ProtoSteer: Steering Deep Sequence Model with Prototypes

Yao Ming, Panpan Xu, Furui Cheng, Huamin Qu and Liu Ren

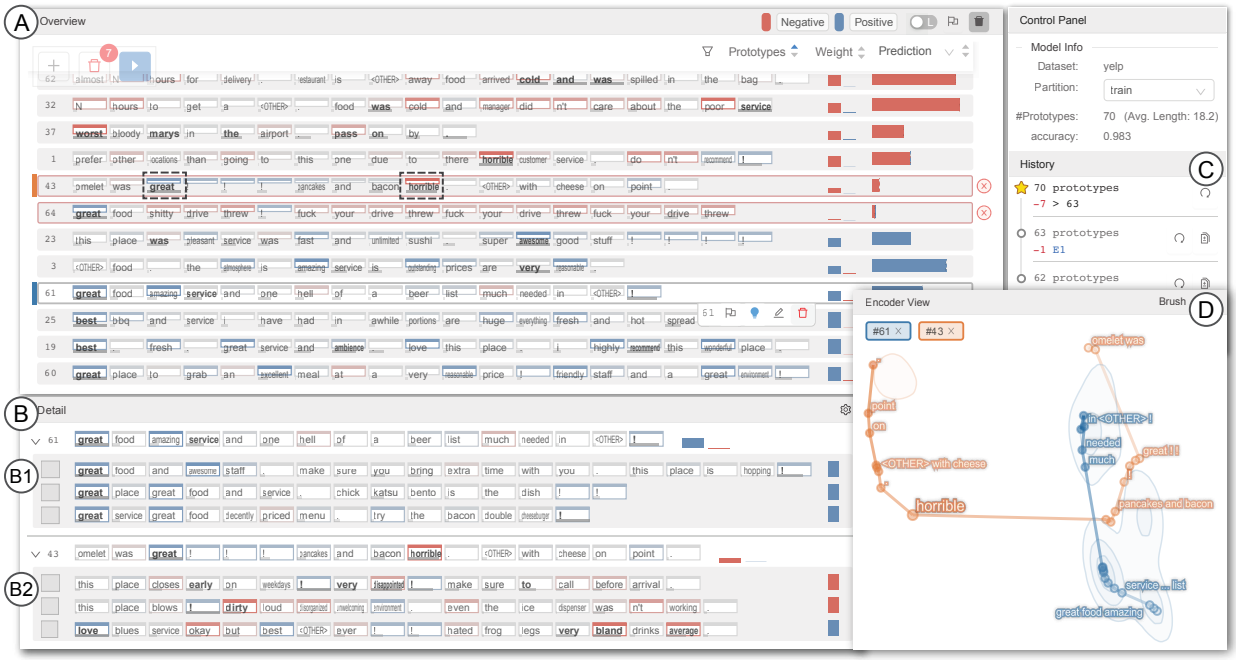

Fig. 1: The ProtoSteer interface for interactively refining prototype sequence network. The *prototype overview* (A) presents a list of prototype sequences and their statistics on datasets. The *sequence detail view* (B) displays the neighboring sequences of selected prototypes. The user can interactively add, delete, and revise prototypes. All edits are traceable in the *editing history* (C), where the user can easily compare or revert changes. For advanced analysis, the *sequence encoder view* (D) projects prototypes as trajectories with a contour map showing its neighboring hidden state distribution.

Abstract— Recently we have witnessed growing adoption of deep sequence models (e.g. LSTMs) in many application domains, including predictive health care, natural language processing, and log analysis. However, the intricate working mechanism of these models confines their accessibility to the domain experts. Their black-box nature also makes it a challenging task to incorporate domain-specific knowledge of the experts into the model. In ProtoSteer (Prototype Steering), we tackle the challenge of directly involving the domain experts to steer a deep sequence model without relying on model developers as intermediaries. Our approach originates in case-based reasoning, which imitates the common human problem-solving process of consulting past experiences to solve new problems. We utilize ProSeNet (Prototype Sequence Network), which learns a small set of exemplar cases (*i.e.*, prototypes) from historical data. In ProtoSteer they serve both as an efficient visual summary of the original data and explanations of model decisions. With ProtoSteer the domain experts can inspect, critique, and revise the prototypes interactively. The system then incorporates user-specified prototypes and incrementally updates the model. We conduct extensive case studies and expert interviews in application domains including sentiment analysis on texts and predictive diagnostics based on vehicle fault logs. The results demonstrate that involvements of domain users can help obtain more interpretable models with concise prototypes while retaining similar accuracy.

Index Terms—Sequence Data, Explainable Artificial Intelligence (XAI), Recurrent Neural Networks (RNNs), Prototype Learning

1 Introduction

In recent years we have observed a growing adoption of deep learning models in sequence data analysis to assist decision-making. Deep sequence models, esp. recurrent neural networks (RNN) can be used to predict patient status by modeling electronic health records (EHR), analyze the topic or sentiment of texts, understand audio signals and have achieved state-of-the-art results in most applications [23, 27].

Despite their impressive performance, RNNs are often considered as "black boxes" due to their complex architecture and massive size of model weights. The lack of ***interpretability*** limits their adoption in many critical decision-making scenarios, where the experts need to understand the reasons of the prediction before deploying the model in production. The recent data protection regulation of The European Union also grants individuals the "right to explanation" for decisions made by machine learning systems [35].

The urgent demand for interpretability has motivated recent research in explaining deep sequence models. Researchers have developed a

- *Yao Ming, Furui Cheng, and Huamin Qu are with Hong Kong University of Science and Technology. E-mail: ymingaa, fchengaa, huamin@.ust.hk.*
- *Panpan Xu, Liu Ren are with Bosch Research North America. E-mail: panpan.xu, liu.ren@us.bosch.com.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

variety of post-hoc explanation techniques to unveil the inner workings of RNNs by visualizing hidden state values [21], identifying salient words in texts [25] and extracting rules from hidden state changes [34]. In the meantime, visualization is becoming an increasingly important approach to help machine learning researchers [18, 21, 28, 44, 45] analyze and diagnose RNNs, and enable domain experts [23] to understand and evaluate RNNs for application-specific tasks.

In addition to interpretability, another challenging issue with the deployment of deep sequence models is their ***steerability*** [3,5]. Many applications will benefit from supporting the domain experts to directly steer the models with their insights and domain-knowledge. For example, a doctor with rich experiences in heart diseases would have valuable knowledge that can benefit a heart failure risk prediction model [23]. Though end-to-end training of deep neural networks is an effective learning paradigm which largely alleviates the need to manually curate data features, it fails to provide the functionality for expert users to steer the model. A machine learning approach that is both interpretable and steerable allows the domain experts to gain insights from the model and in the meanwhile refine the model with their knowledge, creating a mutually beneficial feedback loop.

In this work, we propose a novel interactive visualization approach to *interpretable* and *steerable* deep sequence modeling. We base our approach on case-based reasoning [22], which imitates the natural problem-solving process in people's everyday lives: by consulting similar cases in the past we formulate solutions for new situations. For example, doctors perform diagnostics by recalling similar patients in the past and mechanics identify solutions to machine malfunctions by referencing similar cases before. To realize such case-based reasoning on sequence data, we utilize an inherently interpretable sequence model named ProSeNet (Prototype Sequence Network) [31]. ProSeNet constructs a small set of *prototypical* sequences from the original training data. New inputs are compared with the prototypes in a latent space for inference (Fig. 2). Based on the interface of prototypes, it is now possible to enable steerability as the end-users can directly refine the model by adding, deleting or revising the prototypes.

Involving the end-users in iterative model inspections and refinements brings several benefits: 1) the prototypes constructed by the optimization algorithm may not align well with the domain knowledge, engaging domain expertise can help construct more meaningful and representative prototypes; 2) setting hyperparameters (e.g. number of prototypes) in the model can be a challenging task and may result in sub-optimal results, with visual inspection we can identify the redundancy or insufficiency of prototypes for further refinement.

We therefore design **ProtoSteer** (**Proto**type **Steer**ing), an interactive visual interface to help domain experts inspect, critique and revise the prototypes in ProSeNet. The goal is to support the iterative model refinement for end-users with limited background in machine learning. The design of the system follows the guidelines for constructing interactive machine learning (IML) interfaces as described by Amershi *et al.* [3,5], Dudley and Kristensson [11]. In particular, we distill the following main design requirements:

- Provide explanations of model output and overview of model behaviour.
- Support model diagnostics by visualizing fine-grained performance metrics (e.g. accuracy) to pinpoint the source of errors.
- Reduce user effort in steering the model by leveraging computational support and presenting informative visual cues.
- Convey the consequences of incremental model updates and provide undo and redo options.

We consider these generic design requirements as well as the unique characteristics of ProSeNet to reach the final design of ProtoSteer. To evaluate the effectiveness of ProtoSteer, we conduct extensive case studies and expert interviews. The results demonstrate that user interactions help the system learn more meaningful and concise prototypical representations of sequence data without sacrificing the accuracy. The case studies cover applications in two distinct domains: sentiment analysis on texts and predictive diagnostics of vehicle faults, which illustrates the general applicability of our approach.

The main contribution of this paper is summarized as follows:

- A human-model interaction scheme which helps refine the learned knowledge of both the human and the model.
- The design and implementation of a visual analytics system that allows human-guided refinements of a deep sequence network through interactively editing a set of prototypical sequences.
- Case studies with real-world datasets and expert interviews which investigates the effectiveness of direct steerability of deep sequence networks via interactive visualizations.

2 Related Work

2.1 Sequential Data Visualization

Sequential data visualization is an increasingly important research topic due to its wide applicability in many real-world problems such as clickstream analysis [1, 29], electronic health record (EHR) analysis [15, 36] and machine/vehicle log analysis [9].

One of the main goals of visualizing such data is identifying salient sequential patterns. This can be achieved through novel visualization and interaction designs as proposed in Outflow [49], CareFlow [36] and DecisionFlow [15] and MatrixWave [51]. Due to the increasing volume and complexity of sequential data [10], recently researchers also combine sequence mining techniques (e.g. frequent sequential pattern mining and event sequence clustering algorithms) with interactive data visualization to support explorative analysis of large scale sequence data. Examples of research in this category include FP-Viz [43], TimeStitch [38], Frequence [37], Peekquence [24] and recent work by Wang *et al.* [48], Liu *et al.* [29], Chen *et al.* [9] and Guo *et al.* [16, 17].

Despite the rich literature in sequence visualization, most of the techniques focus on unsupervised pattern analysis. Our work supports predictive analysis tasks on sequence data with explicitly defined labels. The system visualizes *task relevant* prototypes extracted from the original training data as both visual summary and explanation of the model's inner-workings.

2.2 Deep Sequence Learning and Model Interpretation

Deep sequence models esp. RNNs have achieved impressive performance and are widely adopted in many application domains such as audio signal processing, natural language understanding, and electronic health data analysis [27]. However due to the highly nonlinear transformations and the massive number of parameters in those models, they are usually considered as 'black-boxes', which are difficult to comprehend and interact with.

Enable interpretation of the model is usually a first step towards building IML systems [3, 5, 11]. In recent years quite a few techniques have been developed for post-hoc explanation of deep sequence models by visualizing hidden state changes [21, 45], calculating the importance of tokens with regard to the model outputs [2, 8, 33, 34], or mimicking the behaviour of a complex model with simpler ones through model distillation techniques [39, 40]. Post-hoc methods shed light into the decision-making process of the models, however it remains unclear how adjustments could be made to affect the model behaviour. Deep sequence models with attention mechanisms are inherently explainable with the attention weights indicating the importance of each input token [27]. RetainVis [23] visualizes the attention weights and allows users to edit them based on their domain knowledge directly.

ProtoSteer is built on a deep sequence model with inherent interpretability. The model named ProSeNet performs case-based reasoning with a set of exemplary cases (i.e. prototypes) constructed from the historical data. The prototypes explain the model behaviour and enable the end-users to inspect, critique and refine the model based on their expertise in the domain.

2.3 Interactive Machine Learning (IML)

Interactive machine learning complements computational data modeling with direct feedback from the end users [11]. Through careful interaction design, users with no or little expertise in machine learning can apply their domain knowledge to obtain more effective and interpretable models. A few surveys nicely summarize this research field.

Amershi *et al.* [3, 5], Dudley and Kristensson [11] identify the existing approaches and the UI design requirements to involve human-in-the-loop in machine learning systems. VIS4ML [42] formulates an ontological framework to understand how current visualization systems support the various steps in model development. Boukhelifa *et al.* [7] summarize methods for evaluating interactive machine learning systems.

User interactions in IML come in many different forms. Existing IML systems allow users to directly specify the predictive rules [6, 47], provide data sample and/or labels to the system [13], interactively edit the cost of misclassifications in a confusion matrix [20] or adjust the weights of different predictors in ensemble models [46]. The design of the user interface usually has to take several factors into account including the users' expertise, the machine learning model and the properties of the application data [11]. Recently, researchers have also built systems to support interactive sequence learning. One example is RetainVis [23], which allows the users to interactively adjust the attention weights on different tokens in a deep sequence model.

Our work explores a novel approach for interacting with a deep sequence model: users specify *prototypes* which are exemplary sequences and the model performs inferences on new inputs by comparing them with the prototypes. The aim is to build an interpretable model using case-based reasoning [22] with prototypical examples curated by the experts. The system does the heavy-lifting of learning appropriate latent representations of the sequences while the users make observations on the model behavior and revise the prototypes.

3 Background

In this section we briefly introduce ProSeNet [31], which combines deep sequence networks (e.g. LSTMs) with prototype learning to achieve high predictive accuracy as well as interpretability.

3.1 Interpretable Sequence Learning with ProSeNet

Most of the recent research on deep sequence model interpretation focus on post-hoc analysis to obtain the sensitivity of the variables or importance of the individual input tokens [8, 34]. Instead of relying on post-hoc explanations, ProSeNet achieves built-in interpretability through prototype learning. In particular, prototype learning memorizes or constructs a small set of exemplary cases (i.e. prototypes) from historical data and later on refer to those prototypes to perform classification/regression tasks on new inputs.

As illustrated in Fig. 2, ProSeNet consists of three major components: the sequence encoder r, the prototype layer p and the fully connected layer f with softmax outputs for classification tasks. The sequence encoder r converts a variable length sequence s into a fixed length vector representation $\boldsymbol{e} = r(s), \boldsymbol{e} \in \mathbb{R}^m$ using deep recurrent neural network encoders (*e.g.*, LSTMs and Bidirectional-LSTMs). The prototype layer p compares the latent vector representation $\boldsymbol{e}$ obtained through the encoder network with k prototype vectors $\boldsymbol{p}_i \in \mathbb{R}^m$. Through appropriate transformations (Appendix B) we obtain a vector of similarity scores $\boldsymbol{a} = p(\boldsymbol{e}), \boldsymbol{a}_i \in [0,1]$ where $\boldsymbol{a}_i$ is the similarity score between the input sequence and the prototype $\boldsymbol{p}_i$ and $\boldsymbol{a}_i = 1$ indicates that the input sequence has identical embedding with prototype $\boldsymbol{p}_i$. The fully connected layer f with softmax output computes the eventual classification results using the similarity score vector $\boldsymbol{a}$. We constrain the entries in the weight matrix in f to be non-negative for better interpretability. The similarity scores $\boldsymbol{a}$ together with the entries in the weight matrix can be used to explain the classification results.

The prototype vectors $\boldsymbol{p}_i \in \mathbb{R}^m, i \in [1,k]$ learned through back-propagation are not readily explainable as they may not correspond to sequences in real-world applications. Therefore the optimization algorithm performs ***prototype projection*** for every few training epochs, which essentially identifies sequences from the original training data that are most similar to the prototype vectors in the latent space. The prototype vectors are then updated with the latent vector representations of these sequences.

We use an example from sentiment analysis of restaurant reviews to illustrate how prototypes can be used to explain the classification results. The model predicts the input sentence as negative and the most similar prototypes are used to explain the result:

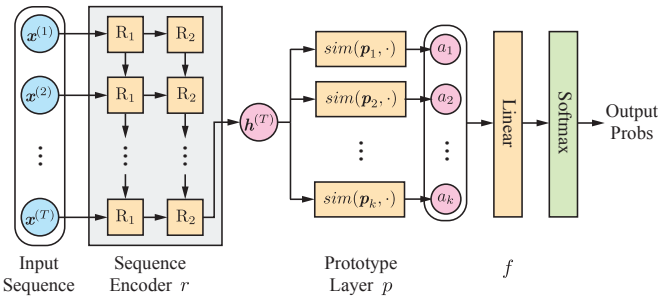

Fig. 2: ProSeNet Architecture. It consists of the sequence encoder layer r, the prototype layer p, and the fully connected layer f with softmax outputs for classification tasks.

Input:	pizza is good but service is extremely slow
Prediction:	Negative
Explanation:	0.69 * good food but worst service (Negative 2.1)
	+ 0.30 * service is really slow (Negative 1.1)

The factors in front of the prototype sequences (0.69 and 0.30) are the similarity scores between the input sequence and the prototypes. Their associated weights in the fully connected layer f are displayed at the end of each prototype. The weights can be interpreted as how confident the prototypes are in determining their neighborhood instances to have negative sentiments.

3.2 Training ProSeNet with User-specified Prototypes

A few issues still remain for automatically learned prototypes. First, it is unknown whether the prototypes align well with the domain knowledge. Second, the identified prototype sequences usually contain redundant or insignificant events which has little contribution to predictive tasks. It is difficult to design automatic simplification techniques that can keep sequential structure that conforms to domain-specific rules (*e.g.*, the grammar in language-related tasks).

Therefore, it is beneficial to incorporate the input from domain experts to search for an optimal set of prototypes. ProSeNet supports incremental updates of the model with user-specified prototypes. In particular, the set of prototypes can be edited by adding, deleting, or revising existing prototypes. Based on the user specification, the incremental training could adjust the sequence encoder r and the fully connected layer f while fixing the set of user-specified prototypes. Empirical experiments show that after retraining it is possible to obtain accuracy on-par with or even superior compared to the original model. The detailed incremental training procedure are described in Appendix B.4.

4 Design Requirements

The goal of ProtoSteer is to directly incorporate input from the domain experts in the model instead of relying on machine learning practitioners as intermediaries for model adjustment. The workflow therefore is different from the traditional paradigm of machine learning model development [11] and emphasizes a more rapid iteration cycle [3]. In this section, we formulate the detailed design requirements of ProtoSteer by surveying related work in interactive machine learning (IML). We also periodically collect feedback from automotive engineers, researchers in natural language processing (NLP), and machine learning practitioners to iteratively refine the design.

In several recent surveys, researchers have summarized a common set of user interface components and guidelines for designing IML systems. Fails and Olsen [13] proposed a model for IML and highlighted the training, feedback, and correction cycle, which are friendly for users with limited technical backgrounds. Dudley and Kristensson [11] analyzed the structural components of IML interfaces including sample review, feedback assignment, model inspection, and task overview. Amershi *et al.* [4, 5] identified design guidelines for general human-AI interaction through extensive user studies.

We adopt the high-level system structure proposed by Dudley and Kristensson [11] and design the workflow correspondingly. After the initial training step, the user first *inspects the model* by evaluating the

learned prototypes. The user then provides *feedback* to the model by adding, modifying or deleting some prototypes. After that, the user commits the updates and the model is *fine-tuned* with the updated prototypes. The process went on iteratively until the user is satisfied with the results.

When designing ProtoSteer, we follow the general design requirements formulated based on the recommendations of the mentioned studies [5, 11] and instantiate each item by considering the characteristics of our study.

R1 Provide explanations of model output and overview of model behaviour.
R2 Support model diagnostics by visualizing fine-grained performance metrics (e.g. accuracy) to pinpoint the source of errors.
R3 Reduce user effort in steering the model by leveraging computational support and presenting informative visual cues.
R4 Convey the consequences of incremental model updates and provide undo and redo options.
R5 Progressively introduce advanced analysis tools for expert users.

R1: In ProSeNet, the prototypes are the keys to understand the model's decision-making process since the prediction results can always be attributed to the nearest prototypes of the input in the latent space. Therefore, it is necessary to have a comprehensive understanding of the prototypes and their roles in the model. In particular, the system should help users answer the following questions:

Q1 *What are the learned prototypes? To what extent do they affect the model's decision?* Displaying the prototypes is the first step towards understanding the model behaviour. Each prototype is associated with a set of weights indicating their relative importance in determining the outcome. For example, a prototype may have a strong association with class "positive". An input that is very similar to this prototype is more likely to be predicted as "positive". Visualizing the weights help identify the most critical prototypes for a certain outcome as well as the negligible ones.

Q2 *Which prototypes are similar to each other?* The system should group similar prototypes to help users identify redundant ones that can be removed from the model without affecting its performance.

Q3 *What are the sequences represented by a prototype?* The users can better assess the representativeness of a prototype by looking at the sequences closest to it in the latent space defined by the sequence encoder. If the prototype cannot satisfactorily represent its nearby sequences the user can specify alternative ones.

R2: Since erroneous results originate from the wrong assignment of prototypes in ProSeNet, by visualizing the performance metrics associated with each prototype it becomes possible to pinpoint where the model fails and which prototype needs to be further refined. Hence the system should help clarify the following questions:

Q4 *How accurate are the prototypes in determining the outcomes of nearby instances? What are the instances classified incorrectly?*

R3: To better engage the users in the iterative model refinement process, the system should provide appropriate visual cues and interactive assistance for revising the prototypes.

Q5 *What are the key events in a prototype determining its output?* To better combine users' domain knowledge with the knowledge learned by the model it is useful to visually present the importance of different events. Besides that, a desirable property of a critical event is that it should consistently have matching events in the neighborhood of the prototype. Therefore we also visualize such matching quality information to support prototype editing.

Q6 *What are the potential candidates for a new prototype?* Creating new prototypes can be a difficult task without any suggestions or examples. Therefore it is desirable that the system can provide recommendations with simple constraints specified by the users.

R4: Finally, in the user-model interaction loop it is necessary to provide appropriate feedback and visibility of the changes such that the users can understand what are the impacts of his/her last update. More concretely, we aim to help users understand the major changes happening to the model:

Q7 *How does the overall performance of the model compare with the previous version? How do the nearby instances change for each prototype and how their predictive results change?*

R5: Our system also shows hidden state information in the sequence encoder r as an advanced feature for more experienced users. The users can be better informed about the models' internals to perform prototype editing, especially regarding the two analytic questions ***Q5*** and ***Q2***. Visualizing hidden state information is commonly seen in many systems for deep sequence model diagnostics including LSTMVis [45], ActiVis [19] and Seq2Seq-Vis [44]. In recurrent neural networks, the change in hidden states usually indicates the occurrences of key events in the sequence [21, 45]. By visualizing how the hidden states change for each prototype the experts can identify the most critical events to keep when simplifying the prototypes (***Q5***). On the other hand, if the hidden state distribution of two prototypes, as well as their neighborhoods are similar it is extremely likely that they can be merged to a single prototype without affecting model performance (***Q2***).

5 PROTOSTEER

In this section, we first introduce the overall architecture and the main components in the visual interface in ProtoSteer. Then we describe the visual designs and the interactive features in detail. When describing the features we refer back to the design requirements in Sect. 4 and show how the system addresses each of them.

5.1 System Architecture

Fig. 3 gives an overview of the system architecture. The system contains a storage module for data and model, a model manager, an analytic module, a query module, and a front-end visual interface. The visual interface contains a rich set of visualizations to support *model inspection* and a series of user interactions to support efficient user *feedback* to the model. The model manager manages snapshots of the model and supports *fine-tuning*, undo, and redo upon requests. The storage module contains the training, validation and test data to support incremental model training. The analytic module collects statistical summaries which could help users better understand the model behaviour. It also performs comparisons between different snapshots of the model to track the changes. Through the query module, the users can search the sequences in the data storage to create new prototypes.

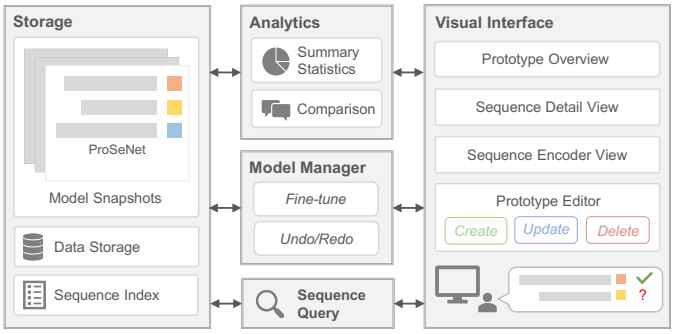

Fig. 3: The ProtoVis system consists of a storage layer, a service layer (containing the model manager, the analytics module and the sequence query module), and a visual interface.

5.2 User Interface Overview

As illustrated in Fig. 1, the main component of the visual interface is the *prototype overview* (Fig. 1A), which displays all the prototypes in the model with rich information to help users understand the model behaviour (***Q1***, ***Q2***, ***Q4***, ***Q5***). In the *prototype overview*, we also visualize the incremental changes after the users edit the prototypes and update the model (***Q7***). The *sequence detail view* (Fig. 1B) displays the neighborhood instances of selected prototypes as well as the prediction results (***Q3***). The *sequence query/edit widget* (Fig. 5C) is displayed on demand to help users query the database when creating or editing prototypes (***Q6***). The *editing history* (Fig. 1C) helps users track and

reverse their edits to the model if necessary. The *sequence encoder view* (Fig. 1D) displays the hidden state information in the encoder layer for advanced users to analyze sequence alignment and compare different prototypes (***Q2***, ***Q5***).

5.3 Prototype Overview

The *prototype overview* (Fig. 1A) displays all the prototypes in the model with necessary information to support detailed inspection of the model behaviour (***Q1***, ***Q2***, ***Q4***, ***Q5***). Each prototype is visualized as a sequence of horizontally arranged rectangular events with event names displayed as text labels. The border color encodes the type of the event (different color encoding scheme is used for different application scenarios). The *sequence detail view* (Fig. 1C) displays the neighborhood instances for user selected prototypes in the *prototype overview* (***Q3***).

Visualize prototype weight (*Q1*). On the "weight" column we display the relative importance of each prototype in determining the different possible model outputs. In classification tasks, it shows the strengths of the association between the prototypes and different class labels. The values displayed are the corresponding entries in the weight matrix W of the fully connected layer f. More specifically, W_{ij} is the importance of the prototype $\boldsymbol{p}_i$ in determining the model output to be label j (Appendix B). The categorical colors differentiate the labels in the classification task, similar as in the "prediction" column.

Summarize prediction performance (*Q4*). For each prototype we compute all the data instances that are closest to it and we refer to those instances as the ***neighborhood*** of the prototype. We show a summary of the model decisions on those nearest neighbors as a stacked horizontal bar displayed on the "prediction" column (Fig. 4C). The width of the bar represents the total number of neighborhood instances. The categorical color indicates the decisions made by the model in the classification task. We use solid blocks to indicate correct classification results and patterned blocks to indicate errors. The width of each colored block represents the number of instances that are correctly / incorrectly classified to the corresponding label. The visual encoding helps users quickly identify important prototypes that represent a large number of instances as well as prototypes which are usually associated with inaccurate prediction results.

Reorder the list (*Q2*, *Q4*). The list of prototypes can be sorted based on several different criteria to address a variety of analytic tasks. By default, the system sorts the prototypes based on their similarity. It performs a hierarchical clustering of the prototypes based on their distances in the latent space and obtains a linear order accordingly, similar as in [12]. Sorting based on similarity groups prototypes that resemble each other and it is therefore easier to spot redundancy (***Q2***). The user can also sort the prototypes by the accuracy of the prediction results. The most problematic ones can be brought to attention for further analysis (***Q4***).

Filter the prototypes. The number of visible prototypes can be reduced by filtering in two different ways: 1) the user specifies a few events and the list will only display prototypes containing one of the events 2) the user selects a prototype and the list will display the top-k prototypes most similar to it in the latent space. Filtering by event helps users narrow down the search if there are interested in a particular one. Filtering by similarity help users identify potential duplicates (***Q2***). In our implementation, we set k to 10.

Visualize event importance (*Q5*). We display the importance of the event as a horizontal bar at the bottom of the rectangle (Fig. 4C). The length is proportional to an importance score to help highlight the most critical events. The importance of each event e in a prototype p is calculated with a leave-one-out strategy. More concretely, the algorithm performs the following steps for each e:

1. Remove e from prototype p to obtain a new sequence p'.
2. Compute the distance between the original prototype p and p' in the latent space defined by the sequence encoder r.
3. Convert the distance to a normalized value between 0 and 1, where 0 indicates that removing e has no effect on the latent representation of p. The normalized value is the importance of e.

The intuition is that the more important an event e is, the farther the prototype will be pushed away from its original position in the latent space when removing e. As shown in Fig. 1A, the long bars below "great" and "horrible" indicate that they are highly important in prototype `#43`. We also display how well each of the event in a prototype aligns with those in the neighborhood sequences. We compute the alignment score as described in Appendix A and use the height of the rectangle to encode the score, as illustrated in Fig. 4C.

5.4 Visualize Model Difference

Users can understand and analyze the changes after updating the prototypes in the *prototype overview* via comparison mode (***Q7***).

Similar as in some popular code or document version comparison tools, the system highlights the prototypes with different colors to indicate different types of edits in the *prototype overview*. The added, deleted and edited prototypes are marked with ⊕, ⊗, and ⓘ respectively. The visual cues help users keep track of the changes. After users commit the edits, the model is fine-tuned with the updated prototypes. How can we visualize the changes in the model after it is fine-tuned? Directly comparing the model parameters may not be helpful since they can be extremely difficult to interpret. Because in ProSeNet, the model's decisions are made based on the inputs' proximity to the prototypes, we decide to visualize the changing neighborhoods of each prototype to summarize the difference in the model before and after the update.

Given that the prototypes are actually prototypical cases defined by their neighborhood, it is useful to verify the neighborhood changes after fine-tuning. Users can click on the compare button on a historical interaction record in the *editing graph* to activate the comparison mode. The diff-view is then displayed on the right of the *prototype overview* to show the number of instances that flow-in and flow-out from the neighborhood of each prototype as two horizontally-stacked bar-charts (Fig. 4C). The bar on the right indicates the data instances flowing out from the neighborhood of a prototype and the bar on the left indicate the data instances flowing in. The lengths of the bars are proportional to the corresponding number of instances. The colored blocks indicate classification results given by the updated model. As the user hovers over a prototype or a bar (Fig. 4), the visualization displays curved edges connecting the origins and destinations of the incoming and outgoing flows.

Before finalizing to the current design, we have considered several alternative designs. Our first attempt is Sankey diagram (Fig. 4A), which intuitively displays the flow-in and flow-out of the data instances. However, it requires twice the screen space to display the prototypes on both sides of the chart. Another possible approach is using a matrix visualization (Fig. 4B) where the rows and columns represent prototypes of two model snapshots, and the entries in the matrix encode the volume of the flow. However it can be space-inefficient as the incremental changes are usually encoded as a sparse matrix. During the design process, the users criticized that it is difficult to associate the values in the matrix with the prototypes, and they often need to go back and forth between the matrix and the prototype sequences. Our design in comparison organizes the information in two levels: the aggregated volume of flows are displayed as bar-charts and the details of the flow direction are displayed on-demand.

5.5 Editing History

The system visualizes the history of the model edits with as a falling list (an example is shown in Fig. 6A). Each node in the list represents a snapshot of the model. The nodes can be revisited and edited again to create alternative branches. Each node shows a summary of the edits indicating how many prototypes have been added (+), deleted (-) or edited (E) in each snapshot. The editing history provides traceability in the system.

5.6 Sequence Encoder View

The system also visualizes the hidden state changes in the sequence encoder r (Fig. 1D), similar as in Seq2Seq [44]. The goal is to help more experienced users inspect the model's internals and edit the prototypes based on insights gained from the hidden state visualization.

The design is centered around the visual representation of prototypes, similar as in the other views. More specifically, it visualizes the

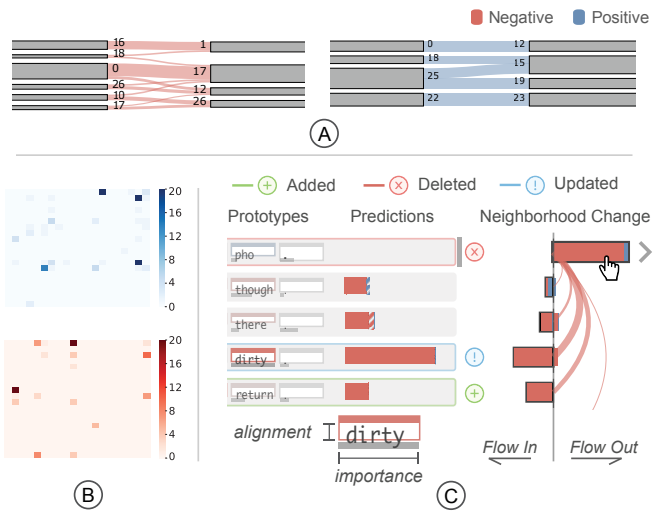

Fig. 4: Alternative designs for visualizing model differences. A: Sankey-like data flow. B: Difference matrix. C: Our current design of aggregated bar-charts with detail-on-demand data flow.

hidden state changes of the prototypes when they pass through the sequence encoder. Since the hidden states are high dimensional vectors, dimension reduction technique is used to project them onto a 2D plane. The projected hidden states of a prototype are connected to form a trajectory. The trajectories help users identify significant changes in the hidden state vector, which usually indicate the occurrences of key events or substructures in a sequence [21] (***Q5***). For example, in Fig. 1D, the word "horrible" pushes the hidden state towards left significantly, indicating that it is a critical word in the sentence.

To visualize the distribution of the prototypes' neighbors we also project their hidden states on the same 2D plane (***R5***). However, projecting such a large amount of data (hidden states of thousands of sequences $\times$ tens of events) would slow down the system (especially for algorithms like t-SNE [30]). Thus, we employ a parametric nonlinear projection technique called kernel t-SNE [14]. The method trains a parametric projection function with a subset of the data (the hidden states of prototype sequences in our implementation). The remaining points are then projected on demand without further optimization using the learned function, which can be completed in linear time.

We design a projection visualization which combines the trajectories of the prototypes and the contour maps which summarize the hidden state distribution of their neighborhoods. This visualization helps identify redundant prototypes whose neighborhood contour maps may have significant overlap with each other (***Q2***). We observe that some projected points are too close on the trajectory and the labels may overlap with each other. We therefore group the points based on their proximity and their labels are displayed collectively near the cluster.

5.7 User Interactions

The system supports a rich set of user interactions to help people refine the model by adding, deleting and revising the prototypes. We have already discussed some of them when introducing the visual components in the system. In this section, we introduce the rest of them.

Direct drag-and-drop. Many of the operations in the system can be completed through direct drag-and-drop. For example, the user can drag-and-drop prototypes to the recycle bin to remove them, to the sequence detail view to inspect the neighborhood instances and to the projection view to analyze the hidden state distributions. The interaction design utilizes a physical metaphor and the users can perform a variety of tasks easily through direct manipulation.

Sequence editor. The system provides a sequence editor (Fig. $5C_2$). The user can directly add and remove events in an existing prototype without retyping the events in a new form.

Sequence query (*Q6*). We provide a search interface to help users create new prototypes with the support from available data. In particular, the user can input a subsequence or phrase q they would like to see in the new prototype. Then, the system will use q to search matched sequences in the database. The system supports two matching schemes: exact and soft matching, which can be used together or separately. In exact matching, each sequence s is scored by $len(LCS(s,q))/len(q)$, where $LCS(\cdot,\cdot)$ computes the longest common subsequence between two inputs. For soft matching, the system first indexes the frequent n-grams in the training sequences with their latent representations encoded by the sequence encoder r. The system searches the n-grams with the closest latent representations to the query q's and returns the sequences containing those n-grams. This allows the users to discover semantically related sequences. An example is displayed in Fig. $5C_1$.

6 Evaluation

In this section, we describe the example usage scenarios and demonstrate how ProtoSteer could be used to steer ProSeNet with user inputs.

6.1 Yelp Review Texts

In this use case, an NLP expert aimed to learn how people typically express their sentiments online. He wanted to build a compact and interpretable representation of the Yelp Reviews [50] with ProSeNet . The dataset contains six million reviews with sentiment labels. Each review is tokenized into a sequence of words. For experimental purpose, the expert created a smaller binary-labeled dataset containing 106 thousand short reviews with less than 25 words. The dataset is further partitioned into 60% training, 20% validation and 20% test set.

However, the expert found it difficult to pick the number of prototypes k that can balance the performance and interpretability at the same time. That is, a larger k tends to yield prototypes with finer-grained semantics and higher accuracy, but the generated prototypes tend to contain more similar and redundant patterns. Thus, the expert started by setting a relatively large k (=70) and trained a ProSeNet with a two-layer bi-directional LSTM encoder. Then he used ProtoSteer to evaluate and condense the model by interactively editing the prototypes. To enrich the sequence visualization, the words are colored based on the frequency of their concurrence with the sentiment labels.

Understand prototypes. At the beginning, the expert scanned through the *prototype list* (Fig. 1A) to see what the prototypes are (***Q1***). With the default similarity-based ordering generated by hierarchical clustering (***Q2***), the expert first noticed a salient clustering of negative and positive outputs along the prediction column. The clustering implies that the model has learned a good measure of sentiment proximity for textual sequences. He proceeded to read a few prototypes, and found that most of the highlighted words in the prototypes are common sentiment indicators including "best", "awesome", "great", "horrible", "worst" and the exclamation mark "!". He then randomly inspected a few prototypes by dragging them into the detail view. The *sequence detail view* (Fig. $1B_1$) shows that the prototype `#61` and most of its neighbors starts with a few compliments similar to "great food" and "great service", and ends with "!" (***Q3***). He also found that some prototypes only have very small weight and some prototypes are very similar. Next, the expert tried to reduce the number and complexity of the prototypes to obtain a simpler and more interpretable model.

Eliminate insignificant prototypes. The expert first focused on eliminating insignificant prototypes. He sorted the prototypes by the number of closest instances (the prediction column) and identified three non-significant prototypes with very few closest instances (*e.g.*, prototypes `#43` and `#64` in Fig. 1A). Then he quickly inspected them one-by-one via the detail view to see their neighbors (***Q3***). All three prototypes do not share a similar pattern with its neighbors. The detail view of prototype `#43` is shown in Fig. $1B_2$ as an example. The discordance between these prototypes and their neighboring sequences suggests they are not good representatives of the data. Due to the insignificance and poor representativeness of these prototypes, the expert removed them from the model by dragging them to the recycle bin resided at the top-left corner of the overview. After that, the expert looks at the weights column and further identifies four prototypes with small weight values,

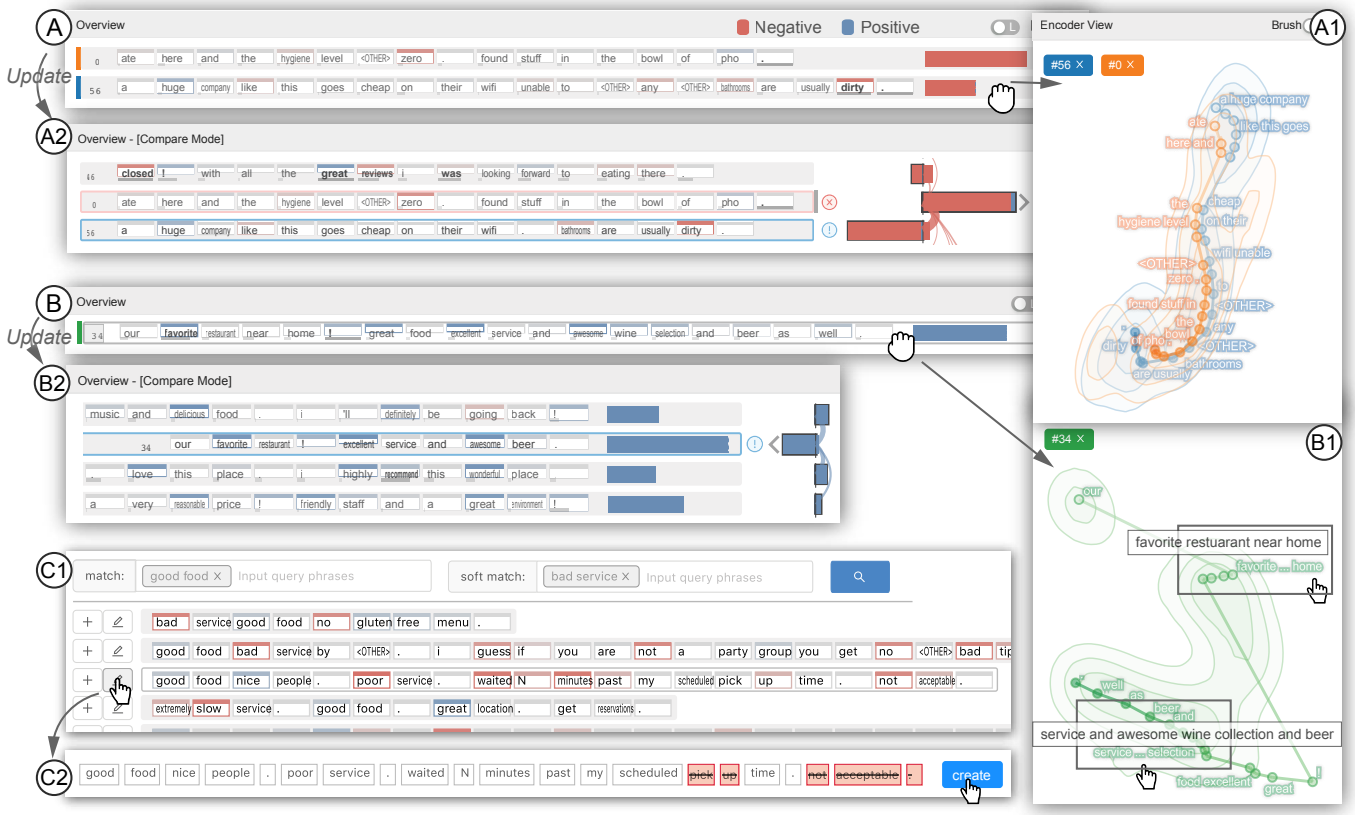

Fig. 5: Steering ProSeNet trained on Yelp Review. A: Evaluate the similarity between two prototypes `#0` and `#56` via *sequence encoder view* (A_1), and reduce the duplicity of them (A_2). B: Reduce the verboseness of a prototype by removing unimportant events and subsequences with little state change (B_1). C_1: Search prototype candidates with key phrases "good food" (exact match) and "bad service" (soft match), and C_2: edit the sequence for cleaner semantics and submit the create operation.

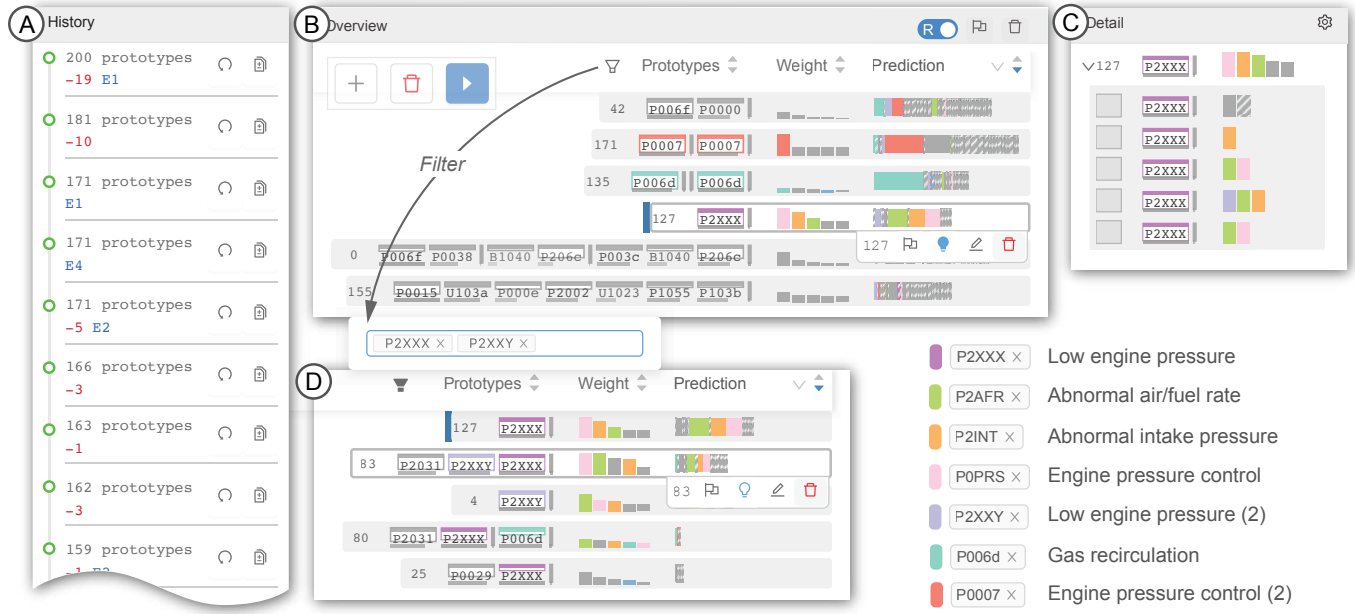

Fig. 6: Pruning and analyzing vehicle fault sequences. A: The interaction history. B: The prototype list sorted by the number of closest instances. Solid fill encodes true positive, and pattern fill indicates false negative. Prototypes `#171` and `#135` present recurring faults of `P0007` and `P006d`. Prototype `#127` indicates an unexpected predicted risk from a single `P2XXX` fault. C: The neighbor instances of `#127` in the validation set. D: Filtering prototypes by faults `P2XXX` and `P2XXY`.

which is also a sign of insignificance. He performed a similar analysis procedure and removed three of them. Finally, the expert clicked the submit button to upload the delete operations. The model is then fine-tuned and the updated performance on test set remains at 94.8%.

Reduce redundancy. Prototypes with similar semantic patterns are undesirable since they bring extra complexity that adds little information to the model or explanation. However, since semantic similarity is often a subjective measure, it is difficult to detect and reduce such redundancy automatically. After sorting the prototypes by similarity (under the model's criteria), the expert quickly discovered a few pairs of prototypes with high similarity scores. He used the *sequence encoder view* (Fig. 5A$_1$) to compare the sequential similarity of prototype `#0` and `#56` and discovered that their neighborhood hidden states' distributions are almost identical (***R5***), which indicates a possible redundancy. After further investigation (Fig. 5A), the expert summarized that both prototypes first complain about the restaurant, and then criticize the dirty environment/food. The expert decided to merge the two prototypes. Thus, he removed prototype `#0` and edited `#56` to emphasize the "dirty" part. After fine-tuning the model for two epochs, the expert verified the change of nearby instances via the model-diff view (***Q7***). As shown in Fig. 5A$_2$, the major flow of data from `#0` to `#56` implies the "merge" of two prototypes is successful.

Improve prototype conciseness. Another issue that the expert discovered is that a lot of prototypes contain wordy phrases that contribute little to the sentiments. The expert wished to simplify each individual prototype to contain only sentimentally significant phrases. Although automatic simplification of the prototypes is feasible, it often produces incomplete sequences that are difficult to read. Via the prototype overview, the expert iteratively selected candidate prototypes that are long and contain possibly insignificant subsequences (the words with short importance bar in the bottom, ***Q5***). He then inspected each of the projected prototype in the *sequence encoder view*. Take prototype `#34` (Fig. 5B$_1$) as an example, he noticed that the words "favorite restaurant near home" are projected very close to each other. The expert recognize that these words only have a small effect on the hidden state (***R5***). The same situation applies to "service and awesome wine collection and beer". Based on these findings, the expert simplified the prototype sequence of 19 words to "our favorite restaurant ! excellent service and awesome beer ." which only contains 10 words (including punctuation). After the edits, he also used the compare view (Fig. 5B$_2$) to examine the change of its neighboring instances, and discovered that there is little outflow of data from `#34`.

Create new prototypes. After sorting the prototypes by accuracy, the expert found that some of the prototypes have a relatively high error rate (***Q4***). He focused on the prototype with the highest error rate, `#44`, and observed that a large portion of the prediction bar is filled with blue striped pattern, indicating a number of instances are wrongly classified as positive. The expert dragged the prototype into the detail view for further analysis. From the detail view, he learned that most wrongly classified neighbor sequences contain a mixture of positive and strong negative sentiment, which is not clearly represented by the prototype `#44` itself. Thus, the expert wished to add a new prototype that reflects the sentiment transition. He clicked the "add" button to open the prototype creator (Fig. 5C$_1$), and queried candidate prototypes (***Q6***) with two phrases "great food" (exact match) and "bad service" (soft match). He selected the third returned sequence, removed some verbose part of the sequence, and clicked "submit" to create a new prototype.

After a few iterations of model updates, the expert performed 26 delete, 2 create and 20 edit operations. The number of prototypes decreases from 70 to 46 and the average length of the prototypes decreased from 18.2 to 14.9. Even though the number and the complexity of prototypes were largely reduced, the performance of the model remains similar (94.8% to 94.9%). In addition, through the interaction with the prototypes, the expert also increased his knowledge about the review data.

6.2 Vehicle Fault Logs

In this usage scenario, an expert in the automotive industry is trying to build a sequence model on vehicle fault logs to predict the future risk of certain faults in each vehicle. With an understanding of the future risks, the mechanics can perform a precautionary examination on the vehicles, which would improve the quality of the maintenance services and lead to higher customer satisfaction.

The dataset contains fault logs of 12,000 vehicles collected during their visit to the repair workshops. Each fault is encoded with a standard five-digit diagnostic trouble code (DTC). A total of 393 different DTCs are used to build a prediction model for the risk of the most frequent 92 DTCs. The expert suggests to group the faults that happen within 14 days as a diagnostics session since most vehicles complete their diagnostics procedure in a workshop within 2 weeks. A special `<SEG>` code is inserted between every two consecutive sessions of DTCs to construct the input sequence. We formulate the risk prediction problem as a multi-label classification task and use the DTCs at the last session as the labels. A ProSeNet with a single-layer Bi-LSTM encoder and 200 prototypes were trained on 60% of the dataset and the recall@5 (the top-five recall rate) [1] on test set is 47.7%. The expert then uses ProtoSteer to analyze the learned prototypes. ProtoSteer provides a color selector to customize the color encoding of events, such that the interface can handle a relatively larger number of events.

Explore prototypical fault sequences. The expert first uses a qualitative color scheme to encode the eight most frequent events and sorted the prototypes by similarity. Going through the list of prototypes (***Q1***), she found that the prediction summary often shares similar colors with some of the events in the input sequence (Fig. 6B). This indicates that these prototypes present recurring patterns where the same set of DTCs appear when the vehicle visits the workshop again. The expert stated that the recurrent DTCs could be faults that are difficult to resolve or insignificant ones that are ignored by the mechanics. Further investigation would be required to differentiate the cause.

Edit Prototypes. With the similarity-based sorting, the expert easily identified some very similar or even duplicate prototypes (***Q2***). The expert quickly went through them and removed 12 redundant prototypes and 7 similar prototypes. After that, she sorted the prototypes according to the number of closest instances, and further condensed the prototype list by removing 10 insignificant prototypes. After fine-tuning, the expert proceeded to simplify another 8 prototypes iteratively by dropping unimportant events in the prototype (***Q5***). The interaction history is shown in Fig. 6A.

Analyze unexpected risks. Next, the expert sorted the prototypes by the number of closest instances to identify the most representative prototypes. After a quick scan, she noticed that the fourth-ranked prototype `#127` shows an interesting unexpected sequential relation: there is a high-risk of three different faults after a vehicle experiences fault `P2XXX` (related to powertrain control). This code indicates the engine pressure sensor detects a low input signal. It is considered to be urgent and indicates conditions that could possibly lead to engine or fuel system damage. Interested in what the code would cause, the expert assigned different colors for the three DTCs: `P0PRS` (engine pressure control), `P2INT` (intake sensor), and `P2AFR` (air/fuel ratio). For detail investigation, the expert then dragged the prototype into the detail view, where more than half of the neighboring sequences developed these faults after `P2XXX`.

Formulate hypothesis. Although they look unrelated at the first glance, the expert further examined these DTCs and found that they all originate from the same subsystem in the powertrain related to air pressure control: `P2AFR` indicates that the air/fuel ratio in the engine is so abnormal that the powertrain control module (PCM) fails to correct it. `P2INT` is related to `P2AFR` because intake sensor is used to help PCM to control the amount of air being allowed to the engine. `P0PRS` usually represents that the pressure captured by the sensor does not meet the expected value. Based on these follow-up diagnoses, the expert further suggested that a low engine pressure (`P2XXX`) is possibly an early sign of these problems but is not properly addressed in previous maintenance sessions. The expert also stated that, though there is indeed a temporal order, `P2XXX` may not necessarily be a cause of these faults but possibly a correlated signal. A human expert is crucial when

[1] See `https://ils.unc.edu/courses/2013_spring/inls509_001/lectures/10-EvaluationMetrics.pdf`

evaluating these patterns to "identify whether this is a causation or correlation". Since in the model the prediction results can be explained through a combination of prototypes it is possible for the domain experts to separate the effect of correlation and causation more easily.

Analyze correlated faults. To gain further understanding of the trouble code `P2XXX`, the expert used the filter function in the overview to filter the prototypes to only four that contains `P2XXX`. She noticed that `P2XXY`, a similar code to `P2XXX` occurs together with `P2XXX` in prototype `#83`. Prototype `#83` has similar prediction results and weights to `#127` (Fig. 6D). The expert added `P2XXY` to the filter and discovered that prototype `#4` (with only `P2XXY`) also share similar predicted risks. This suggests that `P2XXX` and `P2XXY` are more or less equivalent in the system and can potentially be grouped for analysis.

6.3 Expert Interview

We collected feedback from the individual interviews with three NLP researchers and two experts in automotive diagnostics. Two NLP research scientists have ten years' experience in NLP research, and one has five years' experience. The two automotive experts have been developing automotive diagnostics solutions for five years. We first introduced the ProtoSteer interface by walking through simplified versions of the use cases (only the necessary parts to cover the provided functions). Then we asked them to try out the tools by viewing and criticizing the displayed prototype sequences. They were allowed to freely update the prototypes using the interface and commit for fine-tuning. After 30 minutes trial sessions, we collected their suggestions, usage experience, as well as how they will use the tool in their daily work.

Interpretability and Interactivity. The experts showed strong interest in combining interpretability and interactivity. One NLP researcher described that, the most straightforward and also interpretable approach for sentiment analysis uses sentiment dictionaries, which consist of words or phrases with sentiment labels. However, such dictionaries often take a long time to build. *"Your tool has the potential to help build the dictionaries interactively by looking at the prototypes and verify them"*, and this is indeed one goal of designing the system: to enrich both the knowledge of the model and humans. The automotive engineers applauded the idea of working with interpretable prototypes: *"it's much easier to explore and check (the results) than the black boxes"*. They also mentioned that in most cases the diagnostics solutions have to be manually verified and curated before being released to the market.

Domain adaption. Though our goal is to build a generic solution for sequence prediction tasks, we noticed that the experts demonstrated domain-specific suggestions on additional features. The automotive engineers suggested to further integrate static features of the vehicles such as engine types and electronic control unit (ECU) versions in the model. They also demanded for new visualizations to help pinpoint the source of the faults on a vehicle on a system diagram. The NLP researchers asked for additional visual encoding for grammar structures such as part-of-speech or parse trees to help with the prototype updates.

Fine-tuning speed and performance. One NLP researchers said that "the waiting time (for fine-tuning) is a bit long". The other two also agreed with this point after we asked them. They also commented that "this is as expected given the complexity of neural networks", and "smarter (fine-tuning) strategy is indeed an interesting direction". Another interesting discussion among the experts is the trade-off of the performance and model simplicity. We noticed that massive updates performed by the experts were delete operations, which reduce the number of prototypes.

Improvements. Experts from both domains agreed that the visual representation of the prototypes is very straightforward and easy to understand. However, two experts commented that it took a while to fully grasp the "weight" column and the sequence encoder view. One expert suggested that showing an animation from a sequence to its 2d projection could help better explain the relation between the prototypes and the projected sequences for the novice users. Another expert also suggested that it would be beneficial to include detailed explanations of the prediction on individual instances, i.e. how the model combines the prototypes to make the final decision.

7 Limitations and Future Work

Our system as demonstrated has been successfully applied to two real-world datasets with promising results. However, there are still many limitations and we do see quite a few promising research directions:

Scalability to large training dataset. Currently fine-tuning the ProSeNet takes 48 seconds on average for the yelp review dataset (37k sequences with average length of 22) and 4 seconds on average for the vehicle fault logs dataset (7k sequence with average length of 6) on a PC with the same setting. The speed is acceptable for relatively small scale datasets. However, for a larger amount of training data or larger models, the users may need to wait for several minutes and even hours before the model is updated. It requires further research efforts for smarter and more efficient fine-tuning algorithms that addresses the need for real-time interactions. However, one potential workaround is to speed up the algorithm with distributed training, which may require substantial investment in hardware. Another potential approach is to progressively update intermediate training results such that they can observe how the model is adapting with the new prototypes. Strategies like optimistic visualization [32] could also be adopted to provide smoother user experience.

Long-term user study. To fully understand the limitations and the benefits of the system it would be helpful to track the usage of the system through a long-term use study, ideally in a setting with continuously streaming data such that the model needs to be frequently updated to cope with the drift in data distribution.

Termination criteria. The IML loop eventually should come to a stop and a termination criteria is needed. The loop could end when the users believe that the model has reached the right balance between interpretability and predictive performance or there will only be marginal gain when they further interact with the model. However concrete definition of such criteria can be challenging and it is usually application dependent.

Application domains. It could be imagined that a variety of application domains could benefit from the approach presented in this paper. For example, the system can be used to build explainable models for disease risk prediction using EHR data. The doctors can use the system to interactively build a set of prototypes and the model can use these as a reference for prediction. These exemplary cases can also be used as training materials.

8 Conclusion

We introduce a novel human-model interaction framework, ProtoSteer, which directly involves domain experts in steering a deep sequence model without relying on machine learning practitioners as intermediaries. The framework is built on prototype-based reasoning and employs a deep sequence model named ProSeNet which is inherently interpretable. The domain experts can steer the model by directly adding, deleting, or revising the prototypes. An incremental training scheme allows the model to adapt to user specifications while maintaining similar performance. We propose novel interaction and visualization designs to help users inspect, critique and edit the prototypes. Case studies on two real-world application scenarios including sentiment analysis on customer reviews and predictive diagnostics of vehicle faults demonstrate that our approach is able to help the domain experts obtain more interpretable models without sacrificing the predictive performance. We believe that the principles of the proposed approach can be applied to enable steerability for a variety of deep neural network architectures and will open up a wide range of research possibilities in interpretable and steerable machine learning.

Acknowledgments

This research was partially supported by Hong Kong TRS grant T41-709/17N.

REFERENCES

[1] Google analytics. https://analytics.google.com/.

[2] D. Alikaniotis, H. Yannakoudakis, and M. Rei. Automatic text scoring using neural networks. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 715–725, 2016.

[3] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35(4):105–120, 2014.

[4] S. Amershi, J. Fogarty, A. Kapoor, and D. Tan. Effective end-user interaction with machine learning. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[5] S. Amershi, D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. Bennett, K. Inkpen, J. Teevan, R. Kikin-Gil, and E. Horvitz. Guidelines for human-ai interaction. ACM, May 2019.

[6] M. Ankerst, C. Elsen, M. Ester, and H.-P. Kriegel. Visual classification: an interactive approach to decision tree construction. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 99 of *KDD '99*, pp. 392–396. ACM, 1999.

[7] N. Boukhelifa, A. Bezerianos, and E. Lutton. Evaluation of interactive machine learning systems. In *Human and Machine Learning*, pp. 341–360. Springer, 2018.

[8] D. Cashman, G. Patterson, A. Mosca, N. Watts, S. Robinson, and R. Chang. Rnnbow: Visualizing learning via backpropagation gradients in rnns. *IEEE Computer Graphics and Applications*, 38(6):39–50, Nov 2018. doi: 10.1109/MCG.2018.2878902

[9] Y. Chen, P. Xu, and L. Ren. Sequence synopsis: Optimize visual summary of temporal event data. *IEEE transactions on visualization and computer graphics*, 24(1):45–55, 2018.

[10] F. Du, B. Shneiderman, C. Plaisant, S. Malik, and A. Perer. Coping with volume and variety in temporal event sequences: Strategies for sharpening analytic focus. *IEEE transactions on visualization and computer graphics*, 23(6):1636–1649, 2017.

[11] J. J. Dudley and P. O. Kristensson. A review of user interface design for interactive machine learning. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 8(2):8, 2018.

[12] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998.

[13] J. A. Fails and D. R. Olsen, Jr. Interactive machine learning. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, IUI '03, pp. 39–45. ACM, New York, NY, USA, 2003. doi: 10.1145/604045.604056

[14] A. Gisbrecht, A. Schulz, and B. Hammer. Parametric nonlinear dimensionality reduction using kernel t-sne. *Neurocomputing*, 147:71 – 82, 2015. Advances in Self-Organizing Maps Subtitle of the special issue: Selected Papers from the Workshop on Self-Organizing Maps 2012 (WSOM 2012). doi: 10.1016/j.neucom.2013.11.045

[15] D. Gotz and H. Stavropoulos. Decisionflow: Visual analytics for high-dimensional temporal event sequence data. *IEEE transactions on visualization and computer graphics*, 20(12):1783–1792, 2014.

[16] S. Guo, Z. Jin, D. Gotz, F. Du, H. Zha, and N. Cao. Visual progression analysis of event sequence data. *IEEE transactions on visualization and computer graphics*, 25(1):417–426, 2019.

[17] S. Guo, K. Xu, R. Zhao, D. Gotz, H. Zha, and N. Cao. Eventthread: Visual summarization and stage analysis of event sequence data. *IEEE transactions on visualization and computer graphics*, 24(1):56–65, 2018.

[18] F. M. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE transactions on visualization and computer graphics*, 2018.

[19] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau. Activis: Visual exploration of industry-scale deep neural network models. *IEEE transactions on visualization and computer graphics*, 24(1):88–97, 2018.

[20] A. Kapoor, B. Lee, D. Tan, and E. Horvitz. Interactive optimization for steering machine classification. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1343–1352. ACM, 2010.

[21] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[22] J. L. Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6(1):3–34, 1992.

[23] B. C. Kwon, M.-J. Choi, J. T. Kim, E. Choi, Y. B. Kim, S. Kwon, J. Sun, and J. Choo. Retainvis: Visual analytics with interpretable and interactive recurrent neural networks on electronic medical records. *IEEE transactions on visualization and computer graphics*, 25(1):299–309, 2019.

[24] B. C. Kwon, J. Verma, and A. Perer. Peekquence: Visual analytics for event sequence data. In *ACM SIGKDD 2016 Workshop on Interactive Data Exploration and Analytics*, 2016.

[25] J. Li, X. Chen, E. Hovy, and D. Jurafsky. Visualizing and understanding neural models in NLP. In *Proc. NAACL: HLT*, pp. 681–691. Association for Computational Linguistics, 2016.

[26] O. Li, H. Liu, C. Chen, and C. Rudin. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In *AAAI Conference on Artificial Intelligence*, 2018.

[27] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[28] S. Liu, Z. Li, T. Li, V. Srikumar, V. Pascucci, and P.-T. Bremer. Nlize: A perturbation-driven visual interrogation tool for analyzing and interpreting natural language inference models. *IEEE transactions on visualization and computer graphics*, 25(1):651–660, 2019.

[29] Z. Liu, Y. Wang, M. Dontcheva, M. Hoffman, S. Walker, and A. Wilson. Patterns and sequences: Interactive exploration of clickstreams to understand common visitor paths. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):321–330, 2017.

[30] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[31] Y. Ming, P. Xu, H. Qu, and L. Ren. Interpretable and steerable sequence learning via prototypes. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '19, 2019.

[32] D. Moritz, D. Fisher, B. Ding, and C. Wang. Trust, but verify: Optimistic visualizations of approximate queries for exploring big data. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pp. 2904–2915. ACM, New York, NY, USA, 2017. doi: 10.1145/3025453.3025456

[33] W. J. Murdoch, P. J. Liu, and B. Yu. Beyond word importance: Contextual decomposition to extract interactions from LSTMs. In *International Conference on Learning Representations*, 2018.

[34] W. J. Murdoch and A. Szlam. Automatic rule extraction from long short term memory networks. 2017.

[35] Parliament and C. of the European Union. The general data protection regulation. 2016.

[36] A. Perer and D. Gotz. Data-driven exploration of care plans for patients. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pp. 439–444. ACM, 2013.

[37] A. Perer and F. Wang. Frequence: interactive mining and visualization of temporal frequent event sequences. In *Proceedings of the 19th international conference on Intelligent User Interfaces*, pp. 153–162. ACM, 2014.

[38] P. J. Polack, S.-T. Chen, M. Kahng, M. Sharmin, and D. H. Chau. Timestitch: Interactive multi-focus cohort discovery and comparison. In *Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on*, pp. 209–210. IEEE, 2015.

[39] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144. ACM, 2016.

[40] M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI Conference on Artificial Intelligence*, 2018.

[41] E. Rich and K. Knight. *Artificial intelligence*. Tata McGraw-Hill, 1991.

[42] D. Sacha, M. Kraus, D. A. Keim, and M. Chen. Vis4ml: An ontology for visual analytics assisted machine learning. *IEEE transactions on visualization and computer graphics*, 25(1):385–395, 2019.

[43] J. Stasko and E. Zhang. Focus+ context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pp. 57–65. IEEE, 2000.

[44] H. Strobelt, S. Gehrmann, M. Behrisch, A. Perer, H. Pfister, and A. M. Rush. Seq2seq-vis: A visual debugging tool for sequence-to-sequence models. *IEEE transactions on visualization and computer graphics*, 25(1):353–363, 2019.

[45] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):667–676, 2018.

[46] J. Talbot, B. Lee, A. Kapoor, and D. S. Tan. Ensemblematrix: interactive visualization to support machine learning with multiple classifiers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1283–1292. ACM, 2009.

[47] S. Van Den Elzen and J. J. van Wijk. Baobabview: Interactive construction and analysis of decision trees. In *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 151–160. IEEE, 2011.

[48] G. Wang, X. Zhang, S. Tang, H. Zheng, and B. Y. Zhao. Unsupervised clickstream clustering for user behavior analysis. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 225–236. ACM, 2016.

[49] K. Wongsuphasawat and D. Gotz. Exploring flow, factors, and outcomes of temporal event sequences with the outflow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2659–2668, Dec 2012.

[50] Yelp dataset challenge, 2018. Available: `https://www.yelp.com/dataset/challenge`, [Accessed: Nov 1, 2018].

[51] J. Zhao, Z. Liu, M. Dontcheva, A. Hertzmann, and A. Wilson. Matrixwave: Visual comparison of event sequence data. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 259–268. ACM, 2015.

A Event Alignment

When analyzing and interacting with prototype sequences, it is also beneficial to assess the consistence between a prototype and its neighbor instances. This is also the interest of some domain experts as well: "How do I know if the model is getting the most representative prototype from a group of sequences?". The sequence encoder view projects sequences as trajectories to allow users inspect the similarity between the encoded sequential states between two sequences. However, it is still difficult to know whether each individual event presented in the prototype is representative or not. The alignment of each event can be useful to help the user identify poorly representative events and improve the consistency between the prototype and its neighbors.

For a prototype sequence $\{a^{(t)}\}$ and its neighboring sequence $\{b^{(t)}\}$, we aim to measure the alignment between each pair of events $a^{(i)}$ and $b^{(j)}$. We combine the sequential similarity and event similarity and compute the alignment as:

$$\alpha(a^{(i)}, b^{(j)}) = sim_h(a^{(i)}, b^{(j)}) \cdot sim_e(a^{(i)}, b^{(j)}), \tag{1}$$

where sim_h is the similarity function used in the ProSeNet: $\exp(-dist^2(h^{(i)}_{\{a\}}, h^{(j)}_{\{b\}}))$, and sim_e computes the similarity between two events. sim_h can be interpreted as the contextual similarity of two events within specific sequences. For example, the word "great" in "great service" and "not so great" should not be treated as if they are the same. The additional sim_e helps fine-grained alignment by exploiting event similarity. For example, the encoded state sequence of "great food" and "pizza is excellent" may align well at both ends of the two phrases ("great" to "pizza", "food" to "excellent"). However, we expect "great" to align with "excellent", and "food" to align with "pizza". The event similarity measure sim_e can be provided by the user to reflect their interests, or computed based on the event embedding as learned by the model.

B Prototype Sequence Network (ProSeNet)

The prototype sequence network (ProSeNet) is a work **in progress**. Due to page limit, we briefly introduce the architecture of ProSeNet, formulate the learning objective and describe the training procedure here in the appendix.

B.1 ProSeNet Architecture

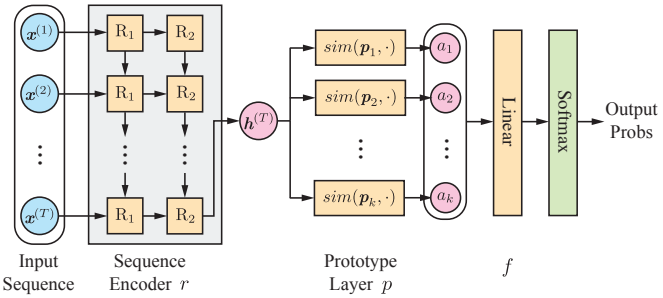

Fig. 7: The architecture of our proposed ProSeNet model. The model consists of three parts, the recurrent sequence encoder network r, the prototype layer p that contains k prototypes, the fully connected layer f, and a softmax layer for output probabilities in multi-class classification tasks.

Let $\mathscr{D} = \{((\boldsymbol{x}^{(t)})_{t=1}^{T}, y)\}$ be a labeled sequence dataset, where T is the sequence length, $\boldsymbol{x}^{(t)} \in \mathbb{R}^n$ is the input vector at step t, and $y \in \{1, \ldots, C\}$ is the label of the sequence. We aim to learn representative prototype sequences (not necessarily exist in the training data) that can be used as classification references and analogical explanations. For a new input sequence, its similarities with each representative sequences are measured in the learned latent space. Then, the prediction of the new instance can be derived and explained by its similar prototype sequences.

The basic architecture of ProSeNet is similar to the one proposed by Li *et al.* [26]. As shown in Fig. 7, the model consists of three components: a sequence encoder r, a prototype layer p, and a fully connected layer f.

For a given input sequence $(\boldsymbol{x}^{(t)})_{t=1}^{T}$, the sequence encoder r maps the entire sequence into a single embedding vector with fixed length $\boldsymbol{e} = r((\boldsymbol{x}^{(t)})_{t=1}^{T})$, $\boldsymbol{e} \in \mathbb{R}^m$. The encoder could be any backbone sequence learning models *e.g.*, LSTM, Bidirectional LSTM (Bi-LSTM) or GRU. In our experiments, the hidden state at the last step, $\boldsymbol{h}^{(T)}$, is used as the embedding vector.

The prototype layer p contains k prototype vectors $\boldsymbol{p}_i \in \mathbb{R}^m$, which have the same length as $\boldsymbol{e}$. The layer scores the similarity between $\boldsymbol{e}$ and each prototype $\boldsymbol{p}_i$. In previous work [26], the squared L_2 distance, $d_i^2 = \|\boldsymbol{e} - \boldsymbol{p}_i\|_2^2$, is directly used as the output of the layer. To improve interpretability, we compute the similarity using:

$$a_i = \exp(-d_i^2),$$

which converts the distance to a score between 0 and 1. Zero can be interpreted as the sequence embedding $\boldsymbol{e}$ being completely different from the prototype vector $\boldsymbol{p}_i$, and one means they are identical.

With the computed similarity vector $\boldsymbol{a} = p(\boldsymbol{e})$, the fully connected layer computes $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{a}$, where $\boldsymbol{W}$ is a $C \times k$ weight matrix and C is the output size (*i.e.*, the number of classes in classification tasks). To enhance interpretability, we constrain $\boldsymbol{W}$ to be non-negative. For multi-class classification tasks, a softmax layer is used to compute the predicted probability: $\hat{\boldsymbol{y}}_i = \exp(z_i) / \sum_{j=1}^{C} \exp(z_j)$.

B.2 Learning Objective

Our goal is to learn a ProSeNet that is both accurate and interpretable. For accuracy, we minimize the cross-entropy loss on training set:

$$CE(\Theta, \mathscr{D}) = \sum_{((\boldsymbol{x}^{(t)})_{t=1}^{T}, \boldsymbol{y}) \in \mathscr{D}} \boldsymbol{y} \log(\hat{\boldsymbol{y}}) + (1 - \boldsymbol{y}) \log(1 - \hat{\boldsymbol{y}}),$$

where Θ is the set of all trainable parameters of the model.

Diversity. In our experiments, we found that when the number of prototypes k is large (*i.e.*, over two or three times the number of classes), the training would often result in a number of similar or even duplicate prototypes (*i.e.*, some prototypes are very close to each other in the latent space). It would be confusing to have multiple similar prototypes in the explanations and also inefficient in utilizing model parameters. We prevent such phenomenon through a diversity regularization term that penalizes on prototypes that are close to each other:

$$R_d(\Theta) = \sum_{i=1}^{k} \sum_{j=i+1}^{k} \max\left(0, d_{min} - \|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2\right)^2,$$

where d_{min} is a threshold that classifies whether two prototypes are close or not. We set d_{min} to 1.0 or 2.0 in our experiments. R_d is a soft regularization that exerts a larger penalty on smaller pairwise distances. By keeping prototypes distributed in the latent space, it also helps produce a sparser similarity vector $\boldsymbol{a}$.

Sparsity and non-negativity. In addition, to further enhance interpretability, we add L_1 penalty on the fully connected layer f, and constrain the weight matrix $\boldsymbol{W}$ to be non-negative. The L_1 sparsity penalty and non-negative constraints on f help to learn sequence prototypes that have more unitary and additive semantics for classification.

Clustering and evidence regularization. To improve interpretability, Li *et al.* [26] also proposed two regularization terms to be jointly minimized, the clustering regularization R_c and the evidence regularization R_e. R_c encourages a clustering structure in the latent space by minimizing the squared distance between an encoded instance and its closest prototype:

$$R_c(\Theta, \mathscr{D}) = \sum_{(\boldsymbol{x}^{(t)})_{t=1}^{T} \in \mathscr{X}} \min_{i=1}^{k} \left\| r\left((\boldsymbol{x}^{(t)})_{t=1}^{T}\right) - \boldsymbol{p}_i \right\|_2^2,$$

where $\mathscr{X}$ is the set of all sequences in the training set $\mathscr{D}$. The evidence regularization R_e encourages each prototype vector to be as close to an encoded instance as possible:

$$R_e(\Theta, \mathscr{D}) = \sum_{i=1}^{k} \min_{(\boldsymbol{x}^{(t)})_{t=1}^{T} \in \mathscr{X}} \left\| \boldsymbol{p}_i - r\left((\boldsymbol{x}^{(t)})_{t=1}^{T} \right) \right\|_2^2.$$

Full objective. To summarize, the loss function that we are minimizing is as:

$$\begin{aligned} Loss(\Theta, \mathscr{D}) = CE(\Theta, \mathscr{D}) + \lambda_c R_c(\Theta, \mathscr{D}) + \lambda_e R_e(\Theta, \mathscr{D}) \\ + \lambda_d R_d(\Theta, \mathscr{D}) + \lambda_{l_1} \|\boldsymbol{W}\|_1, \end{aligned} \tag{2}$$

where λ_c, λ_e, λ_d and λ_{l_1} are hyperparameters that control the strength of the regularizations. The configuration of these hyperparameters largely depends on the nature of the data and can be selected through cross-validation. For each experiment in Section 4, we provide the hyperparameter settings.

B.3 Optimizing the Objective

We use stochastic gradient descent (SGD) with mini-batch to minimize the loss function on training data. Since the gradient of R_e requires the computation on the whole training set, we relax the minimization to be only computed in every single batch. In this section, we mainly discuss the prototype projection technique that we used to learn simple and interpretable prototypes. The optimization procedure iteratively alternates between the SGD and the prototype projection steps.

Prototype projection. Since the prototype vectors $\boldsymbol{p}_i$ are representations in the latent space, they are not readily interpretable. Li *et al.* [26] proposed to jointly train a decoder that translates the latent space to the original input sequence space and thus making prototypes interpretable. However, the decoder may not necessarily decode prototypes to meaningful sequences. We propose a projection step during training that assigns the prototype vectors with their closest sequence embedding in the training set:

$$\boldsymbol{p}_i \leftarrow \arg\min_{\boldsymbol{e} \in r(\mathscr{X})} \|\boldsymbol{e} - \boldsymbol{p}_i\|_2. \tag{3}$$

Each prototype vector $\boldsymbol{p}_i$ is then associated with a *prototype sequence* in the input space. The projection step is only performed every few training epochs (we set to 4 in our experiments) to reduce computational cost. Compared with the original prototype network [26], the projection step saves the efforts of jointly training a sequence auto-encoder, which is computationally expensive. It also assures each prototype to be an observed sequence, which guarantees that the prototypes are meaningful in the real world.

Prototype simplification. Although the prototypes are already readable after projecting to observed sequences in the training data, it may still be difficult to comprehend a prototype sequence if it contains insignificant or irrelevant noisy events.

Next, we introduce a procedure to simplify the projected prototype sequences. That is, instead of projecting a prototype to a complete observed sequence, we project it to a subsequence containing the critical events. The projection step (Equation 3) now becomes:

$$\begin{aligned} \boldsymbol{p}_i &\leftarrow r(seq_i), \\ seq_i &= \arg\min_{seq \in sub(\mathscr{X})} \left(\|r(seq) - \boldsymbol{p}_i\|_2 \right), \end{aligned} \tag{4}$$

where $sub(\mathscr{X})$ is the set of all possible subsequences of the data in $\mathscr{X}$, $|\cdot|$ computes the effective length of the subsequence. Note that the complexity of the above operation is $O(2^T N)$, where N is the size of training set and T is the maximum length of the sequences in $\mathscr{X}$. The cost of the brute-force computation grows exponentially with T, which is unacceptable even for relatively short sequences.

We use beam search to find an approximate solution [41]. Beam search is a greedy breadth-first search algorithm which only keeps w best candidates in each iteration. w is called the beam width. The algorithm first selects w closest candidate sequences to prototype $\boldsymbol{p}_i$. Then it generates all the possible subsequences which can be obtained by removing one event from any of the w candidates. The score in Equation 4 is calculated for each subsequence. The w subsequences with the minimum scores are then kept as candidates to continue the search in the next iteration. The subsequence with the minimum score is the output. The complexity of the algorithm is now $O(w \cdot T^2 N)$. We use $w = 3$ in our experiments.

B.4 Refining ProSeNet with User Knowledge

Next we discuss how users can refine a ProSeNet for better interpretability and performance by validating and updating the prototypes, esp. when they have certain expertise or knowledge in the problem domain. Allowing users to validate and interact with the prototypes can also increase their understanding of the model and the data, which is the foundation of user trust [39].

We assume that the knowledge of a user can be explicitly expressed in the form of input-output patterns which the user recognizes as significant or typical in the domain (*e.g.*, "food is good" is typically a review with "positive" sentiment). These patterns can be regarded as the "prototypes" that the user learned from his/her past experiences. The refinement can thus be done by incorporating user-specified prototypes as constraints in the model.

Based on the users' past knowledge and observation on the model outputs, there are three types of possible operations that they can apply to the model: *create* new prototypes, *revise* or *delete* existing prototypes. After changes are committed, the model is fine-tuned on the training data to reflect the change.

When fine-tuning the model the prototypes should be fixed to reflect the users' constraints. Therefore we make the following revisions to the optimization process described in Section 3.3: 1) instead of updating the latent prototype vectors $\boldsymbol{p}_i$ in the gradient descent step, we use the updated sequence encoder r in each iteration to directly set $\boldsymbol{p}_i = r(seq_i)$; 2) the prototype projection step is skipped. After fine-tuning, the sequence encoder r learns better representations of the data. The user can verify the updated results and repeat the process until he/she is satisfied with the result.