Supplemental Materials for "Data-driven Multi-level Segmentation of Image Editing Logs"

Zipeng Liu zipeng@cs.ubc.ca
Zhicheng Liu leoli@adobe.com
Tamara Munzner tmm@cs.ubc.ca

Table of Contents

Section 1: Features

Commands

We exported 100 million events which contained 1949 unique commands from the database in May 2018. The command frequencies follow a long-tail distribution: a few commands are frequently used, whereas many commands are rarely used. We inspected the distribution of commands ordered by frequencies as a data quality check, particularly those at the head of the distribution. Examples of the most frequent commands are *Open*, *Move*, and *Crop*. The actual number of unique commands in the database is larger than the number of menu items in the PS interface (about 1400), due to non-standard commands like Photoshop (PS) Actions[1], and multilingual versions of the same command. Many of the infrequent ones are the user-defined Actions that are personal rather than universal and thus are not useful for us.

We also note that the command name alone does not always contain enough semantic information to fully disambiguate user intention or the task at hand, however. For example, *Brush Tool*, one of the most frequently used commands, can be used to either paint colors on canvas or edit a layer mask.

Grouping these events by session identifier and ordering them by timestamp, we reconstructed 169,387 sessions. We specifically define a session to be a temporal sequence of events triggered by a user when completing a PS task.

We used word2vec to learn a vector for each command. We conducted a sanity check of the embedding space using the Google Embedding Projector with t-SNE to inspect some of the clusters in the interface, and found that the structure of this space was indeed plausible. For example, in Figure 1, the red circle contains many commands for changing layer blending options.

[1] Actions and the Action Panel in Photoshop: https://helpx.adobe.com/photoshop/using/actions-actions-panel.html

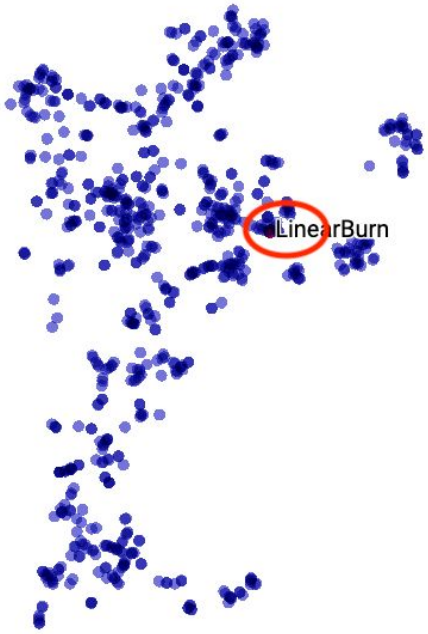


Figure 1: Embedding space neighborhood of *Linear Burn* command.

Figure 2 shows the neighborhood around the *Brush Tool* command, which also has plausible command names such as *Eyedropper Tool* and *Mixer Brush Tool*.

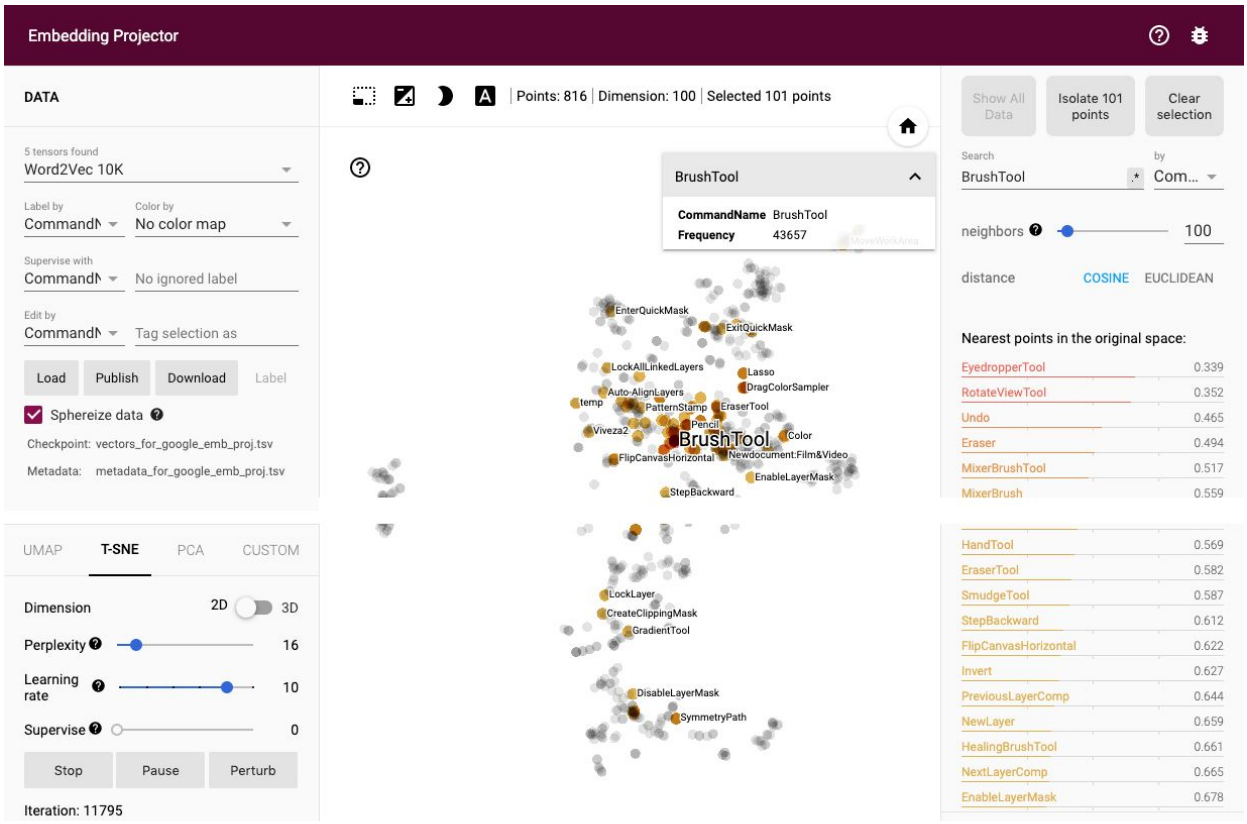


Figure 2: Embedding space neighborhood of *Brush Tool* command.

Finally, Figure 3 shows the neighborhood of the *New Color Fill Layer* command, which creates a new adjustment layer. Again, nearby commands such as *Modify Color Fill Layer* and *Modify Levels Layer* seem to capture reasonable similarity.

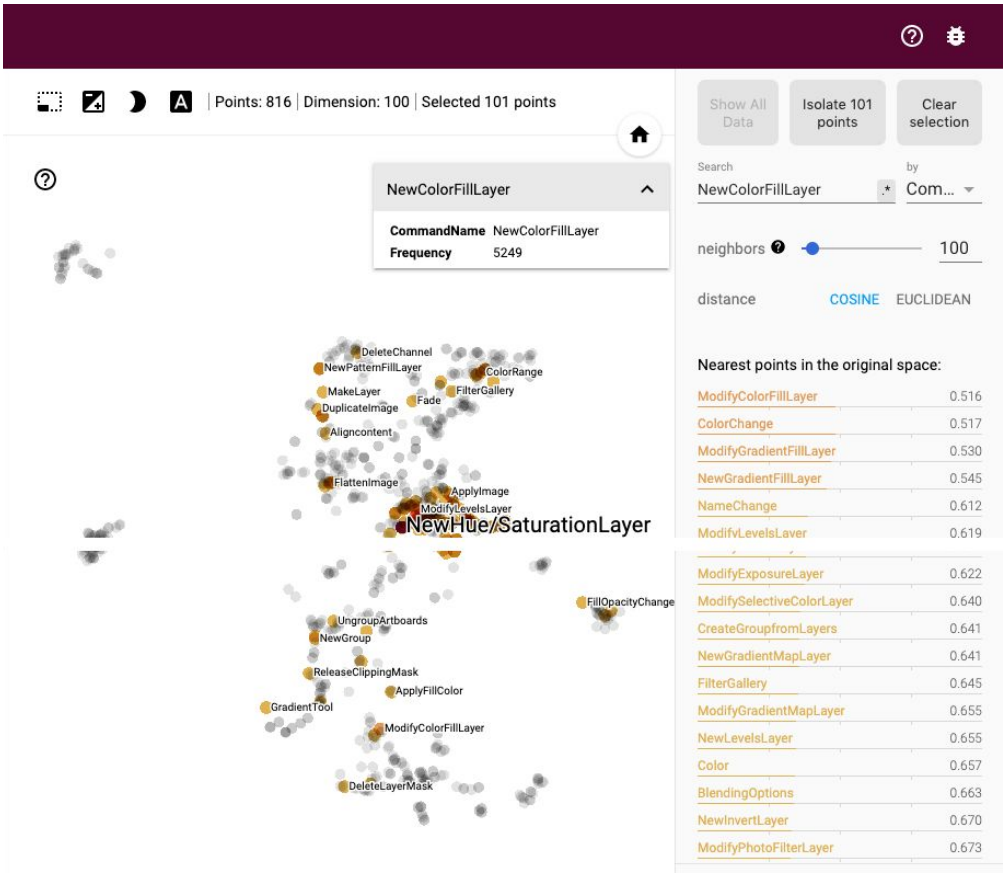


Figure 3: Embedding space neighborhood of *New Color Fill Layer* command

Layers

Example of layer hierarchies

Layer hierarchy examples from three sessions (S1, S2, S14). The annotations on the left show examples of layer relationships and the corresponding layer similarities.

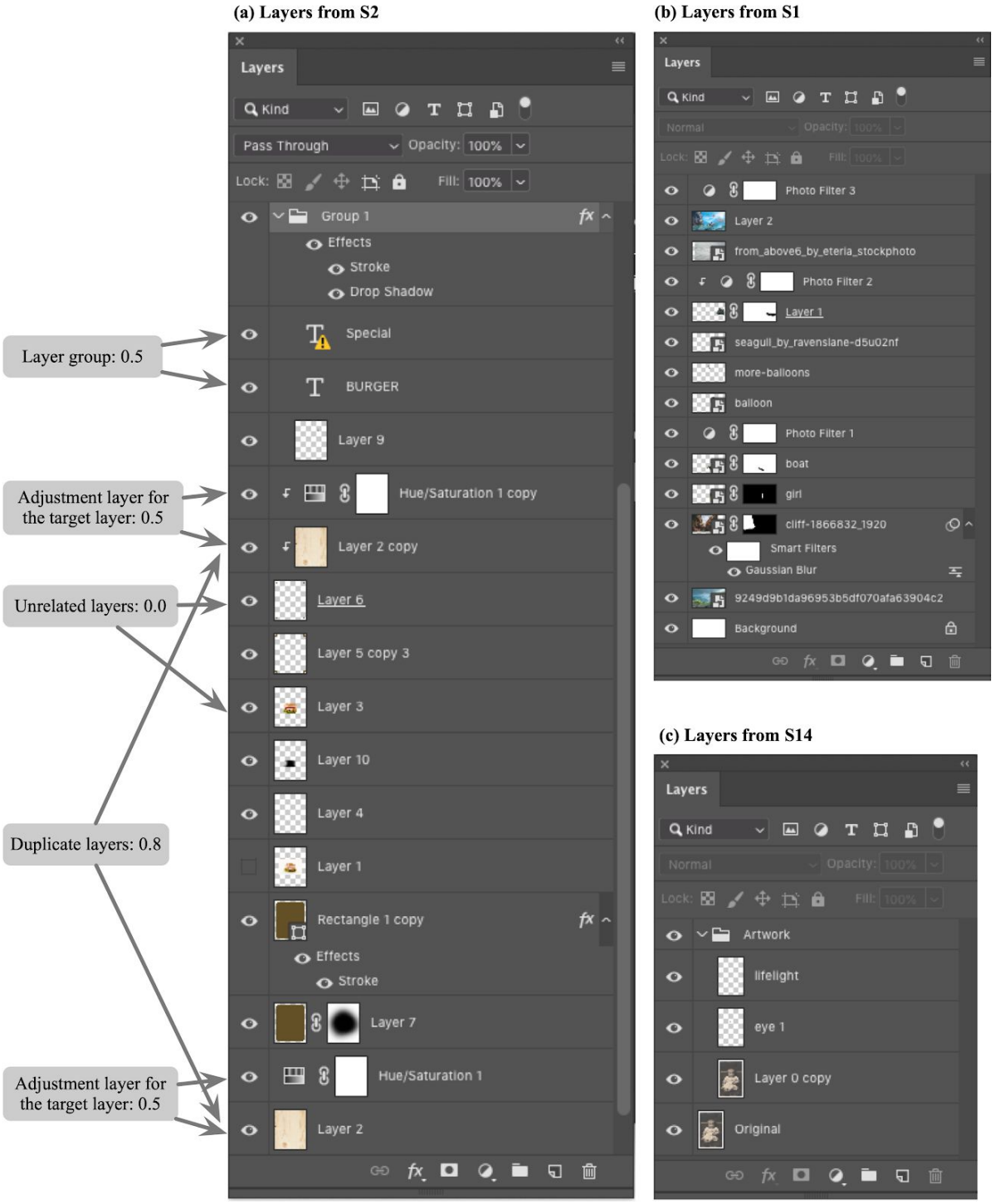


Figure 4: Layer hierarchy examples.

The layer similarity feature computation requires complete information about the layer hierarchy as it has evolved up to any specific time point, so it is derived using information spread across multiple individual event attributes.

Equations to compute the layer similarity.

The overall equation for layer similarity between two layers A and B: If A and B are identical layers, the similarity is maximum (1.0); otherwise, it is the sum of layer similarity for duplicate layers, adjustment layers, and layer group, capped by 1.0.

$$S(A,B) = \begin{cases} 1.0 & \text{if } A = B \\ min(1.0, S_{dup}(A,B) + S_{adj}(A,B) + S_{grp}(A,B)) & \text{if } A \neq B \end{cases}$$

The similarity for duplicate layer is computed as follows.

$$S_{dup}(A,B) = \begin{cases} 0.8 & \text{if A duplicates B or B duplicates A} \\ 0 & \text{otherwise} \end{cases}$$

The similarity for adjustment layer is computed as follows.

$$S_{adj}(A,B) = \begin{cases} 0.5 & \text{if A is adjustment layer of B or the other way around} \\ 0 & \text{otherwise} \end{cases}$$

The similarity for layer group is computed as follows. The length of path between A and B in the layer hierarchy is denoted as d. Most cases are direct siblings (d=2), and the layer similarity would be 0.5.

$$S_{grp}(A,B) = \begin{cases} \frac{1}{2^{d-1}} & \text{if A and B are in the same layer group} \\ 0 & \text{otherwise} \end{cases}$$

Section 2: Data collection

Table 1: Summary of task type

Task type	#Sessions in 1st round	#Sessions in 2nd round	Total
Poster creation	4	4	8
Portrait retouching	3	3	6
Special effect creation	1	1	2

Table 2: 8 collected sessions in the first round

Session ID	Participant ID	Task Type	Reference Image	#Events	Usage in model
S1	P1	Poster creation		275	Training and validation set
S2	P2	Poster creation		391	
S3	P3	Portrait retouching		648	
S4	P4	Special effect creation		130	
S5	P5	Portrait retouching	Same as S3	172	
S6	P5	Portrait retouching		29	
S7	P6	Poster creation		1064	
S8	P7	Poster creation	Same as S1	179	

Table 3: 8 collected sessions in the second round.

Session ID	Participant ID	Task Type	Reference Image	#Events	Usage in model
S9	P8	Portrait retouching		428	Training and validation set
S10	P8	Portrait retouching		180	Test set
S11	P9	Poster creation		261	Training and validation set
S12	P10	Poster creation		323	Test set
S13	P11	Special effect creation		542	
S14	P5	Portrait retouching	Same as S9	426	
S15	P12	Poster creation	Same as S11	380	
S16	P13	Poster creation	Same as S12	297	Training and validation set

Section 3: Hyperparameter Tuning for SVM

We briefly review how a trained SVM model predicts a label given an event. It uses a kernel function that takes the feature vector as input and gives signed distance from the data point to a separating hyperplane as output, which is then converted to the probability of true prediction. Finally, it compares this probability score to a threshold t (model hyperparameter): if probability is greater than t, then this event is predicted as boundary.

There are two hyperparameters in our model: window size k, the number of previous events used in the feature vector, and threshold t, to determine the probability value where an event is predicted to be the boundary. We train the model on the training set, and measure performance on the validation set to tune these hyperparameters, following common practice in machine learning.

Window size k

For each window size k from 1 to 10, we assess model performance after model training by computing and plotting its receiver operating characteristic (ROC) curve, shown in the following figure.
The ROC curve is drawn by varying the hyperparameter threshold t from 0 to 1 and plotting a point on the curve for each value of t. The x axis is the false positive rate, and the y axis is the true positive rate for that threshold value. The area under curve (AUC) of a ROC curve indicates the probability that the model will rank a randomly chosen boundary event higher than a randomly chosen non-boundary event. A common interpretation is that the larger the area is, the better performance the model has. From qualitative inspection of the figure, we can see that the 10 curves are closely intertwined with each other, and they are all far above the diagonal curve representing random guesses, indicating that they are similarly good classifiers. Quantitatively, we can see that the AUCs are also highly similar (min: 0.938, max: 0.962).

We conclude that window size does not influence model performance, and thus we choose k=1 to reduce the size of feature vectors.

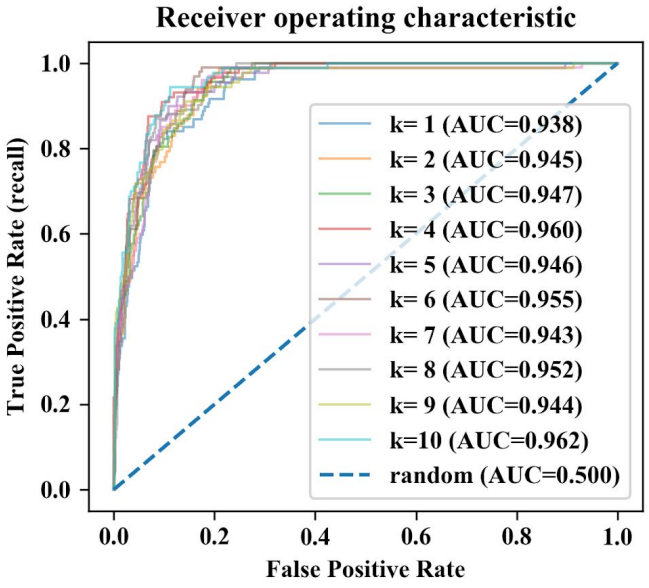


Figure 5: ROC curves

Threshold t

After choosing the window size, we investigate the performance of the model with k=1 under different threshold values. We plot precision (red line) and recall (blue line) against threshold t (x-axis) in the following figure. The trade-off between precision and recall is easy to spot: as threshold increases, precision increases but recall drops. Considering the downstream applications of the segmentation model, we favor recall over precision. That is, we are willing to trade precision for higher recall because there are many fewer positive labels (boundaries) than negative labels (non-boundaries) in the dataset. It is important to correctly identify as many boundaries as possible, and false negatives (boundaries predicted as non-boundaries) are undesirable as it is not easy for end users to identify these missed boundaries. On the other hand, false positives (predicting non-boundaries as boundaries) are less detrimental: although these errors lead to over-segmentation of the logs, users can quickly dismiss them through an interactive interface. We use the F2 score (thick black line), which weighs recall twice as much as precision, to quantify this trade-off choice.

We find a clear winner, t=0.24, which yields the highest F2 score (0.74), as the threshold for our lowest-level segmentation. The corresponding recall is 0.84, and precision is 0.50.

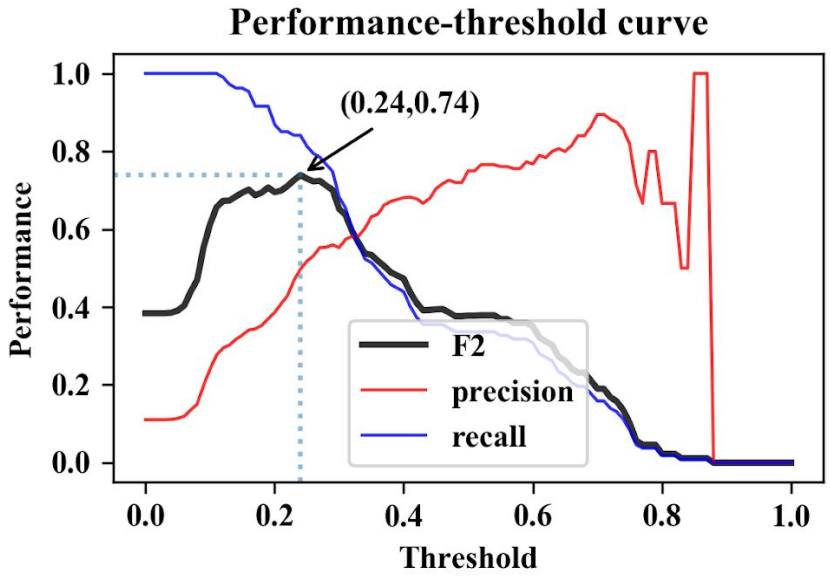
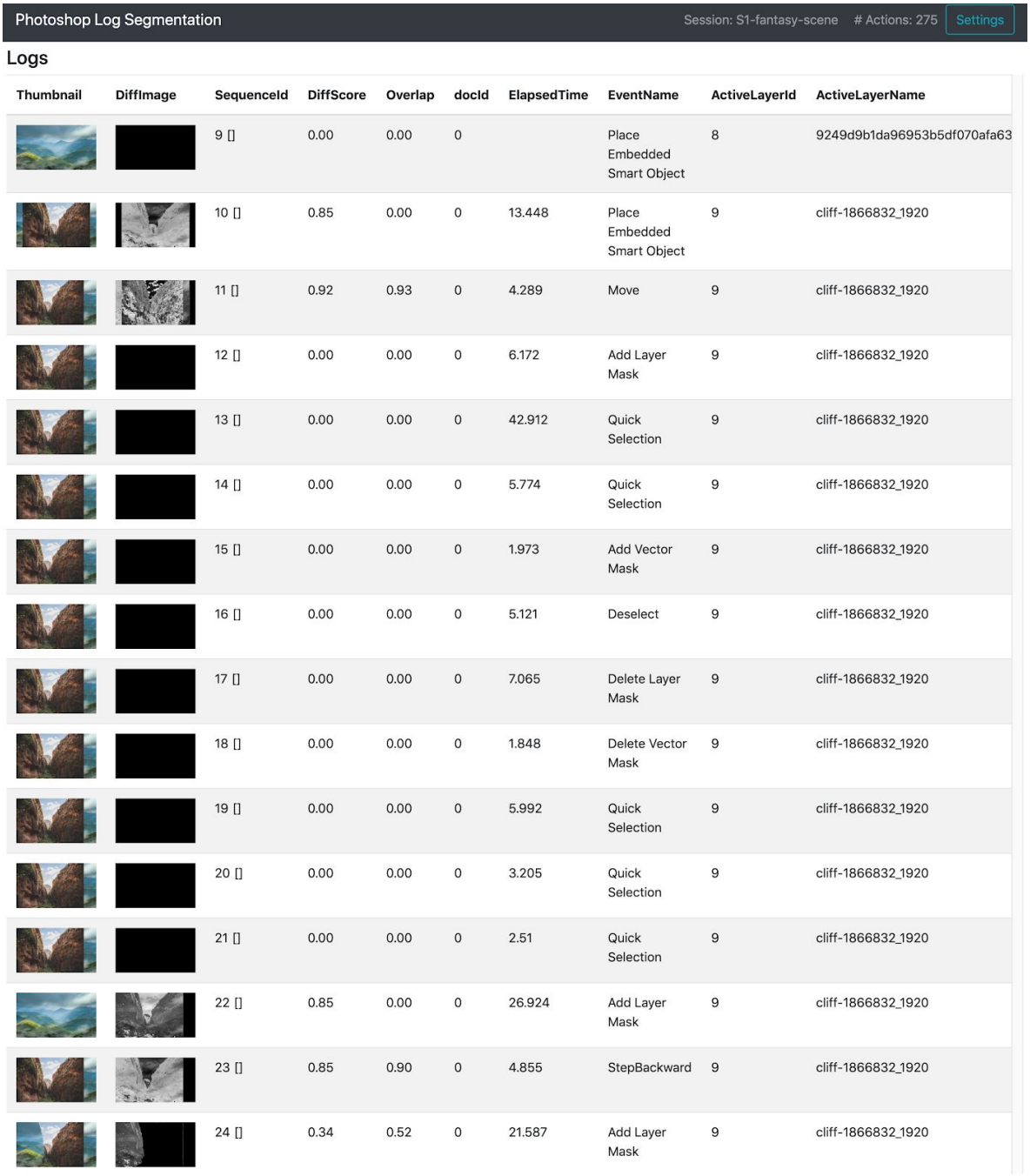


Figure 6: Performance-threshold curve

Section 4: Results

Labeling interface

Figure 7 shows the interface used by paper authors when labeling the sessions. It shows the logged attributes (layer, command, image content, duration) and a few derived attributes such as the diff image.



Thumbnail	DiffImage	SequenceId	DiffScore	Overlap	docId	ElapsedTime	EventName	ActiveLayerId	ActiveLayerName
		9 []	0.00	0.00	0		Place Embedded Smart Object	8	9249d9b1da96953b5df070afa63
		10 []	0.85	0.00	0	13.448	Place Embedded Smart Object	9	cliff-1866832_1920
		11 []	0.92	0.93	0	4.289	Move	9	cliff-1866832_1920
		12 []	0.00	0.00	0	6.172	Add Layer Mask	9	cliff-1866832_1920
		13 []	0.00	0.00	0	42.912	Quick Selection	9	cliff-1866832_1920
		14 []	0.00	0.00	0	5.774	Quick Selection	9	cliff-1866832_1920
		15 []	0.00	0.00	0	1.973	Add Vector Mask	9	cliff-1866832_1920
		16 []	0.00	0.00	0	5.121	Deselect	9	cliff-1866832_1920
		17 []	0.00	0.00	0	7.065	Delete Layer Mask	9	cliff-1866832_1920
		18 []	0.00	0.00	0	1.848	Delete Vector Mask	9	cliff-1866832_1920
		19 []	0.00	0.00	0	5.992	Quick Selection	9	cliff-1866832_1920
		20 []	0.00	0.00	0	3.205	Quick Selection	9	cliff-1866832_1920
		21 []	0.00	0.00	0	2.51	Quick Selection	9	cliff-1866832_1920
		22 []	0.85	0.00	0	26.924	Add Layer Mask	9	cliff-1866832_1920
		23 []	0.85	0.90	0	4.855	StepBackward	9	cliff-1866832_1920
		24 []	0.34	0.52	0	21.587	Add Layer Mask	9	cliff-1866832_1920

Figure 7: Labelling interface

Visualization of sessions in test set

In the main paper we show and inspect 2 out of the 5 sessions (S10, S15) in the test set, and here we show the other 3 test sessions (S12, S13, S14). We include each of the 16 sessions as high-resolution individual PNG files in supplemental materials as well.

Figure 8 shows the visualization of session S12, for a poster creation task. The participant works in a highly organized way, which results in many levels in the human labels. Most of the boundaries in human labels (gaps between grey chunks) for both low- and high-level are also predicted as boundaries in red and blue.

There are a few low-level missing boundaries. For example, in the top highlighted box, participant moved a sphere icon (there are 3 icons in total) using only one event, and then he continued adjusting the positions of the three icons. The model fails to detect this boundary as it thinks the participant is still working on the icons in the icon layer group (high layer similarity) with highly similar commands (*Move, FreeTransform*).

There are many over-segmentations. In the bottom highlight box, the participant tries to put a few layers into a layer group, but they were having trouble locating them in the layer panel, resulting in a series of events on unrelated different layers until he found the correct ones. In this case, the user is having trouble keeping track of the interface, so it would naturally results less structure in the logged data.

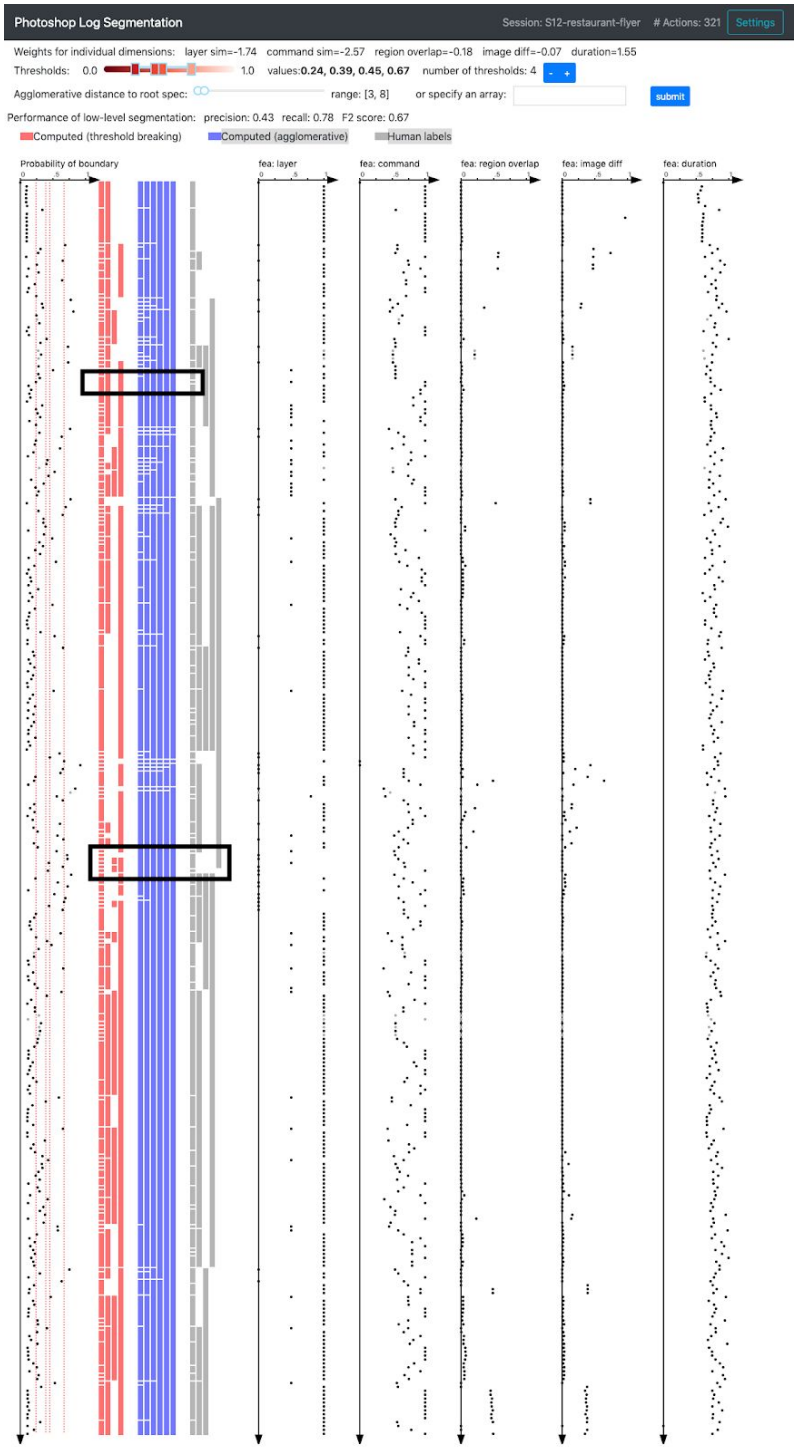
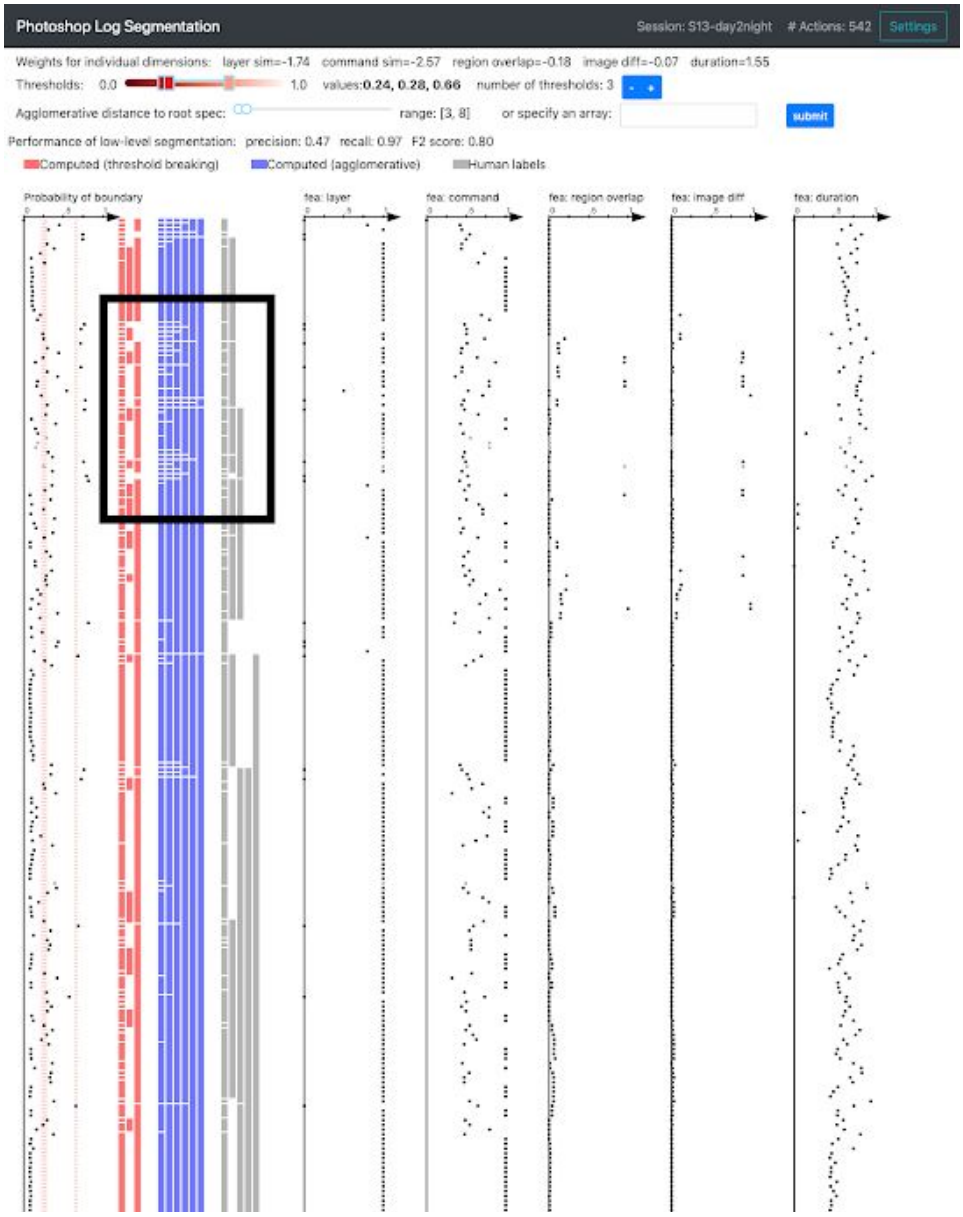


Figure 8: Session S12

Figure 9 shows session S13, a special effect task to turn a day image of a city street into night. The computational results align with the human labels very well for this session, but it comes with a lot of over-segmentation. In the highlighted box, the participant frequently copies and hides a layer as its backup as a way of smart version control: marking the milestone states such that they could go back at anytime later. It involves a lot of layer switching and toggling, leading to many single-event low-level chunks in the computed results.

However, in the high-level chunks (red), the model successfully removes the over-segmentation while keeping the true boundaries in the human labels.



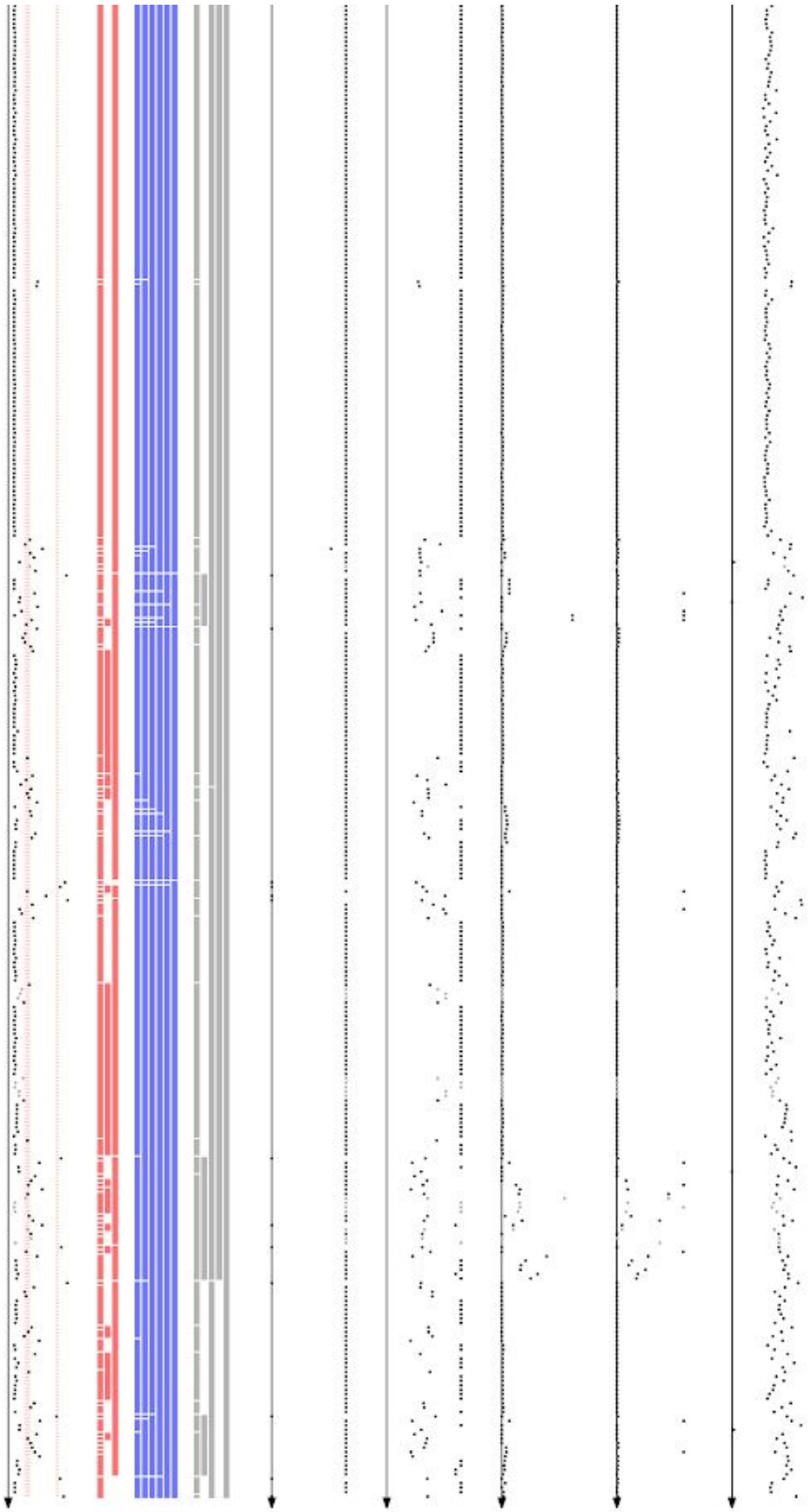
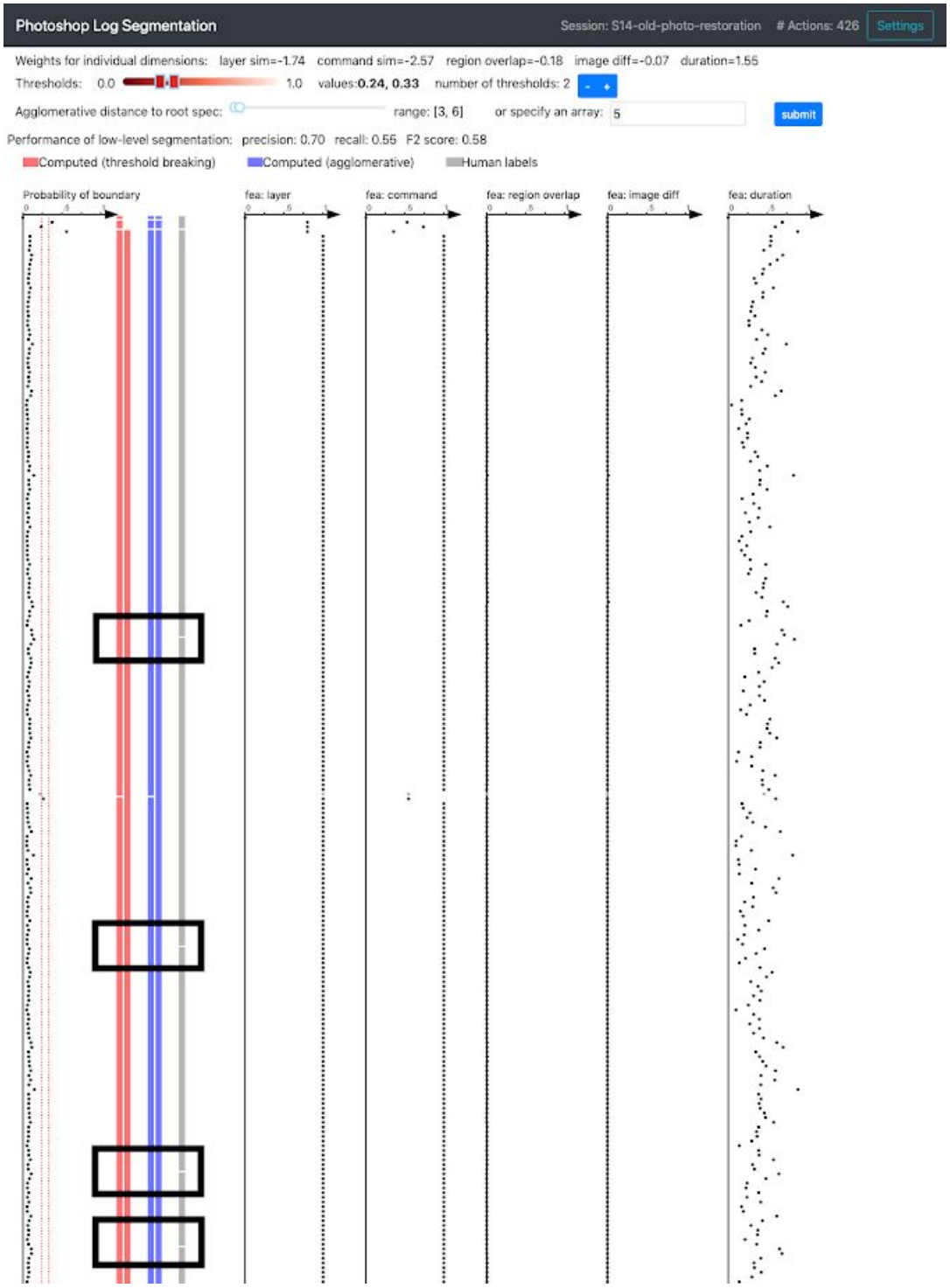


Figure 9: Session S13, special effects task.

Figure 10 shows session S14, a portrait retouching task to restore an old photo. For this session, there are noticeably less chunks in both computed results and human labels. It is also noticeable that the precision is high and recall is low (compared to other sessions). The many missing boundaries (highlighted) happen during the first half of the session, where the participants were fixing cracks on different areas on the photo (upper part → head and face → bottom left → back to face → clothes) but on the same layer. Each event only fixes a small amount of pixels, which leads to the model's failure to capture the change of semantic areas of the photo. In the second half of the session, where the participants were working on the eyes using multiple copy-and-paste phases, the model is able to find true structures that aligned with the human labels.



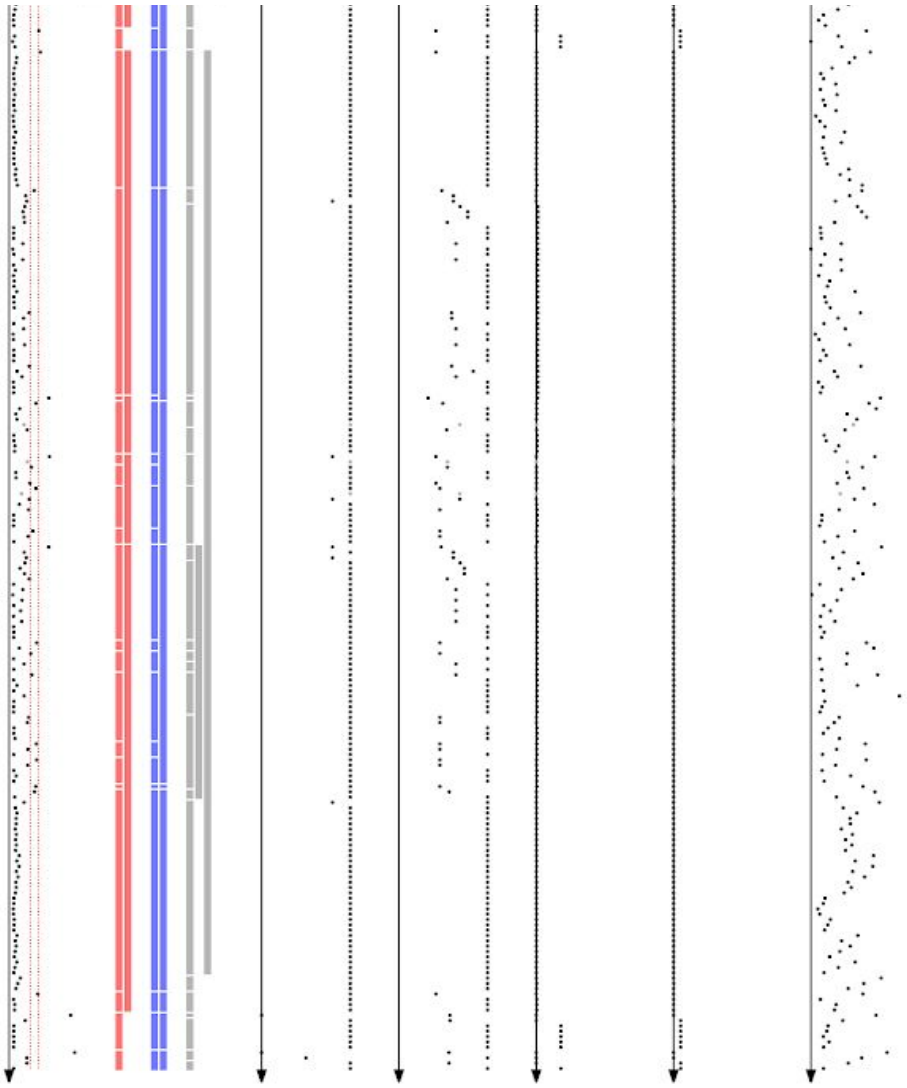


Figure 10: Session S14, portrait retouching task.

Section 5: Rule-based Model

In the early phase of the project, we considered a few rule-based models before coming up with the two-stage model described in the main paper. We present one of them in this section.

In the data collection sessions with Photoshop experts, we observed that when they switched to a new layer or use a different command, it is likely that they finished a subtask and started the next one, especially in poster creation tasks. Therefore, we created a model that segment the event sequence whenever there is a layer switch or command switch. Fig. 11 illustrates the model for session S1, a poster creation session. Note that the visualization is different from the previous figures in this document as they were created in the early phase of this project. Each row represents a layer in the session, and the rectangular blocks represent logged events, sorted by timestamps from left to right. The blue triangles are the ground-truth labels, while the red vertical lines are the predicted boundaries from the rule-based model. We also shows a snapshot image of the current canvas when a "difference score" (computed with layer, command, and image difference) is high from the previous event. We can see that there are many cases of over-segmentation (false positives) and missing boundaries (false negatives). Fig. 12 shows part of a portrait creation session, where the user tends to create less layers and more likely to work on the same layer even if (s)he is doing a different subtask. The errors of the rule-based models are higher in such tasks.

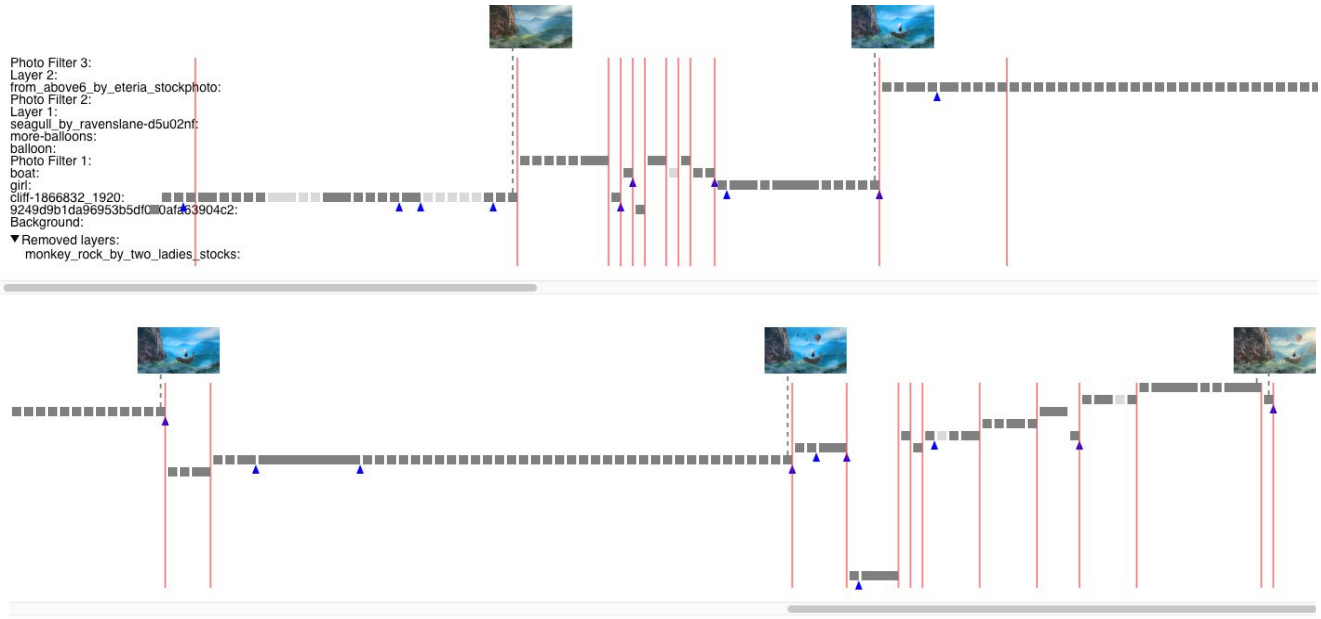


Fig. 11: Illustration of a rule-based model on S1, a poster creation session. The model segments at any events with a layer or command switch. Events are sorted by time and shown from left to right. The top screenshot is the left side of the whole image, the bottom one is the right side.

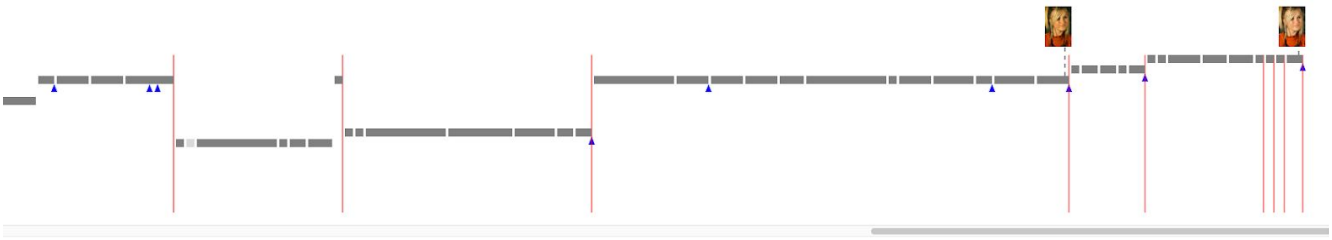


Fig. 12: Partial illustration of rule-based model on S3, a portrait retouching session.