# Comic Frame Segmentation

Sumit Chaturvedi

December 13, 2023

## 1 Introduction

Understanding Comics intersects both Computer Vision and Natural Language Understanding. Yet, both disciplines are ill-poised to study the problem effectively. While Computer Vision primarily focuses on photorealistic scenes, NLU is too tied down to the discrete nature of natural language.

For example, novels are read, one sentence at a time, graphic novels are read, one frame at a time. While delineating sentences is easy, you just have to look up the ASCII code of a *full-stop*, delineating frames is hard.

Yet, this is a crucial sub-problem towards the broader goal of building programs that comprehend comic narratives. This domain comes with a host of sub-problems, such as understanding the relations between characters or how the colors affect the mood of the story (for example, *Ghost World by Daniel Clowes* uses faded bluish colors to suggest adoloscent disillusionment). All these tasks have to work with visual features at some granularity with *frames* being a natural choice.

Therefore, we aim to solve Comic Frame Segmentation. Unlike prior work that assumes axis-aligned frames, our approach considers a variety of shapes. We train an interactive segmentation model, based on Meta's Segment-Anything, to predict frame corners from a query point. Our work utilizes a synthetic, procedurally generated dataset for training and a real-world graphic novel dataset for evaluation. The code, along with instruction on how to run it, is available here.

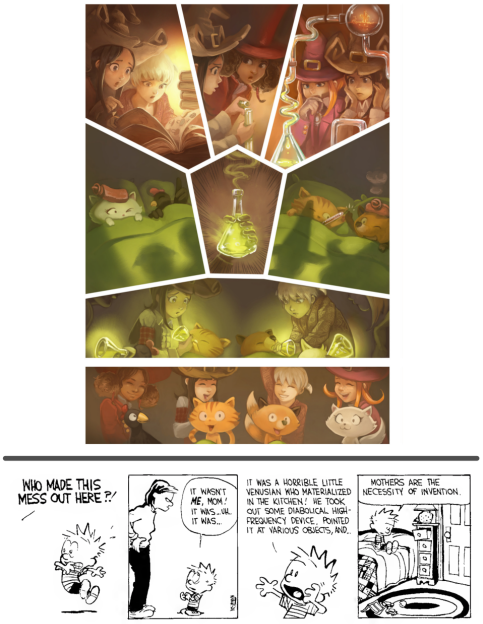


Figure 1: Comic Panel Design Challenges: Top depicts arbitrary polygonal frames from *Pepper and Carrot* by David Revoy Revoy [2023]. Bottom employs traditional axis-aligned bounding boxes but also "negative" frames which don't have a border, rather, are defined by the neighboring context. Such frames are hard to detect by contemporary segmentation models, which are biased towards strong edges (from *Calvin and Hobbes* by Bill Watterson Watterson [1990]).

## 2 Related Work

The recent success of deep learning at digesting large amounts of data offers an alternative way of algorithm design. Before deep learning, people engaged in heuristic design or "feature engineering", now they engage in "data engineering" Ratner et al. [2016].

In "feature engineering", it is often difficult to gauge how a particular design decision will impact performance. This makes "data engineering" an appealing alternative, since often, it is easier to specify how the data will look like. Moreover, guarantees from statistical learning theory Vapnik [1999] ensure that performance improves with increasing data points.

In the context of comic segmentation, many heuristic solutions exist Halford [2023], kum [2023]. However, it is very easy to devise failure modes for them. For example, Halford assumes that frames are rectangular and that there is a gutter between neighboring frames. Therefore, this algorithm is guaranteed to fail in the examples demonstrated in fig. 1.

Taking a leaf from the "data engineering" playbook, we don't attempt to devise heuristics to handle these failure modes, instead, we develop a synthetic comic panel generator. We use this to train a modified SAM model Kirillov et al. [2023], which is the state-of-the-art in interactive segmentation.

## 3 Pepper and Carrot Dataset

To gather the Pepper and Carrot dataset[1], we built a user-interactive polygon segmentation UI was constructed using Label Studio Tkachenko et al. [2020-2022]. This interface facilitated the manual annotation of the vertices of individual frames in the comic panels (which was done by the author).

### 3.1 Dataset Statistics

The statistics of the dataset are summarized in the table below:

[1]The dataset can be accessed in my google drive.

| | |
|---|---|
| # of panels | 264 |
| # of frames | 1184 |
| Frames per panel | $4.84 \pm 1.67$ |
| Points per frame | $4.06 \pm 0.46$ |
| Frame aspect ratio (w/h) | $2.35 \pm 1.60$ |
| % occupancy of frames | $18 \pm 12\%$ |

Table 1: *Statistics of the Pepper and Carrot dataset*: Generally, most frames are quadrilaterals, an assumption that we make in the model. The error bars indicate 1 standard deviation.

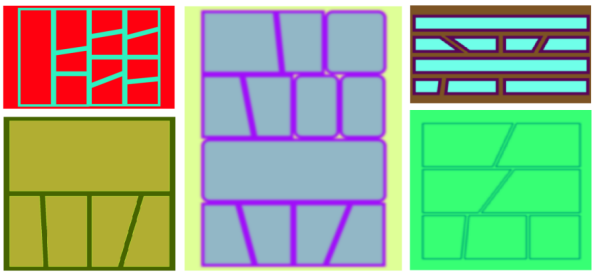

Figure 2: *Procedural Panel Generator v1*: Some examples from the procedural panel generator outlined in algorithm 1. Notice that there is a large gap between the samples produced by this and the comic panels we see (e.g. fig. 1). We need to do further work in order to reduce this gap.

## 4 Methodology

Given the amount of variety in comic book frames, it is unlikely that low-level computer vision heuristics will go a long way in solving the problem in the general case. Alternatively, large scale deep learning methods, such as SAM Kirillov et al. [2023] have shown great promise on challenging tasks like interactive image segmentation, using extremely large datasets. Inspired by these, we opt for a data-driven approach.

Since our Pepper and Carrot dataset is not large enough to finetune the SAM model without overfitting, and constructing a large dataset of comic panels is challenging, we opt to procedurally generate a syn-

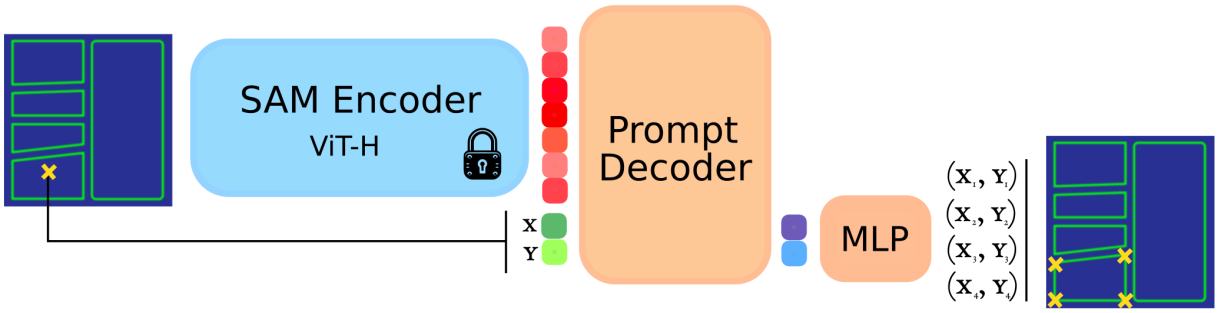


Figure 3: Overview of our model architecture: The input comic panel image is passed to the SAM Encoder (ViT-H variant). The encoded comic panel image and the query point is passed to the prompt decoder (which is a series of Transformer layers). Finally, the query point features are passed through an MLP to predict the corner point coordinates of the comic frame (in clockwise order).

thetic dataset of comic panels (see fig. 2).

Using this dataset, we finetune the SAM ViT-H model (see fig. 3). Specifically, we extract image features from a frozen SAM Encoder. We also extract features for a given query point $(x, y)$. The image features are flattened in scanline-order to form a sequence and the query point features are concatenated along the sequence dimension. The resulting sequence is passed through a trainable prompt decoder (a series of transformer layers). Finally, the features at the query point positions are applied on using a multi-layer perceptron to predict 4 corners of the comic frame.

The 4 ground truth corners are normalized to be within $[-1, 1] \times [-1, 1]$ range. The corners are passed to the model in a canonical order (top-left to bottom-left in clockwise order). This canonicalization is well defined for most of the frames in our evaluation dataset. We noticed that without this step, the network found it difficult to optimize our loss, perhaps due to contrasting supervision signals.

While training, we use the following loss function:

$$L(p_{gt}, \hat{p}) = 0.5L1(p_{gt}, \hat{p}) \quad (1)$$
$$+ 0.25L2(p_{gt}, \hat{p}) + 0.25L_{ro}(p_{gt}, \hat{p}) \quad (2)$$

Here, $L1$ and $L2$ are standard loss functions used in regression. $L_{ro}$ stands for "relative-orientation" loss. Here, we consider edge directions in the ground truth polygon and the predicted polygon and seek to maximize the similarity between the two, using their dot product. Formally, given a set of ground truth edge directions, denoted by $\mathbf{e_j^i}$, and the corresponding predicted edge directions, denoted by $\mathbf{\hat{e}_j^i}$, where $j$ indexes the shape and $i$ indexes the edge within a shape, we aim to minimize the deviation in orientation between the ground truth and the prediction. The loss function for this objective is given by:

$$L_{ro} = \frac{1}{N} \sum_{j=1}^{N} \max_{i=1}^{4} (1 - \mathbf{e_j^i} \cdot \mathbf{\hat{e}_j^i}) \quad (3)$$

In this equation, $N$ denotes the minibatch size. The purpose of the max operation is to consider the largest deviation in orientation for each shape, and then the average of these deviations across all shapes provides the final loss.

In our experience, $L2$ loss is minimized very quickly while still giving poor predictions. Adding $L1$ loss helps with this. We haven't done enough study to justify the $L_{ro}$ loss.

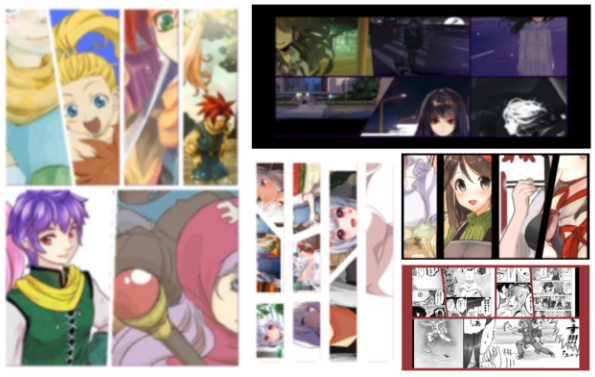

Figure 4: *Procedural Panel Generator v2*: We augment our v1 panel generator with images sourced from the Danbooru dataset. While not completely similar to real comics, they improve our model's performance.

### 4.1 Procedural Panel Generator

We train on two variants of procedurally generated panels. In the first variant, we create empty random boxes. Some examples are visualized in fig. 2 and the generation algorithm is detailed in algorithm 1.

Starting with this base generator, we develop a more sophisticated generator that bridges the gap towards "real-world" comic book panels. Since graphic novels are non-photorealistic and a given panel usually has visually similar images across frames, we download 3.8M images from the 512 px subset from Danbooru Danbooru [2023], a popular anime board. We use CLIP L/14 image encoder Radford et al. [2021] to create an index over them. To simulate visual similarity among images in a panel, we randomly sample one image from the dataset and query $k$ nearest-neighbors (here $k$ is determined by the number of boxes created by the base generator). Once we have this image set, we appropriately crop and composite these images in the boxes. See fig. 4 for visualizations of some training examples.

**Algorithm 1** Generate Comic Panels

**while** True **do**
  Randomly sample aspect ratio, background, border parameters, gutter and margin
  img ← Init with params
  rows, cols ← $\mathcal{U}(\{1,2,3,4\}), \mathcal{U}(\{1,2,3,4\})$.
  boxes ← Calculate box coordinates
  shapes ← Modified boxes with non-axis aligned edges
  layout ← (boxes, shapes)
  Draw shapes and boxes on img
  Randomly apply Gaussian blur to img
  Randomly transpose img and layout.
  **yield** img and layout
**end while**

### 4.2 Training details

We train our model on a single A10 24GB GPU for approximately 2 days. We use Adam Kingma and Ba [2014] optimizer with a constant learning rate of 0.0001. We use batch size of 8. During training, the model sees around 500k synthetically generated panels.

## 5 Experiments

We compare our algorithm against SAM and a heuristic traditional vision based algorithm Halford [2023]. We evaluate both on synthetic and real world comic panels using standard metrics (IoU, PCK@0.1 and L1). With respect to the real world comic panels, we use the Test Set of the Pepper and Carrot dataset (see table 2).

**IoU** simply measures the ratio between area of intersection and union between the ground truth and predicted polygons.

**PCK@0.1** (Percentage Correct Keypoints at Alpha= 0.1) refers to the number of times that the predicted keypoint is within a given radius of the ground truth keypoint. The radius is a product of Alpha and a scale value, typically the length of the diagonal of the prediction range ($2\sqrt{2}$ in our case, since we predict points in the normalized $[-1,1] \times [-1,1]$)

range.

**L1** We match points across two shapes using an $L1$ cost function and report the $L1$ distance between pairs of matched points.

See fig. 6 for qualitative results of our method.

### 5.1 SAM baseline

We run the original SAM model on both our datasets. We follow the same evaluation procedure as for our method i.e. we use the centroid of the ground truth frame polygon as a query point and evaluate the masks using the SAM model. SAM predicts 3 masks for each query point, along with its confidence scores on the result. We use the mask that SAM predicts with highest confidence and calculate it's extent to get an axis-aligned bounding box shape prediction.

Interestingly, SAM does very well on our synthetic dataset (see table 2) but the performance doesn't translate to the Pepper and Carrot dataset. One possible explanation for this (which we haven't confirmed) is that the SAM model is highly biased towards *strong edges*. Pepper and Carrot, unlike the synthetic dataset, has a lot of speech bubbles with strong edges and its likely that SAM picks these up. Without any task-specific training, SAM predicts these instead of the frame, resulting in its poor scores.

### 5.2 Ablations

We compare between our *v1* and *v2* panel generators and show that by qualitatively improving simulation, we are able to achieve better quantitative scores on the real-world Pepper and Carrot dataset (See table 3).

### 5.3 Predicting all frames

Here, we aim to predict all frames *without* any user interactions. We train a version of our model, that for an image and a query point, predicts the confidence score of the query point being near the centroid of the frame polygon along with a polygon itself.

During inference, we uniformly sample query points on the input image. Our model predicts confidence scores and polygons for all of these points. The top-k polygons are selected on the basis of the confidence scores. We apply Mean Shift Clustering on these polygons and discard clusters below a threshold cluster size.

Even though we typically sample 900 query points, the inference step still takes only 2-3 s (on A10 GPU) because we cache the heavy image encoder evaluation and only evaluate the lightweight prompt decoder for different query points. Qualitative results of this approach are demonstrated in fig. 5. Also see fig. 8 for visualization of query point confidence scores.

## 6 Limitations

A major limitation of our method is interactivity which makes it hard to use it for automatic exploration of comic books. While we have presented a version that doesn't require interactivity (see section 5.3), it is not very robust. Moreover, it comes with hyper-parameters that need tuning for it to work at all. See fig. 7 for failure cases of both versions of our model.

## 7 Conclusion

We present a method to interactively segment comic frames. Our method doesn't require extensive data annotation. Instead, we carefully build a simulator for our domain and train a neural network on the simulated, synthetic data. We find that the trained model is able to generalize to "real-world" comics (both qualitatively and quantitatively).

Finally, we remove interactivity by adding a query point confidence score predictor module, which is tasked to identify centroids of frames. We qualitatively demonstrate its performance and discuss its limitations.

## References

Kumiko, the comics cutter. `https://github.com/njean42/kumiko`, 2023. GitHub repository.

| | Pepper and Carrot | | | Synthetic | | |
|---|---|---|---|---|---|---|
| Method | IoU ↑ | PCK@0.1 ↑ | L1 ↓ | IoU ↑ | PCK@0.1 ↑ | L1 ↓ |
| SAM | $0.42 \pm 0.21$ | $0.52 \pm 0.26$ | $0.37 \pm 0.19$ | $0.81 \pm 0.15$ | $0.94 \pm 0.14$ | $0.08 \pm 0.11$ |
| Halford | $\mathbf{0.93} \pm 0.12$ | $0.96 \pm 0.10$ | $\mathbf{0.04} \pm 0.08$ | $0.47 \pm 0.32$ | $0.61 \pm 0.30$ | $0.47 \pm 0.37$ |
| Ours | $0.88 \pm 0.07$ | $\mathbf{0.98} \pm 0.07$ | $0.05 \pm 0.03$ | $\mathbf{0.88} \pm 0.06$ | $\mathbf{0.99} \pm 0.04$ | $\mathbf{0.03} \pm 0.03$ |

Table 2: *Performance metrics for Halford and our method on Test Set of Pepper and Carrot (238 panels) and Synthetic Dataset (1,000 panels)*: Using Hungarian algorithm we match Halford's predicted boxes to ground truth boxes. Unsurprisingly, on the Synthetic dataset, our method has stronger performance. Our goal for this project is to improve our model on both datasets. Error bars denote 1 standard deviation.

| | Pepper and Carrot | | |
|---|---|---|---|
| Method | IoU ↑ | PCK@0.1 ↑ | L1 ↓ |
| Ours (v1) | $0.46 \pm 0.13$ | $0.53 \pm 0.29$ | $0.32 \pm 0.16$ |
| Ours (v2) | $\mathbf{0.88} \pm 0.07$ | $\mathbf{0.98} \pm 0.07$ | $\mathbf{0.05} \pm 0.03$ |

Table 3: *Performance comparison between v1 and v2 synthetic datasets*: On the Test Set of Pepper and Carrot, consisting of 238 panels. Simply improving the simulation of comic book panels dramatically improves performance across all metrics (see fig. 2 and fig. 4 for qualitative visual comparison). Error bars denote 1 standard deviation.


Danbooru. Danbooru: Anime image board. `https://danbooru.donmai.us/`, 2023. Accessed: December 12, 2023.

Max Halford. Comic book panel segmentation, 2023. URL `https://maxhalford.github.io/blog/comic-book-panel-segmentation/`. Accessed: 27-10-2023.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems*, 29, 2016.

David Revoy. *Pepper and Carrot*. Open Books, 2023.

Maxim Tkachenko, Mikhail Malyuk, Andrey Holmanyuk, and Nikolai Liubimov. Label Studio: Data labeling software, 2020-2022. URL `https://github.com/heartexlabs/label-studio`. Open source software available from https://github.com/heartexlabs/label-studio.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

Bill Watterson. *Calvin and Hobbes*. Comic Press, 1990.

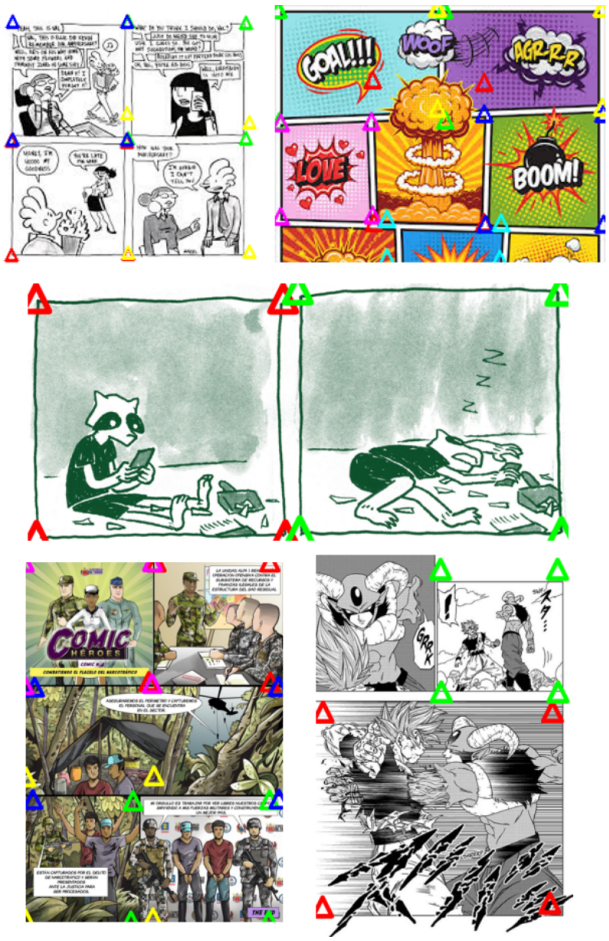


Figure 5: *Predicting all frames automatically*: By predicting confidence scores to predict a frame anchor point, we are able to automatically detect all frames in the panel. Notice that our model is able to detect frames when they are packed densely, and even when the lines between frames are blurred (top right).

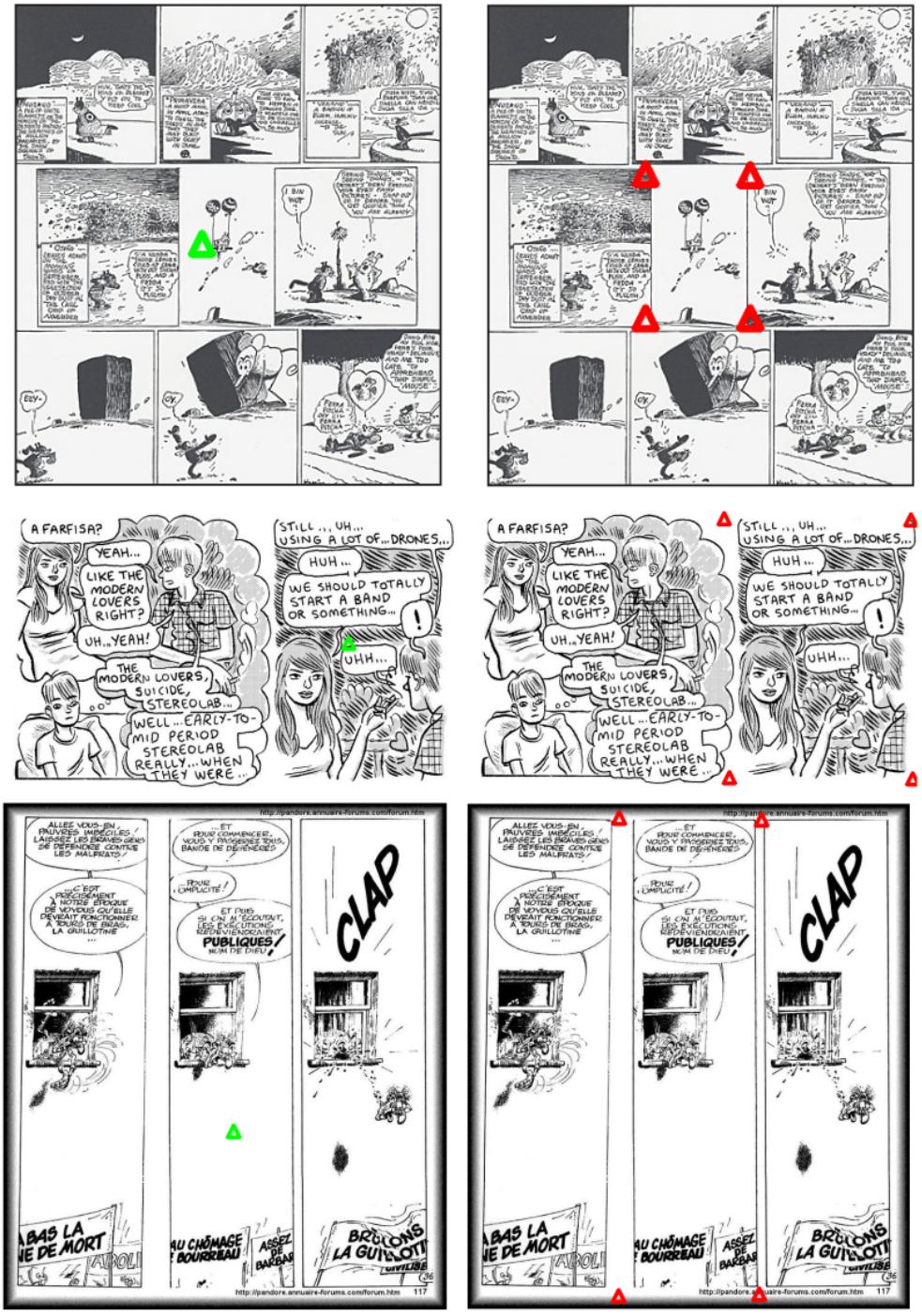


Figure 6: *Qualitative results of interactive frame prediction*: The query point is in green and the points of the predicted shape are in red. Notice that our method is able to predict accurate frame shapes in diverse scenarios, even when the frame is realized, not through its own boundary but through the boundary of other frames (top row) and when there is no discontinuity between frames (middle row).

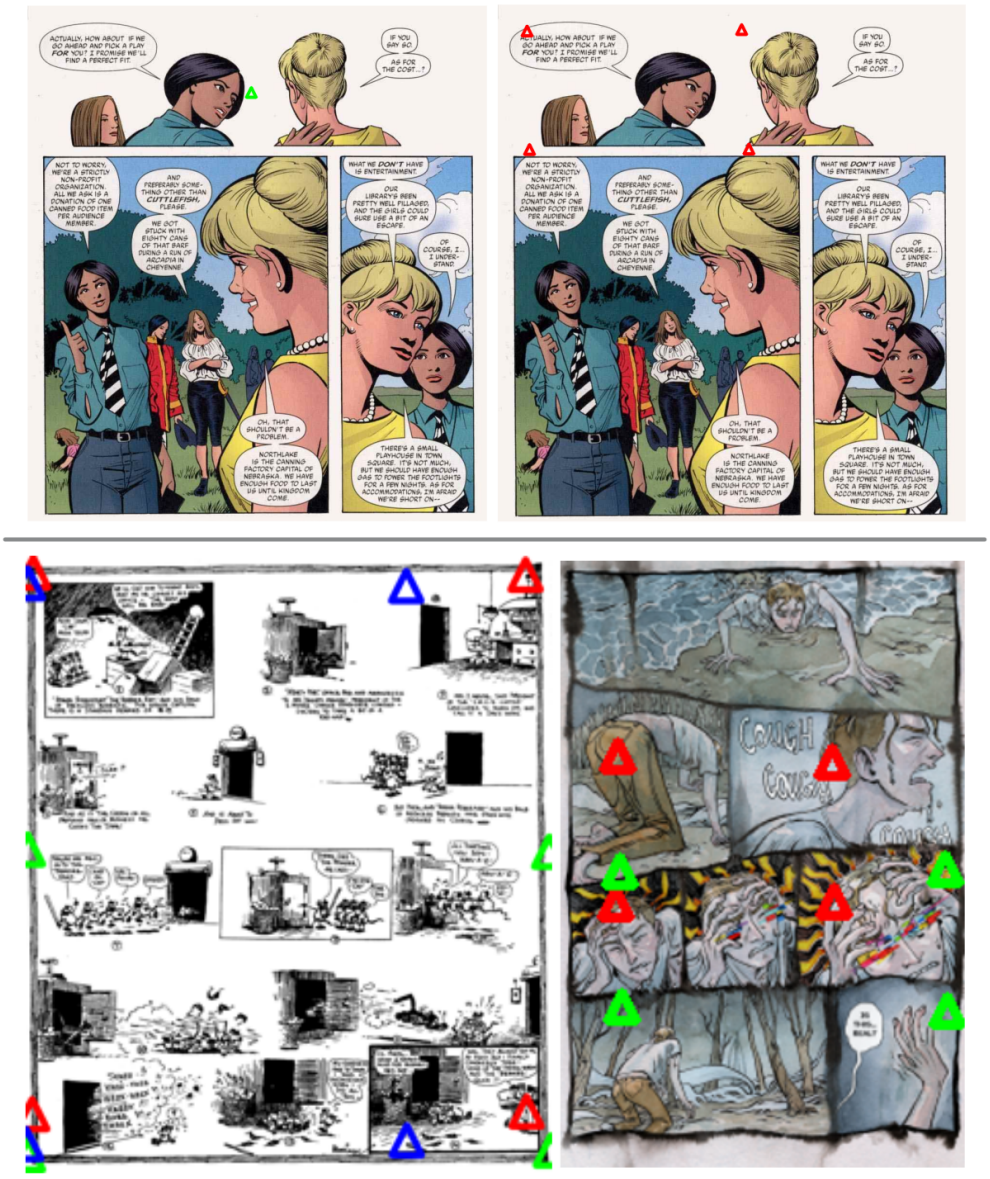

Figure 7: *Failure Cases*: We show failure cases of both our interactive (top) and automatic (bottom) methods. In the top panel, the frame's boundary is non-existent and our method doesn't reliably place the points where we'd expect them i.e. right across that row. In the bottom panels, we illustrate two hard cases where the automatic method fails to do anything meaningful.


Figure 8: *Visualization of our model's confidence scores for query points*: Here, we sample a 40 by 40 grid of query points on the comic panel (left) and plot confidence scores (right), with high confidence values in *white* and low confidence values in *black*. During training, the model is supposed to predict high confidence for points near the centroid of each frame and low confidence elsewhere.