that is more along the lines of defining features that are agnostic of what happened, and more related to how one would detect something IF it happened. In both cases, appropriate messaging needs to be included.

Q3: Is identifying features easy or difficult? Is determining actions easy or difficult? Please add some explanation.

[A]

Certainly some of the key features were obvious. Points like peak, rise and fall (by tangent calculation). But not all of the locations were easy. Understanding how sensitive the values should be, or what some of the minor key features should be, was not necessarily straightforward. But like other projects, it is a matter of thinking, considering, talking with others, and trying things out. As a group of designers, it was important for us to discuss the outputs and how they appear, and can be controlled.

One extension could be to offer the user additional control. The reader can then prioritize certain events. They may be interested in local events nces; and through an interface it would be possible to allow a user to select the types of events and sensitivity to only play items that they may specify.

[B]

It is somewhat challenging, but that is the very essence of it - to make you consider the progression of the story in a more 'mathematical' way. There is also an element of built-up. Say, one creates a simple story, with one time series. The selected features can be used for another time series. Then for two time series at the same time. This forces another choice. Do we treat each time series independently, or do we interweave their progression. And how does that create different narratives? Feels like creating multiple views for comparison, but with animation, text and over time.

Q4: What are the pros and cons of the current workflow, i.e., the storyboard and its feature-action DPs are interpreted by a VIS software engineer who implements the storyboard by using the existing algorithms for detecting features and visualization software components for realizing actions, and creating new ones when necessary?

[A]

The workflow is suitable, but does rely on the software engineer to both interpret the work, and also create the necessary events in code. Software has needed to be created to encode these features. These functions could be published, and generalized. Because we are dealing with time-series data, there are many functions (that the s/w engineer has written) that could be used and generalized for other time-series data projects.

[B]

There are opportunities for further automation. If the approach is generic enough, the software engineer in-the-loop could be replaced by a computational entity. This implies possibly a more stringent structure than what we have, for defining a story, and therefore some form of declarative language, that would be decoded (either by a person or some form of AI) in a more deterministic manner. There is little point in making the creation of the story more structured/mathematical if one is not to exploit that down the line, even further than we have.

[C]

A number of storytelling visualization authors wrote books and gave talks about their approach, e.g., create initial data visualization using excel, transform them to higher quality visual artifacts using Adobe Illustrators with highlights and annotation, and if animation is needed, use other professional video making software. Some may write a small specific piece of software for users to access a data set through interactive visualization, e.g., as part of an online article. These traditional approaches would not scale up to many datasets as this approach can. I am not aware of any advanced tools that are currently available for enabling this approach.

For some large organizations, such as newspapers and national data providers, it is perhaps necessary, productive, and likely cost-effective for authors to be supported by professional software developers. For small organizations and individual storytelling authors, having tools for enabling them to author such storytelling visualization will be wonderful. To deal with regional data, it is not possible to use the workflow of excel-illustrator-video, or the workflow of software-per-dataset.

Q5: Would story designers (including those who do not program) wish to use a more "advanced interface" for specifying feature-action DPs, such as a database, a scripting language, an API, etc.? Do you think what may be the most appropriate for the designers?

[A]

An advanced interface could be created to be able to create these feature-actions. This could be something like a visual programming environment. Or may be a library of typical functions could be created that can be used by other designers.

[B]

Yes, linking back to the comment about a more declarative way, one could envision an interface where storytellers can pick and match story

segments (tropes?) and combine them, or use them as overarching elements. For instance, having a time series and detecting peaks through - could be easily wrapped in a single, reusable function. If anything, this makes creating the story more object-oriented (or trope-oriented?) way.

Q6: Any other comments

[A]

It was exciting to see the stories come to life. The meta instructions enables core elements of the story to be exposed that would not necessarily be created by hand. The design process has also been fun.

[B]

We mentioned it many times, but the gaming world uses similar techniques to build stories in a more procedural way (AI-generate if I am not mistaken, that is where I heard the term tropes). Any procedural approach implies a degree of abstraction which can be exploited to create a more modular approach for the end user. Like lego's put together to tell a (physical) story, bound together by one simple notch-to-gap tooth link (sorry, maybe gone a bit too far with the metaphor)

From the Perspective of Technical Development

Please feel free to add any comment from the perspective of a visualization software engineer. The questions below are just prompts.

Q1: What are the pros and cons of the current workflow, i.e., the storyboard and its feature-action DPs defined by a story designer are interpreted by a VIS software engineer who implements the storyboard by using the existing algorithms for detecting features and visualization software components for realizing actions, and creating new ones when necessary?

[D]

A programmer can create these visualizations easily by calling the correct vis functions that will exhibit the desired behavior. Given time, a formalized markup language could be created for the lookup tables. This would make it possible to generate new animations completely automatically. The set of actions is likely to expand as more stories are designed. Completely new actions, not preempted by the software engineer, can take long to implement. Such a workflow would benefit from a formalized language and set of fixed actions, however this may hinder the creative process of storytelling.

[C]

If [D] says "easily" for "a programmer", we can interpret that the demand for previous experience of visualization programming is not high. Note that meta-authors can always provide more guidance on visual representations and visual designs.

The notion of feature-action design patterns seems to be acceptable to both meta-authors and the developers. Initially, it appeared that meta-authors found it challenging to specify features. Later it became clear that this was because of the challenge in anticipating features without seeing every time series, rather than the notion itself. The meta-authors commented on that. Nevertheless, once the meta-authors had communicated their feature-action design patterns in a qualitative way, the technical developer was able to translate them relatively easily. In various meetings, there was never a situation where such feature-action design patterns needed a lot of discussion. The technical developer was able to make such design patterns into the lookup table easily.

[E]

The current story creation process can be semi-automated by creating web templates based on the existing storyboards, and providing story designers an option to write stories using decorative data interchange format, e.g., JSON, YAML.

[C]

The current workflow is perhaps the most appropriate for the need to deliver the storytelling visualization without any readily available tools, except the software environments such as Observables and D3.js. It will be desirable to develop software techniques and systems that can support such workflows. The algorithmic pipeline, the algorithms, and the software components developed in this project can provide a good basis for future developments.

Q2: When there is not an existing feature-detection algorithm, how long, on average, it may take to formulate a new feature-detection algorithm based on a specification by the story designer?

[D]

The feature detection algorithms used were rather basic. A benefit of creating stories for the general public is that the features we use have to be understandable and simple. The development time for a feature detection algorithm varied between a few hours or a few days. The algorithms had to be tuned over time to suit the data and produce more consistent results.

[E]

I think it depends on how complicated the function is. [D] has already implemented many feature detection algorithms, e.g., raise, fall, peak etc.

[C]

The speed of developing feature detection algorithms in the project was surprisingly quick, considering that this aspect was largely started from scratch. The total amount of time required to develop feature detection algorithms for processing time series data seems to be scalable as these algorithms will be reusable. Given a specific data space, such as time series data, it is possible to collect most commonly used feature detection algorithms. In the future, if we need to build storytelling visualization for imagery data, there are many feature detection algorithms in software such as OpenCV and MATLAB.

Q3: When there is not an existing visualization software component for an action (e.g., a new type of highlight or a new style of animation, how long, on average, it may take to write a software component based on a specification by the story designer?

[D]

This can take much longer to implement depending on the scale of the action. It can take up to a week to accomplish. For example, the storytelling visualization in story-3 works in a very different way to all other stories. It demanded an animated timeline, a new area for annotations to appear and a scrolling ui element for controlling the visualization. The entire development took the longest (roughly 4 days). Whereas, new annotation styles or small visual changes can take less than an hour. If the visualization team were under time pressure, it may be worthwhile to rank new actions by importance and complexity - so software developers can prioritize which actions to implement first.

[E]

The visualization components (e.g., multi-line time series, annotations, timelines, animations etc.) are implemented using d3.js and as modular software components (e.g., JavaScript class and function). Developing a new visualization component using d3.js should be quick and easy, and it will work with the existing components.

Q4: Given a specification, such as Story 2 or Story 3, how many person-days it may typically take to implement? Do you think there is a better way to reduce the cost of implementation?

[D]

It takes less than a day to implement Observable if the story requires no new actions. This time is then extended by the need to implement new actions and the time to convert an Observable notebook to a JavaScript file for the server. To reduce implementation time it may be worth directly implementing the story on the server, removing the need to use Observable. However, Observable is a very good drafting software. It is very quick to make changes and you can see those changes instantly. Whereas with Node, every change requires you to rebuild the project. This would take roughly a minute each time, greatly reducing the ability to debug issues. Thus, time gains from removing Observable may be minimal.

Another solution would be to define formalized language in the storyboards that enable the software to automatically create visualizations by parsing the storyboard. On the other hand, this would require the storyboard artists to learn this pseudo-markup language which may increase time to create storyboards.

[E]

The data processing functions (e.g., Gaussian, feature detection, semantic event, etc.) and visualizations components (multi-line time series, annotations, timelines, etc.) are implemented as reusable classes and functions. We can reuse these data processing functions and visualization components to implement any new storyboard. There are currently four stories created, and out of them, three are unique and story-4, and story-2 are similar templates. For me, who had to read and understand [D]'s code took two hours to create, test, and deploy story-4 based on story-2, and it should be much quicker for [D].

This story creation process can be reduced down to a few minutes. We can develop a user interface that story designers can use to upload a story file and specify a template. We can improve our system to automatically generate serialized story objects from the uploaded story file and save the story to our database. We can create generic templates, e.g., based on our three existing templates and save them to our database.

Q5: Would story designers (including those who do not program) wish to use a more “advanced interface” for specifying feature-action DPs, such as a database, a scripting language, an API, etc.? Do you think what may be the most appropriate for the designers? What kind of “advanced interface” is more cost-effective to develop and maintain?

[D]

A downside of the scripting language is that it is intimidating for non-technical story designers. Another option would be an interactive form-like webpage where story designers choose options from a set of drop down menus. The webpage can construct the feature-action table from this form. As a webform it should be cost-effective to maintain, however, it would take a few person-days to implement. Furthermore, it would have to be updated every time a new feature or action are created.

[E]

We can introduce a formalized markup, scripting, or data exchange language to specify stories. Using markup language (e.g., XML) or scripting language would be complicated and require more technical knowledge for story designers to specify stories and time-consuming for developers to write parsers for a markup language. Story designers should be able to specify stories using simple and declarative syntax, such as using data interchange formats (e.g., JSON, YAML, etc). It will be quick and easy for story designers to specify or write stories using declarative syntaxes and programmers to implement parsers for a widely used data-interchange format. We can generate instances or serialized objects from a story written by the designers, save the objects into our database, and generate storyboards in a webpage by processing the story objects on a web template.

Q6: Any other comments

From the Perspective of Public Engagement

Please feel free to add any comment from the perspective of a public engagement officer and/or an ordinary user. The questions below are just prompts. Please see the four visual stories at <https://vis.scrs.uk/story-boards> using Chrome.

Q1: Please comment on the mode of engagement of the stories? How would you compare telling a story about the data with the “standard” mode of online visualization of COVID-19 data, e.g., input a region and see a time series?

[F]

There has been a surfeit of data relating to COVID-19 in the public domain, which may well seem like information overload to members of the public. Without context, a time series may not convey much meaning. The story telling both provides context and divides the data into digestible segments, allowing the viewer to look at the situation leading up to an event and following after it, and to understand/reflect on each segment before moving on to the next. The context provided allows the data to be considered against both wider events, such as vaccine rollout, and time – in relation to the pandemic overall. Because the stories are told linearly and from a local area perspective, the user can situate themselves both in place and time in relation to the events, which adds to the human interest and relatability of the data.

[C*]

Two ordinary viewers, both commented that they have not seen such storytelling software, and they guessed: if it were available some time in 2020, it would get used a lot. As many places in Asia are currently having tough times, they would also be interested in such storytelling visualization showing their local data. Apparently daily data available in public is fairly detailed or localized.

Q2: Please comment on the story design and visual design of the stories?

[F]

The event descriptions are simple and effective, giving clear information about vaccination dates and other key landmarks. The speed of the animation provides momentum, and the event pauses give the user control over how fast to move on with the story. The instructions are clear and easily understandable. The title or headline (Basic story of COVID-19 in Bolton / Comparison of waves between...) give the user an understanding of what to expect / what they are about to see. The colors provide differentiation in the comparison story however consideration should be given to users with color vision deficiency and to accessibility options for visually impaired users.

[C*]

Comments from two ordinary viewers. Simple is good. Because it takes only one selection button for a region and then the play button to start, it is actually slightly easier than many search interfaces. Because it is easy to use, it makes one want to try a few regions in addition to my own. Ideally, there are more regions available, and there is a map for selecting a region.

Q3: Please comment on which section of the public may be interested in viewing such visual stories?

[F]

The stories are likely to appeal to members of the public who are regular news viewers, who listen to scientific or factual podcasts, take an interest in the context of the pandemic and how policies or actions affect its progress. These individuals may be actively searching for more information in order to develop a deeper understanding. They are likely to share and discuss facts they have learnt from reading and viewing documentaries, podcasts and in-depth articles, and to have an understanding about misinformation and what sources of data are reliable.

[C*]

Both ordinary viewers said that they would use the software if they were available in 2020, and perhaps early part of 2021. As not many people are wearing masks nowadays, they do not feel a need at the moment.

Q4: In the process of developing these stories, we confirmed that the techniques for realizing such storytelling visualization are not trivial and do not seem to exist before. This might explain why such a form of public engagement was not available during the COVID-19 pandemic. Do you think any other reasons that may hinder such a form of public engagement?

[F]

The sheer proliferation of data and the length of time that COVID-19 data has been available to people may hinder public engagement, through ‘news fatigue’. It may be necessary to communicate a compelling ‘USP’ (Unique Selling Point) for this style of storytelling visualization ie what makes it different from other data visualization available during the pandemic

The fact that it is a very visual technique may hinder engagement by some groups of the general public - and the format will not appeal to all.

[C*]

Both ordinary viewers felt that they did not know enough to answer this question. But they said: if it is too hard to do, why not have such techniques for other types of public data that are always available and interesting to some people, such as local house prices, school performance, transport stats., etc.

Q5: Designers of storytelling visualization are used to design a storyboard for a specific piece of data. It took some effort for the designers in this project to become comfortable in designing a “generic” and “intelligent” storyboard for time series in different regions. Do you think other designers will be able to design storytelling visualization?

[F]

This issue is not limited to visualization but is a common tension area between what researchers/scientists want to /are able to communicate from data or research results, and what constitutes a good ‘story’ to make it engaging and palatable to the public, and to make it stand out in an information saturated context. Designers may feel they are commenting on/assigning an opinion to the data rather than simply presenting it – by presenting an event alongside data, weight is given to the event i.e., by its inclusion the inference is that it is significant.

[C*]

Both ordinary viewers felt that they did not know enough to answer this question.

Q6: Do you think if one may apply this technology to other types of data, e.g., house prices in a region, crime rate in a region, etc.?

[F]

I can imagine this sort of storytelling would be useful and of interest to individual people and groups in terms of significant and topical events, for example the rise of energy and fuel prices, and the context of world events. Comparison of different cities, areas or regions would be useful in terms of house prices or crime rate.

[C*]

See Q5

Q7: Any other comments?

[F]

From a communications point of view when telling a story a key question we always keep in mind is ‘so what?’ ie what will this mean to the reader/viewer? What is the impact? This sometimes has a disconnect with ‘science for science’s sake’, i.e., science for the sake of knowledge. A communications professional’s job is to pinpoint the knowledge that will be accessible, engaging and understandable to the audience. So for the designer or researcher coming at a story from the point of view of ‘what story are we trying to tell here’ may be counter-intuitive – putting the cart before the horse.

[C*]

One ordinary viewer asks if it would be possible for an individual to create such a story to share with friends. Some people are quite keen to share data, such as their daily walking step counts.

Appendix F: Evaluation Records – ML Storyboards

Original texts collected from ML experts (M, N, O, and P) during the evaluation process.

Q1: How do you feel about viewing ML results using computer-generated storytelling animation?

[M]

I found it genuinely helpful. ML workflows can run for a long time (in my case study, workflows lasting up to 18 months), and it is easy to lose track of what happened during an experiment. The animated stories replay the events of a run in order and point at the moments that matter, such as accuracy jumps or best-so-far configurations. The provenance story in Fig. 9(a) of the paper shows this, and the video demonstrates the animations in action.

[N]

The animation is refreshing, and I kind of expect this would be done more or less someday, and this could be very useful for those who are not in the VIS community. For complex data, the animation can reduce visual clutter and draw the attention to those interesting points automatically, this is particularly useful. The provenance story also shows that it can be difficult to catch some hidden relationship from a static plot as we may need to connect events in our mind across time. It can also save time for long-running experiments when we have too many logged events. So people don't miss the important points.

[O]

I really do think this animation for ML model results is useful, especially when time span of the project is long, like you have mentioned in the paper. I especially like the PCPs in the multivariate storytelling study because there's always multiple hyperparameter combinations to select. You do not know which one is optimal given a dataset and a task before the model is really trained and tested. This design definitely helps the hyperparameter selection because you can simply see the best combination in one single plot, and we all know hyperparameter selection is important. I think this is pretty much like doing grid search of hyperparameter but here you are able to visualize it. I think this also gives you quick access to other suboptimal combinations, which could be also insightful. Based on this, is it possible also to highlight the importance of different hyperparameters (e.g., kernel size, layer, channel) and the importance of the different values used for each hyperparameter.

[P]

Yes, I will watch, it would be effective for me to understand how the whole pipeline work, especially for iterative or parallel algorithms.

Q2: In comparison with static visualization, does computer-generated storytelling animation have any merits or demerits?

[M]

Merits: animation provides temporal ordering and guided attention - it tells the viewer where to look and in what sequence. I found the parallel coordinates plot easier to understand when the story revealed it event by event; I think that animation reduces the confusion caused by clutter.

Demerits: animation is slower to consume than a static chart, and comparing two distant events is harder.

[N]

Merits: we can have guided exploration so that the cognitive cost is lower. The guidance is not just static, but can be time-series. Static visualization need people to do this manually. Also, the creation of MSD is automatic, so that it saves time and cost and there is no need to do static ones with redesigning or duplication.

Demerits: the viewer needs to remember things in a long story, and if there are many important events though time, it could be difficult. And there could be some bias from the meta-author (? if i understand the paper correctly). The meta author has the right to decide which feature is important, so it is also a distortion.

[O]

Yes, I believe it produces the dynamic process of model training and testing especially when timeline is important in this context. It might also help the team to do the project management. For example, given the current plots, how long to go and how many models more should we test.

Meanwhile, if I'm not mistaken, I believe the important features to be shown in the plots are pre-defined. If the team decides to change the model or the testing metrics rather than the Accuracy, you might need to change your underlying rules accordingly.

[P]

When I initially look at a figure, it needs me to spend some effort to understand how to read that figure, needs me to keep reading and

connecting the description text in paragraphs with the figure - this is good for initial understanding/onboarding with the problem/project; the bad side is I would worry that the animation could only show things that researcher already understand (in a clearer way, if made by researcher), or only show per-instance level information (require high workload to watch and compare the animation of each instance, often many instances). Another issue would be when I am using the visualization to debug some thing, the problem may only appear at one component of the whole system, while statis vis can quickly provide snapshots of each component, the animation may always show a few components and that may cause waste of time.

Q3: How do you feel about the facts that these storytelling animation clips are automatically generated using an API?

[M]

Nobody would hand-author a story for every ML training run, so the meta-storyboard (MSB) approach is extremely useful as it lets an author define feature-action patterns once, and the MSB API then generates stories automatically for new, previously unseen data streams.

[N]

It kind of lower the barrier to entry this area, and the scalability allows people to generate stories with consistency without introducing too much bias. The only thing I feel not sure is that, as it has feature-action rules defined by the meta author, can the API understand cultural significance very well? something can have significant peak but politically or socially important event might not be trigger by that peak, just like section 5.4.

[O]

This is actually impressive. I have actually seen and used one of these applications (visualizing the training process), but it was embedded in a dedicated deep learning package and was used for a specific task and dataset for demonstration only. Instead, if you can generate animation with an API, that means its generalizability is much better compared to the case I mentioned, which is going to be very useful. If we can easily do this with simple addons, rather than spend time thinking of how to do it, I believe its utility is much more universal.

Q4: For ML team leaders, if it is not costly, would they consider such computer-generated animations useful?

[M]

Yes, team leaders can quickly observe the progress and provenance of all ML workflows, and auto-generated stories are an efficient way to do that without reading raw logs.

[N]

Personally I would say, yes, definitely. This is quite obvious, not only based on the paper's evidence, but more like from the logical needs of ML management, given that it is not costly. Most ML teams have rather considerable number of workflows simultaneously, and sorting too many things out for a human being is not a very nice thing to do. So people do need this tool to observe, update, and then have feedback and adjustment. Also, it could be very helpful to people new to the team. Just like this paper, I would spend less time if I can see the animation to understand whats going on.

[O]

I would think so. Deploying such tools would not cost too much computations power while it could give you some additional insights on model development and save you some time for hyperparameter selection.

[P]

I personally will definitely find this very useful, because it's always a hard thing to sync the understanding of all teammates, no matter be that useful or not as static vis for each person daily debug and develop. Moreover, there are other tools available, like TensorBoard, MLflow, Weights & Biases, Neptune, ... may worth comparing

Q5. For ML researchers / developers, without any tool for generating such animations automatically, would they be happy to create these routinely?

[M]

No.

[N]

No. at least without my visualization education, i do not have such habit and people often do not realize how visualization could be important to literally anything. People won't spend time to do something non-scalable (like the static ones, need to redesign every time), the cost is simply too much. Also, its more like a mismatch of skills. ML researcher normally won't pay much attention on the visualization and cannot realize that this could actually help the ML research very well, and they cannot create visualization very well.

[O]

To be honest, no. Like I said in Q3, if I could do it simply with an API, I would love to. But instead if I have to write extra codes to implement this function, and this might be only useful in very limited scenarios, I would not bother to. So I reckon that the accessibility is very important for this software.

[P]

Personally no, would only consider making these if need to present a large project to others with little context. One animation per experiment is too much work and static vis may already carry enough information for daily research.

Q6: Do you have any suggestions for future development of meta-storytelling, e.g., the concept, algorithm, API, visualization plots, user interfaces, and any other aspects.

[M]

- Providing standard templates, or declarative formats (JSON/YAML) so meta-authors need less developer help.
- More ML-specific feature detectors, e.g., train/test divergence (overfitting), loss plateaus and instabilities.
- Integration with common ML logging tools (e.g., MLflow, TensorBoard) so the API can consume existing log formats.
- Natural-language or LLM-assisted authoring of feature-action tables.
- Exporting stories as short video clips for sharing.

[N]

So current algorithm uses a k segments defined by the peaks, which i believe can be dynamically changed by algorithm based on the data's information entropy. If this year is boring, then just skip it, it does not necessarily need to be the peak that decides the k. But considering the importance of the peak, I am very happy to accept the current version.

Also, a slider along the timeline could be very helpful, didn't see that in the paper, some plots are not very clear and detail cannot be seen clearly.

In ML, there could be too many visualization that can be used to monitor the overall process, which means it can be task-adaptive. So select which accuracy metric (or multiple ones) can be the meta-author's decision.

[O]

The current version is built on the log files after the model training and testing. If it can be implemented like real-time visualization during the training, it will be more interesting. For more complex models such as transformers or even large language models, the training process is complicated, and it might require extra attention for hyperparameter selection. For example, transformer model training might experience several glitches and spikes during loss descending. Bad parameters might lead to model to be totally poorly trained. If I could spot such failures at the very early stage, I could save a lot of time and computational power, and re-do the training with another set of parameters.

[P]

This pipeline may also leverage other existing vis analysis tools inside it; uncertainty features (significance, seed variance, confidence bands) may be considered as well before deciding the best accuracy result; more ML features (may or may not need); let the viewer re-rank/filter what's important, not just the author.

Q7: Please let us know if you have any other comments.