Sharing and Automation for
Privacy Preserving Attack Neutralization

(H2020 833418)

D3.8 Demonstrator of Visual Support for Designing Detection Models, Initial Version (M21)

Published by the SAPPAN Consortium

Dissemination Level: Public

H2020-SU-ICT-2018-2020 – Cybersecurity

Document control page

Document file: D3.8
Document version: 1.0
Document owner: Franziska Becker (USTUTT)

Work package: WP3
Task: T3.5 Visualization support for the design of attack and anomaly detection models
Deliverable type: Demonstrator
Delivery month: M21
Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document History:

Version	Author(s)	Date	Summary of changes made
0.1	Franziska Becker (USTUTT)	05.10.2020	Added initial outline and basic descriptions.
0.2	Franziska Becker (USTUTT)	11.01.2021	Adjusted outline and text.
0.3	Franziska Becker (USTUTT)	27.01.2021	Imported word document content for review.
1.0	Franziska Becker (USTUTT)	29.01.2021	Incorporated review comments.

Internal review history:

Reviewed by	Date	Summary of comments
Arthur Drichel (RWTH)	28.01.2021	Fixed typos and minor grammar issues; minor wording changes; fixed figure numbering;

Executive Summary

This deliverable describes the first iteration of two visual analytics systems developed in the scope of task T3.5 “Visualization support for the design of attack and anomaly detection models”. First, the context of these systems in SAPPAN is discussed. Then, each of the two systems is described in detail. The first system is situated in the area of explainable artificial intelligence, as it aims to support developers of deep learning models to better understand their model results and inner workings. Its starting point is the SAPPAN use case for domain generation algorithms and the deep learning models developed for it. The complete system, including all its visualizations, interactions and implementation details is then described, followed by the second visual analytics system for the visualization of host profiles. This system illustrates network behavior and allows its users to evaluate the performance of machine learning models that perform host classification. Finally, the deliverable discusses future work that may be performed for the final iteration of these systems, including evaluation strategies and further extensions of both systems.

Contents

Executive Summary **3**

1 Introduction **5**

2 SAPPAN Context **5**

3 Visualization of Deep Learning Models for DGA Classification **5**

3.1 Explainable Artificial Intelligence (XAI) 5

3.2 Visualization Concept 7

3.2.1 Overview Visualization 8

3.2.2 Analysis Visualizations 10

3.2.3 Interaction Concept 13

3.2.4 Implementation Details 14

4 Visualization of Host Profiles **17**

4.1 Visualization Concept 17

4.1.1 The Overview Dashboard 18

4.1.2 The Detail View 19

4.1.3 The Comparison Screen 19

4.2 Implementation 20

5 Future Work **21**

5.1 DGA Model Visualization 21

5.1.1 Uncertainty and Collaborative Learning 21

5.1.2 Sensemaking and Reasoning 21

5.1.3 Evaluation 22

5.2 Host Profile Classifier Verification 24

6 Summary **24**

References **24**

1 Introduction

This deliverable documents the work performed towards completing the initial part of task T3.5 "Visualization support for the design of attack and anomaly detection models". In the first section, we discuss the context of this task in the SAPPAN project. Then, we present two visualization systems developed for this deliverable. The first system aims to support designers of deep learning models to analyze their model's behavior and performance, while the second system lets network experts explore host profiles generated from network data.

Any results described in this deliverable detail initial versions that may not be complete and can be subject to change in the future.

2 SAPPAN Context

In the context of SAPPAN, the efforts described in this deliverable can be seen as part of local detection. Local detection is often carried out cooperatively by automated algorithms and trained models while handling is performed by human operators, such as experts in security operation centers. Before automatic solutions can be deployed to real-life scenarios, they must be thoroughly tested and understood. The visual analytics systems developed for this part of the project aim to assist that process in two different areas of application. First, related to the SAPPAN use case #2 described in deliverable D3.4 “Algorithms for analysis of cybersecurity data”, we designed and implemented a system to support designers of deep learning models assess the performance and behavior of their models after the training phase. Second, we created a system to analyze the state of a network as well as host profiles predicted by a model. Since these visual analytics systems are not intended to be utilized by end-users such as security experts, both were implemented as stand-alone applications independent of the dashboard developed in work package 6.

3 Visualization of Deep Learning Models for DGA Classification

The overall objective for the related task is to support designers with the development of machine learning models. In general, support can be offered in regards to the goodness of the model (i.e. its performance) or the designer’s understanding of the model. The former may be achieved through an interactive tool to rapidly test and compare different hyper-parameters for the same model architecture while the latter could be accomplished with a post-training comparison between different models and their performance related to features in the data. For this task, we chose the latter approach, situating our visualization concept in the domain of explainable artificial intelligence.

3.1 Explainable Artificial Intelligence (XAI)

In order to design better models, it is imperative to understand how and why they come to a specific result. Often, machine learning models are divided into those that are explainable, such as shallow decision trees, and those that are not, such as deep learning models. To complicate matters, explainability of artificial intelligence models

has been found to inversely correlate with learning performance [1]. This makes apparent the need and rising popularity of the area of explainable artificial intelligence, which as an interdisciplinary study field aims to improve models' intrinsic explainability and develop algorithms or visualizations to interpret models. Unsurprisingly, the interest in this area and the number of publications has dramatically increased in recent years, as depicted in Figure 1 from Arrieta and colleagues [2].

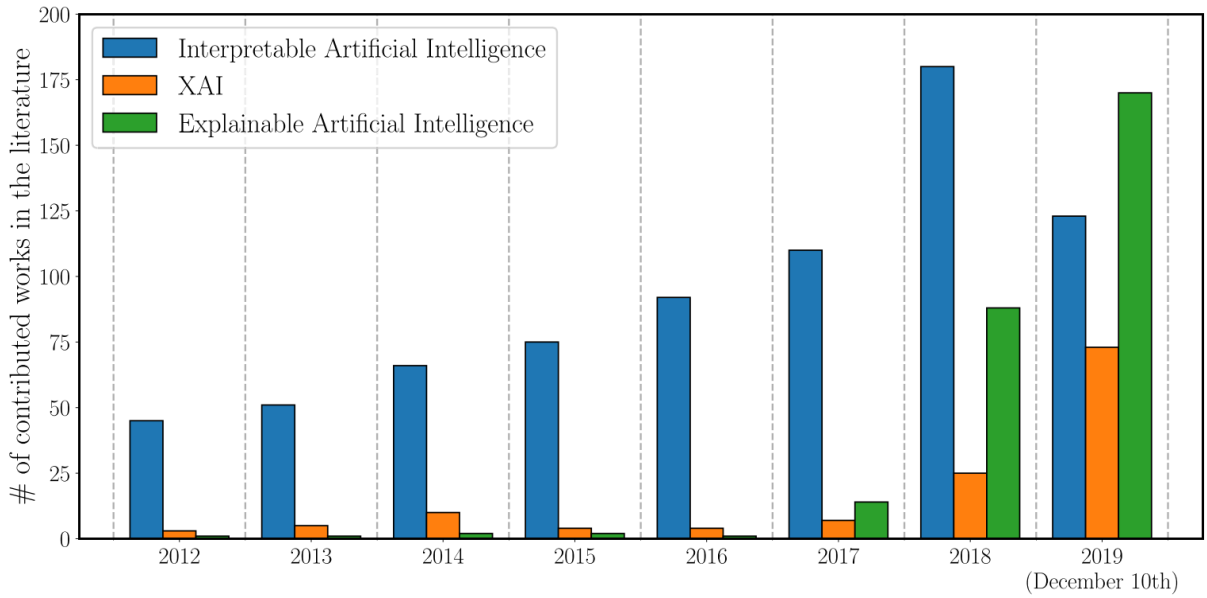

Figure 1: Combined number of publications related to artificial intelligence and explainability in recent years, taken from Arrieta [2].

Prominent methods developed in this field belong to the broader category of attribution, that use the gradient to, for example, find the pixels in an image that most contribute to the model's classification of that image. This can be visualized using the size of the pixels to indicate their influence or by overlaying a heat map (see Figure 2).

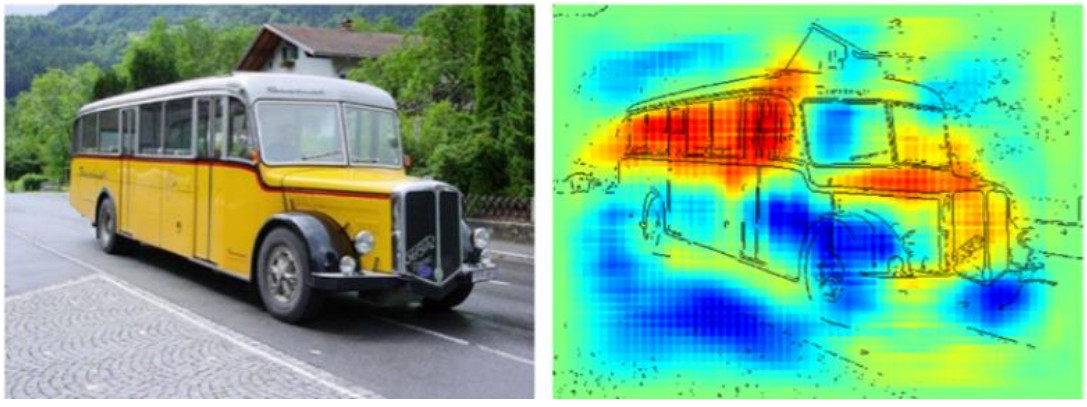

Figure 2: Attribution for a convolution model using layer-wise relevance propagation, visualized as a heat map. Red areas contribute positively to the classification 'bus' while blue pixels can be interpreted as evidence against the classification 'bus'. Image taken from [3].

Another type of visualization named 'feature visualization' (ref. Figure 3) that is exclusive to convolutional models uses optimization to create a prototypical image for a

specific filter that maximizes the activation for that filter. In this way, it was found that convolutional filters in early layers often act as detectors for simple shapes like edges.

In [4], Olah and colleagues use feature visualization to show what a particular convolutional layer *sees* in an image that represents the class 'Golden Labrador' by maximizing the dot product of the original activation with that of the activation of each filter at each location of the input image (see figure Figure 3). This allows us to get an idea of what a specific convolutional layer makes of the image.

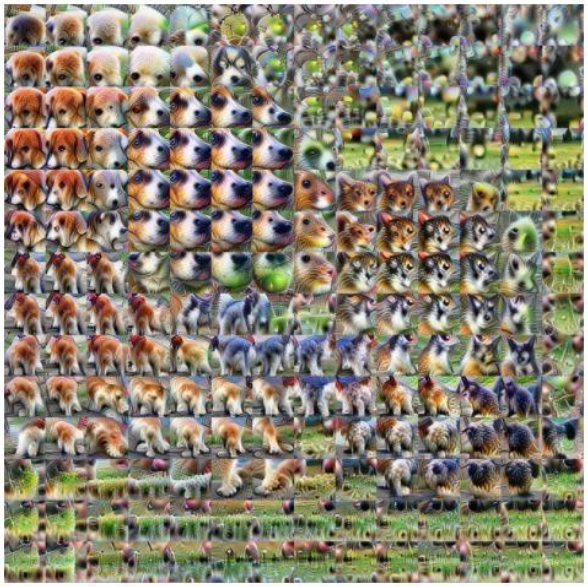

Figure 3: Feature visualization for a single layer in a convolutional model, optimized for the 'Golden Labrador' class. Taken from [4].

In the field of visualization and visual analytics, many different systems have been developed to examine machine learning models and their decisions. Most publications visualize the model after training, but some have developed interactive learning systems where the user can inspect the intermediate the model performance during training and even steer further training by resolving errors such as false labels or adjusting the model's architecture. A large portion of related work considers only a specific type of model, with convolutional models being the most prevalent architecture, most likely due to images' intrinsic interpretability. In a similar vein to our concept, Rauber and colleagues [5] calculate 2D projections of the model's activation of fully connected layers, showing how the projection changes over the training epochs. However, they do not investigate all layers of the model, do not support a large number of classes and do not provide any explanation beyond the projection itself.

3.2 Visualization Concept

One goal of this task is to develop a visual analytics system that provides designers of deep learning models with the tools to better understand how their classification models make decisions. We chose the deep learning models for domain generation algorithms (DGAs) from the DGA use case as our application case, since this was the most advanced SAPPAN use case at the start of development. In order to formulate require-

ments for our system, we scanned related literature [6, 7, 8] and consulted with machine learning experts from partners at RWTH Aachen in online meetings and via email. The resulting requirements are based on three pillars:

1. Compare results of different models
2. Improve the understanding of model results
3. Visualize inner workings of the model

Given the application case of domain generation algorithms, some restrictions are laid upon any possible visualizations. For one, the data of interest is non-natural text data, meaning that any depiction of the data itself must go beyond a pure display of it, as is the case for image data prevalent in many XAI visualizations. Another challenge comes from the number of existing classes of DGAs, which with 92 at the time of writing is much larger than for the majority of classification datasets according to a survey in [9]. These restrictions prompted us to show the model and its training data on a global and a local level respectively. This allows the user to get a complete view of model performance, whilst the inspection of local subsets also enables us to apply techniques that do not scale sufficiently for larger sets.

3.2.1 Overview Visualization

For the global overview, we created a visualization that communicates per-class performance on a detailed level in combination with two feature distributions in a histogram matrix (see Figure 4). The histogram matrix has a row for each class in the dataset, with the name being rendered transparently in the background of the row, and has three columns in total.

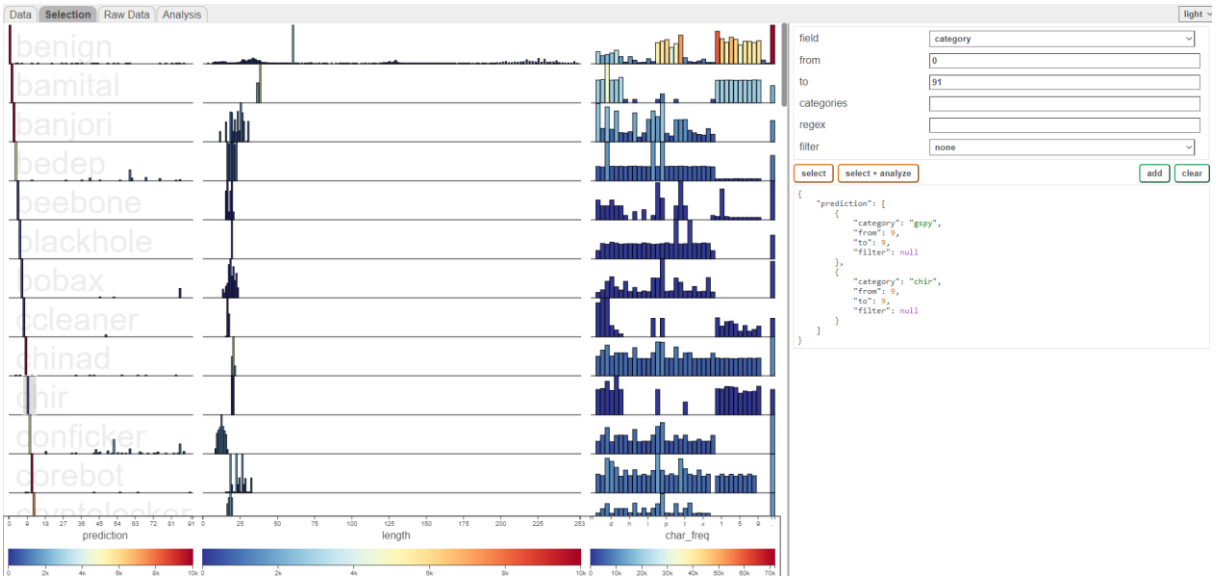

Figure 4: Overview visualization with a histogram matrix (left), selection form field (upper right), and selection representation (lower right).

In the first column, a histogram of all predicted classes is drawn which visualizes what common performance metrics compute. The x-axis indicates the class via its index and the y-axis indicates the number of predictions for the class of this row. Thus, the per-class accuracy can be judged by comparing the size of the bar at a class' correct index with all other bars in the row. In order to let the user estimate the overall classifier performance, we color the bars according to a global color map. To provide better

awareness of class membership, each row in the histogram matrix also displays the class name in the background with reduced opacity.

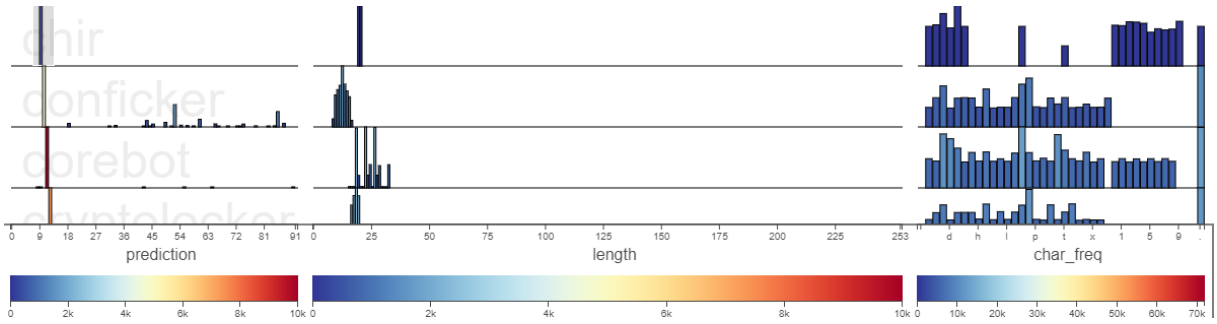

Figure 5: Close-up view of some rows of the histogram matrix.

The second column shows the discrete length distribution of all instances in a class. Since it features the largest number of distinct values and we want to avoid aggregating or binning the data, the width of this column is double the width of both other columns. This enables us to show a bar for each value on the x-axis such that the differences in height and color can still be perceived. In the third column, we show the distribution of valid characters for the domains of that class. In each row, the height of the bars is scaled relative to the maximum value of that class (local scale) and the color is chosen based on a color map that uses the maximum value of all classes (global scale).

The secondary function of this visualization is to define a selection to analyze in the next view. This is achieved via brushing (ref. Figure 5), which can be performed locally in a row or globally on any of the three axes. This brushing acts an as inclusive disjunction, meaning that any column in any row can be brushed, which will add anything that is contained in the brushed region to the selection. For example, given the row for the class ‘conficker’, the user can brush the bar for the value ‘10’ in the prediction column and the bars for the values from 10 to 15 in the length column. As a consequence, the selection is made up of instances that belong to the class ‘conficker’ and are predicted to belong to ‘conficker’ *or* have a domain length between 10 and 15.


```
{
    "prediction": [
        {
            "category": "gspy",
            "from": 9,
            "to": 9,
            "filter": null
        },
        {
            "category": "chir",
            "from": 9,
            "to": 9,
            "filter": null
        }
    ]
}
```


Figure 6: JSON representation of the current selection.

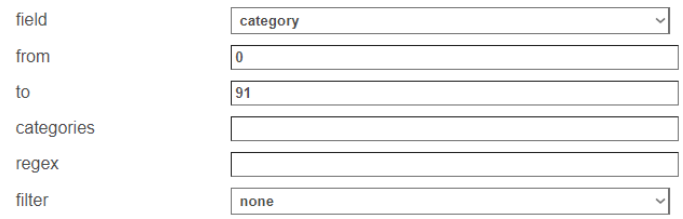

Figure 7: Selection form to let the user define a complex selection.

For additional transparency and to reduce mental load, a formal representation of the selection, as it is send to the backend, is displayed in JSON format (ref. Figure 6). In order to formulate a more complex selection, the user is provided with an interface that lets her build a query using simple HTML elements. Each query contains a key-value pair for the following properties: First, the field of interest (category, prediction, length, characters) is specified using a dropdown menu. Then, the values for this field that should be included in the selection can be set via two input fields that represent the

(inclusive) start and end value. When the user selects a field of interest, the start and end values are automatically set to the minimum and maximum allowed value respectively to improve usability. Next, there are three additional options that can be configured to refine the selection:

1. Categories: specifies a list of classes (using class indices) that should be considered for the selection.
2. Regex: specifies a regular expression that each domain must match to be included in the selection.
3. Filter: applies one of four filters that select only ‘correctly predicted’, ‘incorrectly predicted’, ‘false positive’ (only malicious instances predicted to be benign) or ‘false-negative’ (only benign instances predicted to be malicious) instances.

3.2.2 Analysis Visualizations

After the user made a selection, the next visualizations allow for a deeper analysis of the inner workings of the model. These visualizations are based on the model nodes’ activations, since they function as the model’s view of the data and are available for any kind of deep learning model, meaning that the visualizations can be constructed for different deep learning model architectures. Activations can be viewed on different levels of aggregation, starting at the node level with no aggregation, which is the activation of a single node in a layer. In the context of convolutional models, this refers to a single convolutional filter which outputs a matrix. For a recurrent model, this would be the scalar output of a recurrent unit. The next level, which is the one we decided to use, aggregates activations on a per-layer basis, which combines the activations of all nodes in the layer. For some types of architectures there are more ways of aggregating activations, but since these are not available for all types of deep learning models, they are not an option for our system. The model's activations are high-dimensional, making their visualization challenging, especially when the dimensions carry no easily comprehensible meaning. The latter implies that common used visualizations like parallel coordinates, scatter plot matrices or glyphs only provide limited usability while their disadvantages remain. These techniques do not scale well with large numbers of dimensions, may not scale visually with large numbers of instances and can be hard to interpret with semantic meaning for individual dimensions or features.

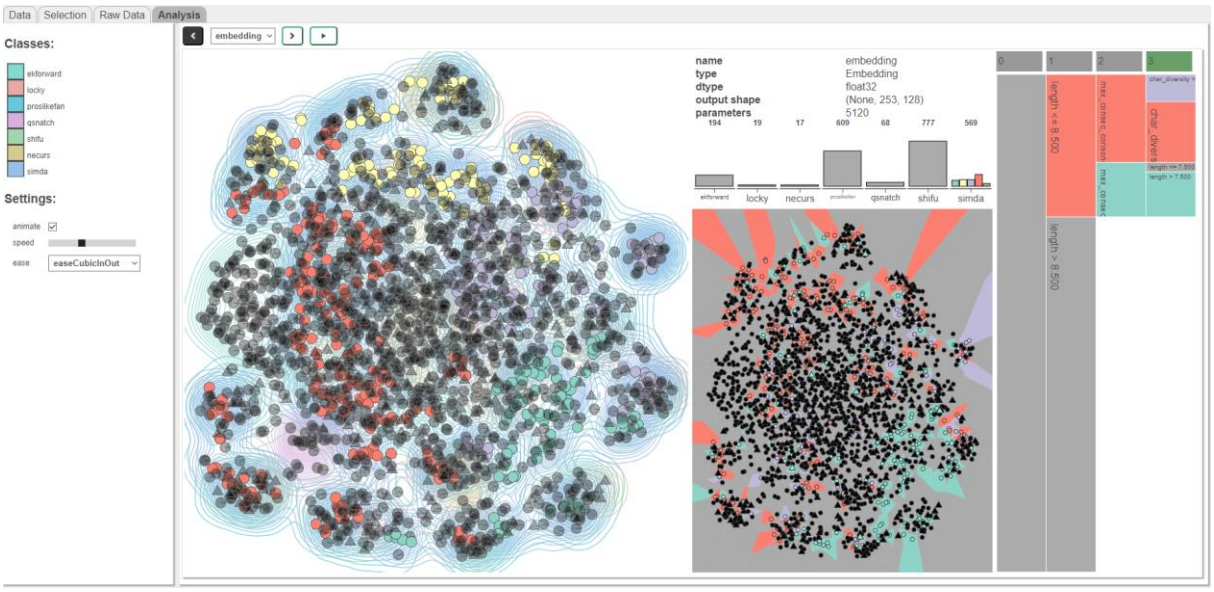

Figure 8: Analysis view including a class legend (top left), animation settings (middle left), and four visualizations (right).

We display four visualizations (see Figure 8) for each layer of the model. This creates the opportunity for the user to investigate at which layers the model differentiates between different instances in order to get a better understanding of its functionality.
The first visualization (see Figure 9) makes use of the layer's activations in two ways. We employ HDBSCAN [10] clustering on the high-dimensional activations and then perform dimensionality reduction using the UMAP algorithm [11] to get a 2D projection of the activations. We use the 2D projection to draw a scatter plot and density contours where the clustering labels determine the color of the glyphs in the scatter plot. Correctly predicted instances are displayed as circles and incorrectly predicted instances are shown as triangles. Density contours are drawn behind the scatter plot in order to let the user combine information about clustering, class membership, prediction correctness and spatial proximity at a glance.

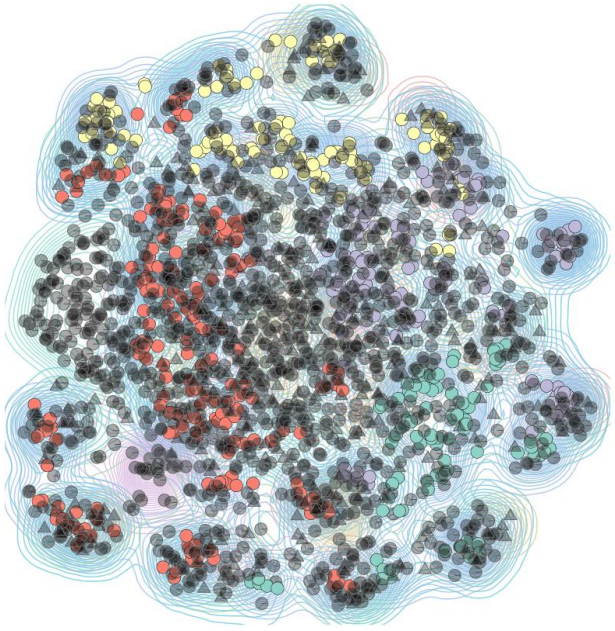

Figure 9: Density-scatter plot consisting of four clusters and seven classes.

The next visualization as shown in Figure 10 is of a supplementary nature and displays how clusters are distributed for each class in the selection. This allows the user to get a quick overview without having to study the scatter plot in detail. For each class, we create a bar chart with the number of instances on the y-axis and the cluster on the x-axis. The bars are colored according to the cluster they represent and their width is determined by number of clusters for the class, since each class has the same space available to display the cluster distribution. In addition, we also show information about the current layer, such as the layer's name, its data type, the data type used by the nodes in the layer, the output shape of the layer and the number of trainable parameters.

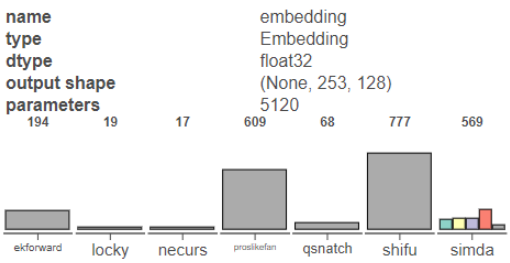

Figure 10: Overview visualization depicting cluster distributions for each class.

The two final visualizations (see Figure 11 and Figure 12) let the user explore the clustering result we computed previously by means of a decision tree trained to explain the clustering labels. First, the decision tree itself is visualized as an icicle plot (see Figure 11), where the size of the rectangle is determined by the number of instances in the selection that belong to this node. Each node includes text describing the branching criterion itself, e.g. 'length <= 8.5' which contains all instances where the domain length is 8 or less (since there are not fractional lengths in this context). The color of a node depends on the color of its children, whereas the color of leaf nodes is determined by the cluster it designates. On top of the icicle plot we show a colored button to indicate the level of the tree currently displayed in the last visualization, the Voronoi diagram (see Figure 12). From the 2D projection calculated earlier, we compute a Voronoi diagram and color the Voronoi cells according to how the corresponding instances are separated by the current level of the decision tree.

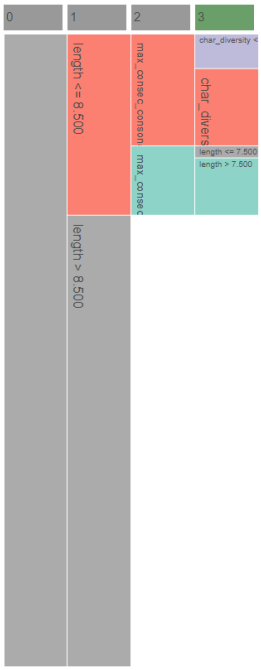

Figure 11: Trained decision tree visualized as an icicle plot.

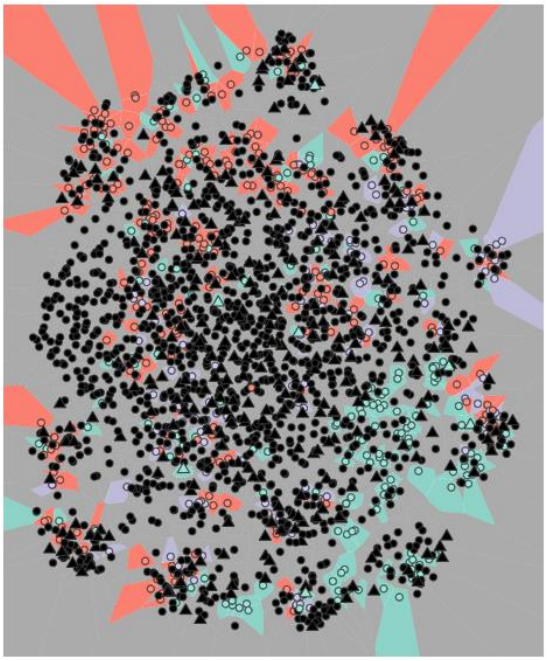

Figure 12: Voronoi diagram that connects the decision tree to the selection.

3.2.3 Interaction Concept

The overview visualization marks the entry point of the analysis, where the user can get a holistic view of the data and the classifier's performance. Bars in the histogram are sized using a local scale and colored according to their value on a global scale. This allows for an investigation on both the local and the global level and also avoids perception problems that could occur when using a global scale for size. Hovering over a bar shows more detailed information for that bar such as the value it represents, the class it belongs to and the feature value it represents. Using brushing mechanisms, the user can define a selection over either a class and a feature value when brushing the bins in a row or over only a feature value by brushing over the global axis at the bottom.

The analysis view offers several interaction mechanisms, most notably related to the decision tree which functions as the center piece of 'interpretation'. Each level of the decision tree can be selected by clicking on its label button, which changes the colors of the accompanying Voronoi diagram to reflect the separation induced by this decision tree level. This bridges the cognitive gap between the decision tree and the instances in the selection, since the feature values used for the decision tree training are not directly visible. Single nodes of the decision tree can also be highlighted in the Voronoi diagram by clicking on the particular node in the icicle plot. This colors all instances belonging to this node white, which can be useful in cases that neighboring nodes use the same color. In the scatter plot, the user can hover over any instance which triggers the display of a tooltip with additional information for that instance, like its associated domain, class and prediction. To better visually distinguish different classes and instances that belong to this class in the scatter plot, both the density contours and the

glyphs of a particular class can be highlighted by clicking on the class' color rectangle in the legend. Highlighting in this case means that the contours for this class are drawn in thicker black lines and the glyphs are drawn with a neon pink outline that provides sufficient contrast to both color maps used for cluster and classes.

3.2.4 Implementation Details

The developed visual analytics system is web-based, consisting of a JavaScript frontend and a python backend using Django [12]. The visualizations in the frontend make extensive use of the D3.js framework [13]. In the 2019 general meeting in Prague, consortium members decided that the users of machine learning-related visualizations developed in the scope of task T3.5 are model designers, not model users like analysts in security operation centers. As a result, the system was not implemented as part of the dashboard developed in work package 6, but was still designed as a web-based architecture in order to keep future possibilities open.

The pipeline for a typical interaction sequence starts with the selection of a model, which triggers the display of the per-class summary visualization. The data for this visualization is precomputed and stored in a relational PostgreSQL database. On the client side, a selection is formulated either via brushing or via interaction with the selection form. The selection is defined on several levels. On the first level, it is defined by a dataset it belongs to. On the second level, a selection is defined over either a class or feature value. On the last level, a selection may be further refined by specifying a regular expression its domains must match or setting a filter to only include correctly predicted instances. Such a JSON selection is send to the backend, where it is parsed into a Django query object, where each query is combined with the previous through a logical disjunction.

The resulting query object is used to probe the PostgreSQL database to get all matching domains used for training and testing. As this selection may be arbitrarily large, confined only by the size of the dataset, sampling is necessary to maintain interactivity. Sampling occurs in a stratified manner on two levels: by ground truth and by prediction. Should a selection exceed a chosen threshold (we chose threshold values between 2000 and 4000, depending on available computing performance), all instances in the selection are divided into (ground truth) classes that exist in the selection. Then, in each class, instances are uniformly sampled based on their prediction in order to cover as much of the prediction variation as possible. This stratified sampling strategy is presented in the pseudo code below.

```
function stratified_sampling(samples, query_result, max_size):

    filter out instances in the query_result that already exist in samples
    class_counts = normalized counts of classes in the selection

    for name, percent in class_counts:

        subset = instances in query_result with class = name
        pred_counts = normalized counts of prediction classes in subset
        uniform = 1.0 / number of classes in subset

        for pred_name, pred_percent in pred_counts:

            X = min(len(pred_subset), uniform * max_size * pred_percent
            add X instances from subset to samples
```

Algorithm 1: Pseudo code depicting stratified sampling strategy to reduce selection size.

The resulting final selection then undergoes several steps to create the data required for the analysis visualizations. First, instances are fed to the model as input and the activations this produces are then captured and downsampled by averaging to speed up subsequent calculations. Then, in case activation dimensionality exceeds 100, PCA [14] is applied to reduce dimensionality to that threshold. This is followed by the application of HDBSCAN clustering and a further dimensionality reduction using UMAP to calculate a 2D projection. HDBSCAN clustering was one of three candidates chosen for initial experiments, alongside agglomerative clustering and OPTICS, which is a variation of DBSCAN [15]. We chose candidates based on input parameters, scalability for large number of samples and common use cases. HDBSCAN's results showed the most promise since its computational complexity scales better than OPTICS and in contrast to hierarchical clustering it is noise-aware, meaning that instances may not be assigned to any cluster if they are more likely to represent some form of noise. It is also possible to use HDBSCAN as a soft clustering algorithm, where each instance is assigned probabilities for all clusters.

To generate the 2D projection, we chose the UMAP algorithm which performs a kind of non-linear dimensionality reduction based on preserving manifold structure in a low-dimensional embedding. It has been show to scale better computationally, as illustrated in Figure 13, and may provide more meaningful global distances than its popular competitor t-SNE [16].

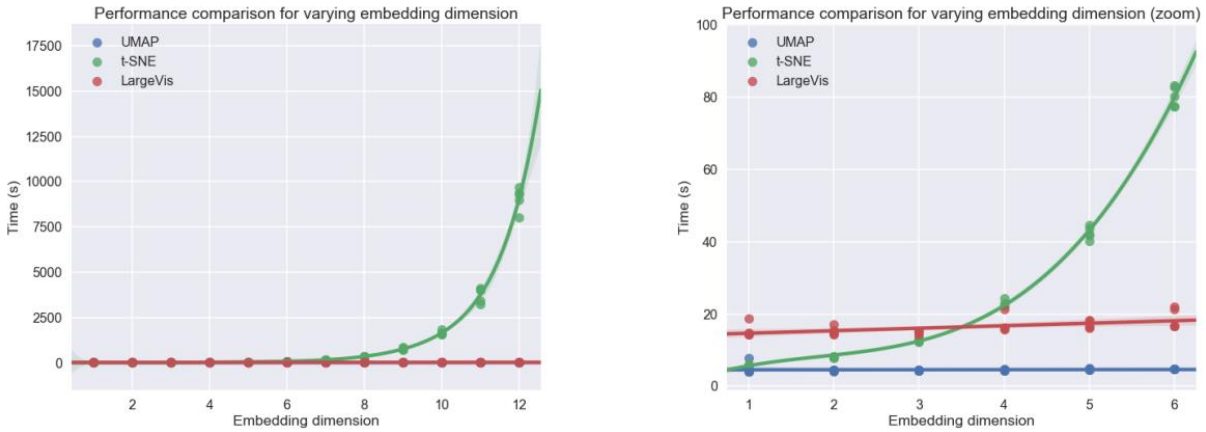

Figure 13: Runtime comparison for UMAP [11], t-SNE [16] and LargeVis [17] algorithms with respect to the embedding dimension on the Pen digits [18] dataset (left). Detailed view of the left chart for embedding dimension six or less, where UMAP appears to scale linearly (right). Graphs taken from [11].

The labels from the clustering step are subsequently employed for the generation of the decision tree. The decision tree uses the 'DecisionTreeClassifier' implementation from scikit-learn [19] with enabled maximum cost-complexity pruning to avoid overfitting and to produce trees that result in manageable visual complexity. This process is repeated for each considered layer in an automated fashion, i.e. the client automatically requests the data for the next layer as soon as the data for the current layer has been received. This makes the delay induced by the processing pipeline less noticeable, since the user will most likely study the generated visualizations for the current layer before moving on to the next. The complete processing pipeline is displayed in Figure 14.

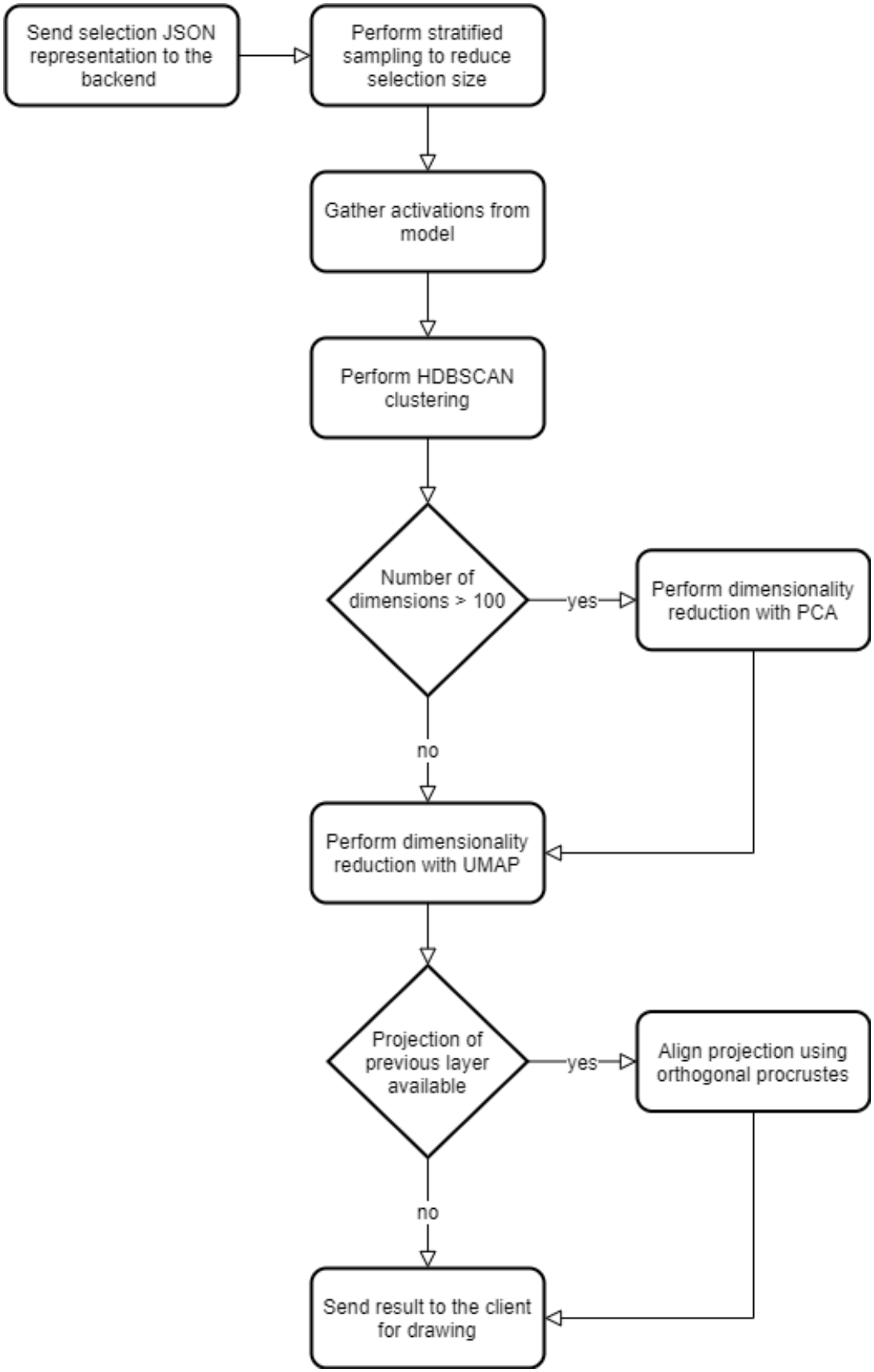

Figure 14: Flowchart depicting the processing pipeline executed for each layer sequentially.

4 Visualization of Host Profiles

While the visualization of deep learning models for DGA classification focuses on increasing the designer's understanding of the model. The visualization support for host profiles provides a first step towards the performance evaluation of machine learning methods of host classification: data visibility. In other words, the main goal of the host profile visualization is to provide the user with insights regarding the state of the network, as provided by the Netflow probes, and allow them to verify the correctness of the model outputs.

For introduction to host profiles, as well as details on data acquisition and data evaluation, please refer to the deliverable 3.4 Algorithms for analysis of cybersecurity data and its chapter on Endpoint profiling.

4.1 Visualization Concept

The visual analytics system for host profile verification is based on a top-down approach to visual analytics, where the analyst begins at a global level overview of the monitored network and with the use of various filters, drills down to a smaller selection of hosts, which may be suitable as a baseline targets for the classifier. These can then be labelled with a known profile and compared to the profile proposed by the classifier. Thus allowing simultaneous labeling of known hosts and classification of the unknown ones. These profile labels based on Netflow data in conjunction with data on system events form the basis for the endpoint profiling as outlined in the SAPPAN showcase #3 and subsequent anomaly detection as described in SAPPAN showcase #4.

The VA system for host profiles consists of three sections, following analysis steps outlined above:

- The overview screen serves as an entry point of the analysis. Its primary objective is to visualize hosts and their attributes in relation to their subnet membership since most of the networks use subnet-based segregation. The heat map and the associated filters facilitate an informed host selection process, as well as an exploratory analysis of the host properties.
- The timeline section allows the analyst to determine, whether or not the measured metrics represent stable behavior or a temporary spike in host activity. In the case of the latter, it may not be desirable to use such a host in the baseline selection.
- The comparison tab allows for the comparative analysis of the selection and manual labeling. The host selection can be subdivided further based on varying metrics.

All the sections have predictable user navigation and interaction layout. From right to left, each section starts with filtering and configuration options, followed by a set of visualizations specific for the section. A special attention is paid to the cognitive complexity of the visualizations. A simpler types of visualizations are preferred if the users are more familiar with how they work. The familiarity levels for the visualization techniques were taken from the survey conducted as part of deliverable D2.3.1 Visualization Requirements, in which users have expressed great familiarity with timeline-based visualization techniques and filter-based interaction approaches.

4.1.1 The Overview Dashboard

The first screen displays the whole /16 subnet of the monitored network as a zoomable heatmap based on the values of the selected metric. Each host is represented by a single point on the map identified by their IP address. The white patches of the map either signify that the addresses were discarded by the filtering options, or the data for given IP addresses are missing. This can happen either due to the host not being monitored, or the IP address not being assigned.

Figure 15: Overview visualization.

The configuration options include a choice between Inbound/Outbound traffic, the metric displayed on the map, the projection (layout) of the hosts, coloring either by a given range or partitioning by a given threshold, and then options based on the selected coloring type.

The heatmap visualization offers three types of projection: cartesian, Hilbert, and Morton. In the cartesian projection, the points are arranged into a grid where their x coordinate matches their IP subnet identifier and their y coordinate is given by their IP host identifier. The Hilbert and Morton projections project the cartesian arrangement onto the respective space-filling curve, packing IP addresses within the same subnets into a shared square, as opposed to a line in the cartesian view.

When the heatmap is being colored by range, the user may constrain the displayed values to an arbitrary interval. If no values are provided, the minimum and maximum of the given metric across the whole dataset is assumed. The points on the map are then linearly colored according to a user-defined color scheme: a gradient with an arbitrary number of transition points. Should the user choose to color the map by a threshold value, the points are chromatically separated into two groups by their distance from the threshold value (a median by default). That is the points with a value of the median will be colored with a color in-between the gradient ends, while those at the extremes will be closer to one of the gradient ends.

Although at its present state, the heatmap is colored by the values of a single metric, multiple approaches for visualization of compound metrics have been explored, however, the optimal solution is yet inconclusive. The main issue when displaying multiple metrics at once is the sudden increase in cognitive complexity of the visualization, directly proportional to the number of displayed metrics. The most promising concepts revolve either around using configurable weighted compound index value, or displaying the rate of similarity after a baseline profile has been selected.

The heatmap allows the user to identify the points of interest inside the monitored landscape by filtering out irrelevant and coloring relevant hosts. The state of the visualization is being serialized into the URL on every change, allowing for saving and recalling the setting later as well as sharing them with other users.

4.1.2 The Detail View

The second screen becomes available once the user selects a particular host from the overview. This one provides the user with the development of its metrics over time using a set of zoomable line-charts (see Figure 16). The screen contains either one or three-line charts, depending on the data available. Some hosts may not contain data on both inbound and outbound traffic. In such cases, only one-line chart is displayed. The charts allow the user to quickly identify whether or not a state reported by the heatmap belongs to a regular state of affairs, or if the value is a recently encountered spike in values. The time frames of the charts can be locked together, such that a brushing-zoom operation on one will propagate the new time frame to others. The user can choose an arbitrary set of metrics to display.

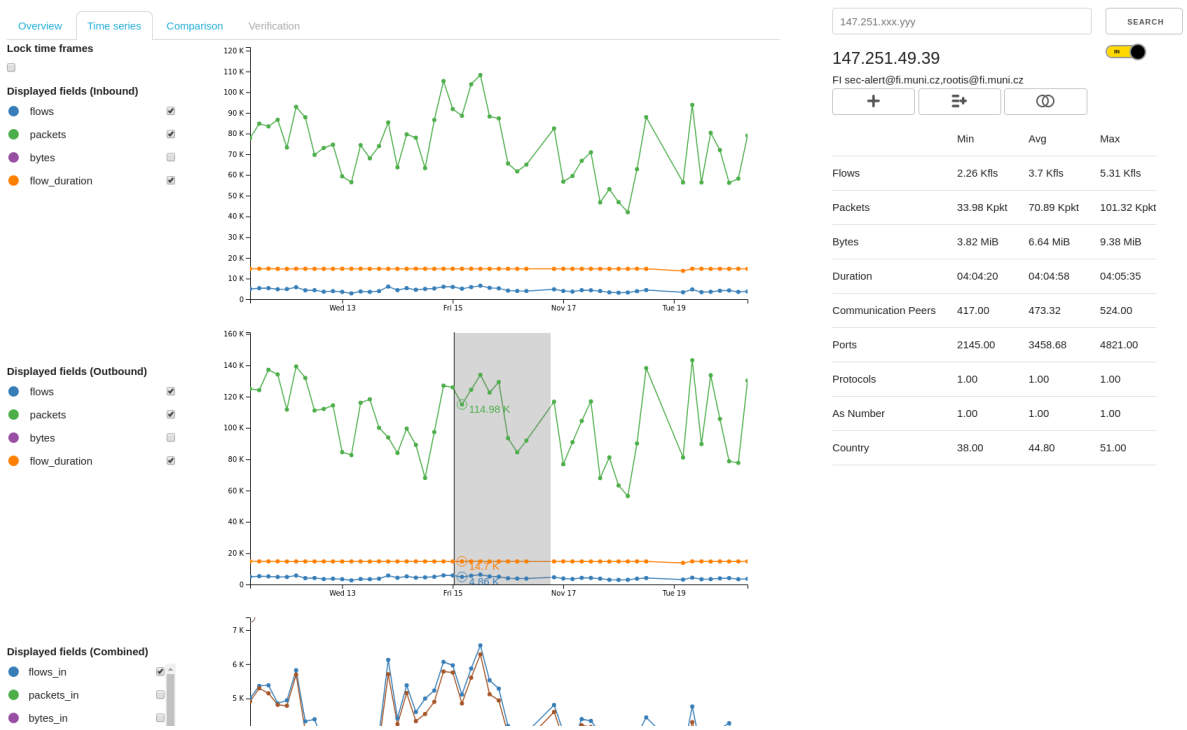

Figure 16: Detail view.

4.1.3 The Comparison Screen

The user may choose a subset of hosts for comparison using the icon buttons in the upper section of the information column. These are then added to the comparison screen (see Figure 17), which allows the user to quickly compare the metrics of the

selected hosts and classify them into their corresponding profiles. The upper part of the interface contains a parallel coordinates chart, with hosts as the horizontal lines and the selected metrics as the vertical ones. It allows the user to quickly spot differences among the hosts. However, it was found that the chart becomes rather confusing when a larger (> 10) number of hosts is being compared. Therefore, we included a set of radar charts in the bottom part of the screen that allows the user to compare and classify the hosts on selected metrics. All unclassified hosts are displayed in the main chart. The user then creates an arbitrary number of groups for the hosts. The host can be moved from the main chart onto one of the group ones. The groups should be persisted on the server. The current limitation of the labeling system is that it assumes a single category for each host. In reality, a tag-based approach should be more suitable and most likely will be added in the future.

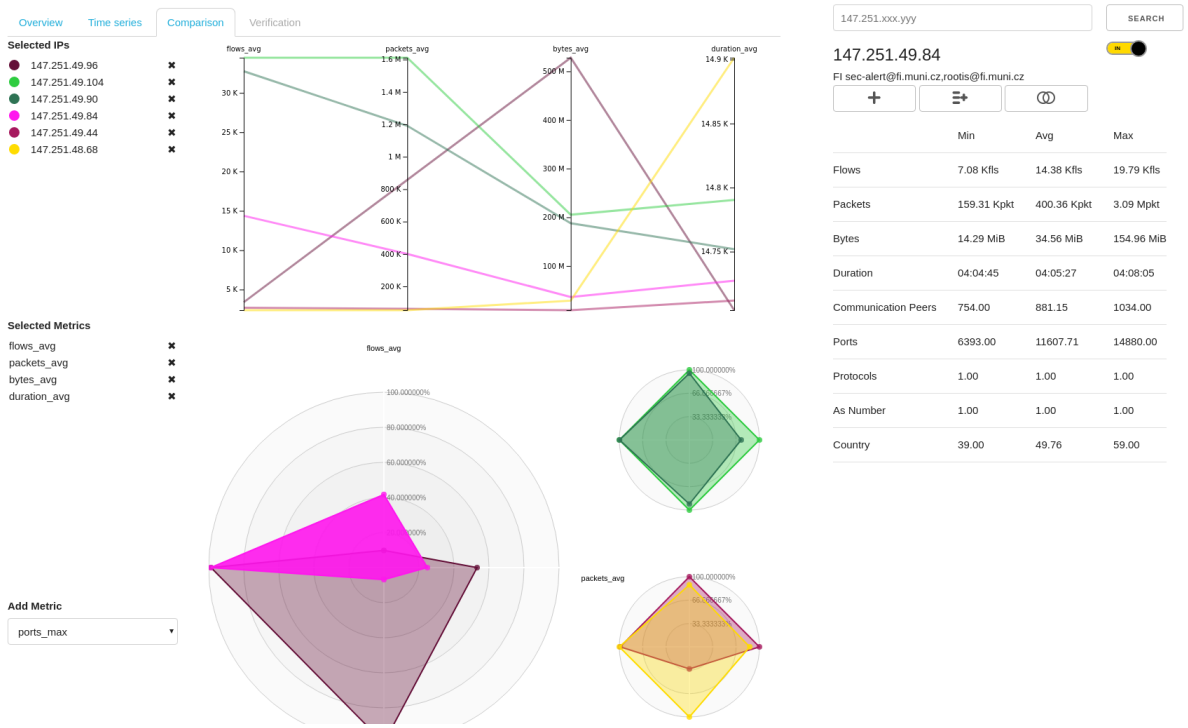

Figure 17: Comparison view.

4.2 Implementation

The visual analytics system for host profile data visualization is a web application written in pure JavaScript with heavy use of the D3js library [13]. The data themselves are collected using a set of python scripts and stored in PostgreSQL database. These are then exposed to the frontend application using PostgREST, a thin wrapper around the database, providing authenticated access to CRUD operations over REST API. The visualization itself operates on aggregated data, with configurable aggregation window. Currently set to 4 hours as default. The main advantage of this technology stack is that it does not require any external dependencies, aside from the software mentioned and therefore does not need any kind of complex build system. The whole application can be set up and deployed in a matter of a few minutes.

Another important aspect of the system is the fully-serializable application state, which is stored in the URL itself, allowing for easy sharing of analysis results without the need for centralized state storage. The only exception being the categorization data used in the comparison step, which are persisted along with the IP address in the database.

5 Future Work

This section discusses any future work that may be performed for the final version related to this deliverable or efforts beyond this project.

5.1 DGA Model Visualization

Since this report details an initial version, we intend to keep working on this system and formally evaluate it.

5.1.1 Uncertainty and Collaborative Learning

Sources of uncertainty can be found in almost any area, and machine learning does not pose an exception in that regard. To create awareness for uncertainty that stems from machine learning and in particular from the collaborative learning settings explored in work package 5, we plan to explore how the prototype developed in this task may be extended in task T5.5 of work package 5.

5.1.2 Sensemaking and Reasoning

We started exploring the addition of functionalities that let the user keep track of their insights and support them during their analysis. The analysis may be viewed as a sensemaking or reasoning activity and could consequently benefit from the incorporation of principles and findings from those domains. For our first exploration, we added the option to record automatically generated descriptions of parts of the visualizations together with a timestamp. As an example, the user can right-click on a bar in the overview visualization which opens a dialog where the user can add custom tags and a custom description, but we provide a generated gist of the insight that may be derived from the bar following this scheme: “50 instances (33%) from class ‘X’ have a ‘length’ value of 18”. Figure 18 shows an example dialog that opens when right-clicking on a bar in the overview visualization and Figure 19 displays several recorded insights and their timestamps.

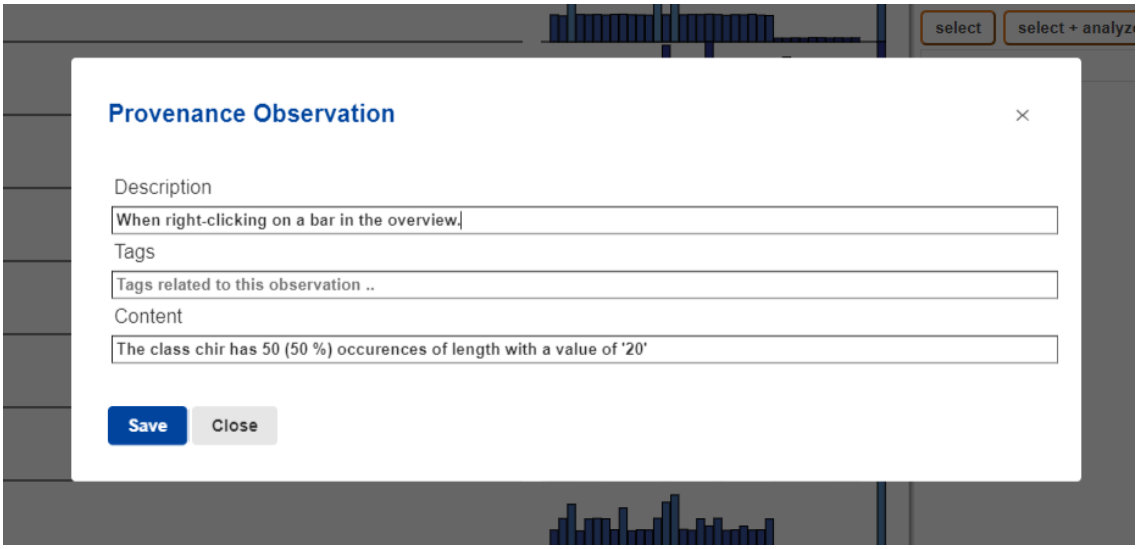

Figure 18: Dialog that is displayed after right-clicking a bar in the overview visualization.

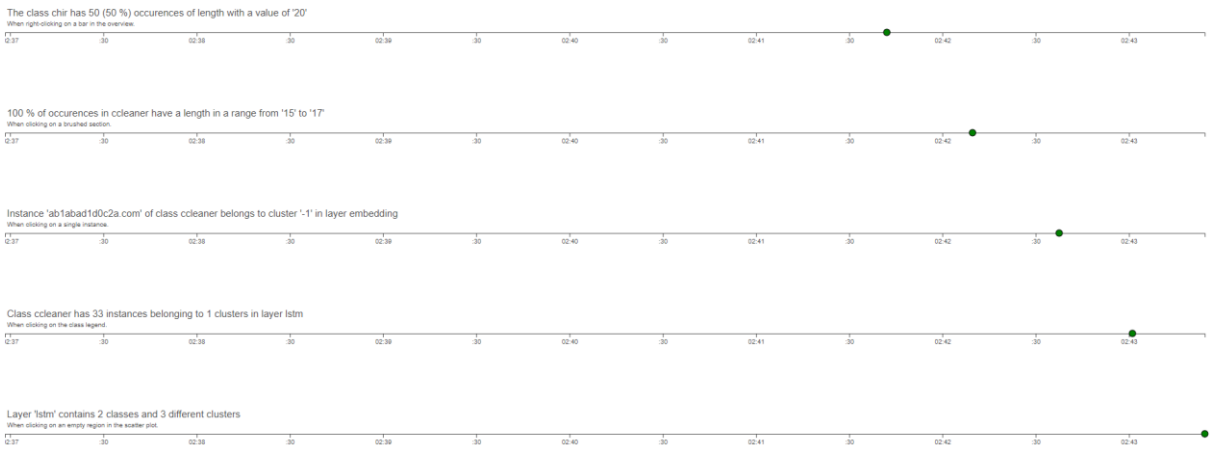

Figure 19: Overview of all recorded observations.

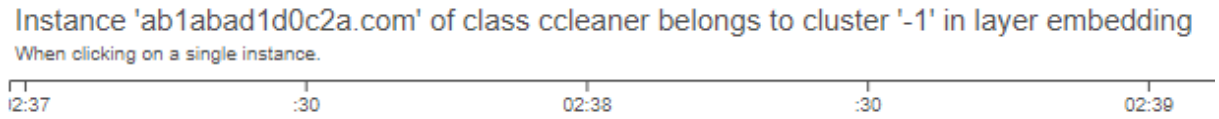

Figure 20: Details of one particular observation in the overview of all observations.

5.1.3 Evaluation

Visualization connects the capabilities of automation and human intellect and therefore needs to be evaluated not only for correctness but also for compatibility to human perception and reasoning mechanisms. For complex visualization systems targeted at expert users, qualitative evaluation is often the evaluation means of choice, as it is hard to find the required number of experts and formulate the right *tests* a quantitative evaluation would demand. Consequently, we plan to perform expert interviews that capture the usability and utility of our developed visual analytics system. While usability can be judged directly from user feedback and comments, investigating the utility is a more challenging endeavor.

We plan to orient our evaluation on the scenarios presented by Brittany and colleagues [20] in a recent workshop for XAI research. They identify five use cases for which a visualization might be used:

1. Model debugging and validation: Does the model work as intended, why does it make mistakes?
2. Model selection: Which model is the best for a given task (beyond performance metrics)?
3. Mental model and model understanding: How does the model behave? Is there something to learn about the general model behavior?
4. Human machine teaming: Can the task be performed better by just the model, just the person or both working together?
5. Model feedback, challenging and prescription: When is the user affected by the model's decision? How does the user react to model decisions?

First, they recommend defining and measuring a performance baseline to compare against. Depending on the task and explanation, one can choose either human only,

machine only or human and machine as the baseline performance. This helps in measuring the effect of the explanation on performance or identifying overtrust in model decisions. For our system, we consider model debugging and validation, model selection and mental model and model understanding as the most important use cases to evaluate, which can be tested using the following downstream tasks from [20].

Model debugging and validation: (1) Given an incorrect model decision and corresponding explanation, determine the reason the model made a mistake. This may be set up by adding errors to the data such as wrong class labels in our context or providing too little data for some classes. Reliability between participants can be determined from the found mistakes and the users' ability to correctly describe the reasons for the model's mistakes are measured with and without the explanation. (2) Identify if the model will improve from additional training. Provided three datasets and a model trained on one of these datasets, the user must decide whether the inclusion of the other datasets in training would improve performance. (3) Given a model that exploits artifacts or loopholes of the data, describe the model's behavior. Performance for this case can be measured by determining whether the participants discover undesirable behavior in the model's decision making. This may be engineered by adding patterns to specific classes in the training data.

Model selection: Determine which model is better suited for a given task. Several variants for the experiment can be performed. One variant is to rank multiple models which are presented to the participants either with or without the explanation for comparison. Another option is to measure users' performance in identifying which one of two models would perform better on an unseen test set.

Mental model and model understanding: (1) Given examples of past behavior, extrapolate what a model will do given unseen input. First, users are given a series of inputs and the matching model outputs. Then, users are given a previously unseen set of inputs and must give their expected model output. Some users will have the corresponding explanation available while the control group will not have that. The accuracy of users' predictions then provides an indicator for the quality of the users' mental model (of the machine learning model). (2) Given information about a model's past performance, match the model to a novel output. A variant for this use case for one model would be for the researcher to manipulate the output of a model and then see if the participants formed a well enough mental model to spot these modifications. (3) Given a model's past behavior, identify if explanations speed up users' creation of a mental model. In this task, users do something similar as in the previous use case by matching model output to previously unseen input. After a learning phase, they enter an evaluation phase where they are timed to see how their speed changes depending on the presence of an explanation for the model outputs.

Some of these scenarios may require the generation of a custom dataset and model with specific characteristics. This can also alleviate bias that may be present for users that are already familiar with data used for model training. For our evaluation, we intend to implement one scenario from each use case and test them with machine-learning experts. Given the global health crisis, the evaluation may be limited in the number of participants we can procure and what we can evaluate in a remote setting.

5.2 Host Profile Classifier Verification

When it comes to the host profile visualizations, a notable area of improvement, as well as planned future addition to the system, is the ability to modify the model input values and see the model classification output changes in real time, thus increasing the intuition for the model's reasoning. Such capability would also come in handy, in the model debugging process, since it would allow to verify hypotheses regarding the classifier behavior in an interactive and understandable way, without the need for direct analysis and exposure of the inner workings of the model. At the moment the VA support for such system is in very early stages of development.

6 Summary

This report illustrates two different visualization systems developed to support experts in designing models for the tasks of DGA classification and host profiling. The former is only an initial version that still has to undergo empirical evaluation.

References

[1] D. Gunning, "Explainable artificial intelligence (xai)," *Defense Advanced Research Projects Agency (DARPA),* 2017.

[2] A. B. Arrieta, N. Díaz-Rodriguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion,* pp. 82-115, 2019.

[3] S. Bach, A. Binder, G. Montavon, K. Frederick, K.-R. Müller and W. Samek, "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation," *PLoS ONE,* 10 July 2015.

[4] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye and A. Mordvintsev, "The Building Blocks of Interpretability," *Distill,* 6 March 2018.

[5] P. E. Rauber, S. G. Fadel, A. X. Falcao and A. C. Telea, "Visualizing the Hidden Activity of Artificial Neural Networks," *IEEE Transactions on Visualization and Computer Graphics,* pp. 101-110, 1 January 2017.

[6] A. Chatzimparmpas, R. M. Martings, I. Jusufi and A. Kerrean, "A survey of surveys on the use of visualization for interpreting machine learning models," *Information Visualization,* pp. 207-233, 2020.

[7] F. Hohman, M. Kahng, R. Pienta and D. H. Chau, "Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers," in *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[8] J. Choo and S. Liu, "Visual Analytics for Explainable Deep Learning," *IEEE Computer Graphics and Applications,* 2018.

[9] D. Ren, S. Amershi, L. Bongshin, J. Suh and J. D. Williams, "Squares: Supporting Interactive Performance Analysis for Multiclass Classifier," in *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[10] L. McInnes, J. Healy and S. Astels, "hdbscan: Hierarchical density based clustering," *Journal of Open Source Software,* 2017.

[11] L. McInnes, J. Healy and J. Melville, "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction," *arXiv preprint arXiv:1802.03426,* 2018.

[12] Django Software Foundation, "The Web framework for perfectionists with deadlines | django," Foundation Django Software, [Online]. Available: https://www.djangoproject.com. [Accessed 29 January 2021].

[13] M. Bostock, "D3.js - Data-Driven Documents," [Online]. Available: https://d3js.org. [Accessed 29 January 2021].

[14] A. Hervé and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics,* pp. 433-459, 2010.

[15] M. Ester, H.-P. Kriegel, J. Sander and X. Xiaowei, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *KDD'96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[16] L. Van der Maate and G. Hinton, "Visualizing data using t-SNE," *Journal of machine learning research,* pp. 2579-2605, November 2008.

[17] J. Tang, J. Liu and M. Zhang, "Visualizing Large-scale and High-dimensional Data," in *WWW' 16: Proceedings of the 25th International Conference on World Wide Web*, Montréal Québec, Canada, 2016.

[18] D. Dua and C. Graff, *UCI Machine Learning Repository,* University of California, Irvine, School of Information and Computer Sciences, 2017.

[19] F. Pedregosa, G. Varoquaux, A. Gamfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* pp. 2825-2830, 10 November 2011.

[20] D. Brittany, M. Glenski, W. Sealy and D. Arendt, "Measure Utility, Gain Trust: Practical Advise for XAI Researchers," in *IEEE Workshop on TRust and EXpertise in Visual Analytics (TREX)*, Salt Lake City, UT, USA, USA, 2020.