Baltic J. Modern Computing, Vol. 14 (2026), No. 2, pp. 308–327
https://doi.org/10.22364/bjmc.2026.14.2.03

InstanceSketch: Robust Sketch Instance Segmentation with Cluster Refinement

Yana SVIRHUNENKO, Yaroslav TERESHCHENKO, Pavlo TYTARCHUK

Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

yana.svirhunenko@knu.ua, y.tereshchenko@knu.ua, pavlo.tytarchuk@knu.ua

ORCID 0009-0002-2300-0655, ORCID 0000-0002-8451-7634, ORCID 0009-0006-4783-1946

Abstract. Stroke segmentation is crucial for interpreting free-hand sketches, supporting tasks ranging from recognition and generative modeling to semantic analysis. However, existing methods often struggle to distinguish between similar component categories and typically rely on large models unsuitable for on-device deployment.
In this paper, we present *InstanceSketch*, a two-step pipeline designed to segment complex sketch objects, and *InstanceSketch-Scene*, an extension that decomposes entire scenes containing multiple objects into semantically meaningful parts. By integrating convolutional neural networks and Transformers, our approach enables efficient breakdown of sketches into interpretable components. An important advancement is a novel method for refining labels of similar components using clustering algorithms to ensure accurate separation between visually similar stroke groups. Our proposed method not only matches but also exceeds the performance of existing state-of-the-art techniques, while operating efficiently with significantly fewer resources, making it an attractive solution for on-device applications and serverless environments.

Keywords: stroke segmentation, sketch analysis, deep learning, transformers, clustering

1 Introduction

Free-hand sketches, despite their abstract nature, follow an underlying semantic structure where each stroke or group of strokes represents a distinct and meaningful part of the object. The stroke segmentation task typically involves assigning a unique semantic label to each stroke to determine the component to which it belongs (see Figure 1). Understanding and segmenting these components is crucial for a wide range of applications, including sketch-based retrieval, generative modeling, and human-computer interaction. Despite being well-studied, stroke segmentation remains challenging due to the varying levels of abstraction, diverse depiction styles, and the inherent sparsity and noise in free-hand sketch data.

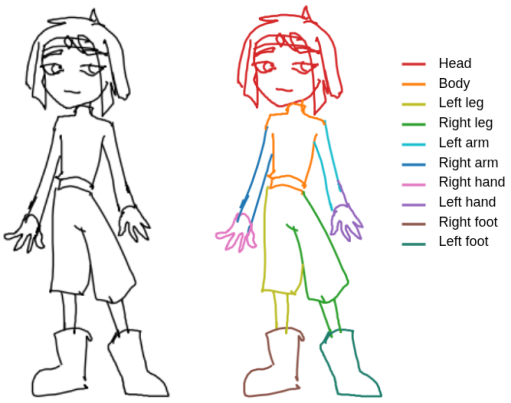

Fig 1. Visualization of the stroke segmentation task. The figure shows an input sketch (left) and the corresponding color-coded stroke labels. Example drawn from the authors' custom dataset.

Existing stroke segmentation techniques generally fall into two broad categories: rule-based and learning-based approaches. Early rule-based methods utilized handcrafted geometric heuristics—such as stroke proximity, continuity, and curvature—to group strokes into semantically meaningful parts (Sun et al., 2012; Stahovich, 2004; Herold and Stahovich, 2011). While conceptually simple, these methods lack robustness when applied to the high variability and abstraction of freehand sketches.

Recent research has turned to deep learning-based methods. For instance, SketchGNN (Yang et al., 2021) models a sketch as a graph, where nodes represent points, and edges encode stroke connectivity, and applies graph convolutions to predict point-level semantic labels. While it achieves improved segmentation accuracy, it struggles with boundary precision in sketches with high abstraction. S3NN (Zhang et al., 2022) introduces a hybrid CNN-RNN-GNN architecture that combines visual context, sequential stroke information, and spatial refinement. Although effective on complex scene-level sketches, it incurs high computational costs and slow inference times.

Other learning-based strategies include the dual-CNN approach (Zhu et al., 2018), which uses two CNN streams to extract stroke-wise features for part-level sketch segmentation. CNN-CRF models (Zhu et al., 2020), which leverage Conditional Random Fields (CRFs) to refine CNN outputs by modeling spatial dependencies for enhanced semantic labeling, have also been proposed. However, these approaches rely heavily on specific stroke-level features, which limits their generalization across diverse sketching styles. RNN-based models such as SketchSegNet (Wu et al., 2018) treat stroke segmentation as a sequence-to-sequence problem, using RNNs to encode the stroke sequence and decode semantic labels. Yet, this method is sensitive to variations in drawing order, a common challenge in unconstrained sketch input.

Recent advancements in deep learning have led to the popularization of Transformer-based architectures for sketch understanding, particularly due to their ability to model long-range dependencies between disparate strokes. One notable example is the Sketch-

ESC model (Zhu et al., 2024), which utilizes a Transformer backbone with a semantic component-level memory module. This module maintains a learnable memory bank containing prototypical stroke group representations learned from training data, enabling the model to focus on key structural elements within each sketch category. However, its reliance on a Long Short-Term Memory (LSTM)-based stroke embedding extractor reduces effectiveness when dealing with long or highly variable stroke sequences.

Another prominent architecture is ContextSeg (Wang and Li, 2024), which combines a CNN-based stroke encoder with an autoregressive Transformer; however, this mechanism is prone to error accumulation and results in slower inference times.

HCTSketch (Svirhunenko et al., 2025) combines a CNN for feature extraction with a Vision Transformer (ViT) (Dosovitskiy et al., 2021) to enable simultaneous stroke segmentation, addressing limitations of earlier methods. However, it often struggles with visually similar components, frequently mislabeling or mixing them.

Despite notable progress, a clear research gap remains: existing methods commonly fail to distinguish between structurally symmetric or visually similar components (e.g., left vs. right limbs of an animal), rely on heavyweight architectures with tens of millions of parameters that are unsuitable for on-device deployment, and assume single-object inputs, which limits their applicability to real-world scene-level sketches.

The central research question is: *Can a lightweight two-stage pipeline combining CNN and Transformer encoders with cluster-based label refinement match or exceed the segmentation accuracy of larger, established models?* The aim of this paper is to develop a lightweight yet accurate stroke segmentation pipeline that overcomes these shortcomings, using a methodology that integrates convolutional feature extraction, Transformer-based stroke encoding, and unsupervised clustering for refinement. Specifically, the research objectives are:

1. To design a two-stage pipeline that combines an initial stroke classification model with a cluster-based refinement step capable of resolving ambiguity among visually similar components without requiring an explicit count of those components.
2. To demonstrate that competitive segmentation accuracy can be achieved with significantly fewer parameters than existing methods, making the pipeline suitable for resource-constrained environments.
3. To extend the single-object pipeline to multi-object scene sketches by incorporating an object detection stage that isolates individual objects before segmentation.

The experiments are conducted on a custom dataset combining manually annotated sketches with images drawn from the publicly available Quick, Draw! dataset (Ha and Eck, 2018), covering seven object categories.

To achieve these objectives, we present *InstanceSketch*, a novel two-stage pipeline for stroke segmentation in freehand sketches that we propose and develop entirely in this work. In contrast to other methods that rely solely on direct label assignment, the final stage of our pipeline refines labels by learning discriminative stroke embeddings that are then used for clustering, allowing the method to adapt to cases where the exact number of components is unknown.

We also present *InstanceSketch-Scene*, a scene-level extension of *InstanceSketch* developed as a further original contribution, which prepends a YOLO-based detection

stage to isolate individual objects and infer their classes, used as context to guide per-object stroke segmentation (Section 7.1).

Both *InstanceSketch* and *InstanceSketch-Scene* achieve accurate and efficient performance, which is essential for applications ranging from digital art editing to autonomous sketch interpretation systems.

2 Proposed Method

Given an input sketch composed of separate strokes represented as point sequences, our goal is to assign a semantic label to each stroke, indicating the object part or component to which it belongs. Our method introduces a robust pipeline designed to segment free-hand sketches, which consists of two stages:

1. **Initial Stroke Segmentation.** Utilizes a model comprising CNN and Transformer blocks. CNNs are effective at extracting features from strokes represented as 2D images, while Transformer encoders leverage attention mechanisms to capture relationships between different strokes. Finally, a classification head is used to get the initial label for each stroke.
2. **Stroke Label Refinement.** Enhances the initial segmentation by refining labels for challenging cases where similar-looking components, like different limbs of an animal, might be hard to distinguish. Uses a similar architecture (CNN followed by Transformer) to obtain stroke embeddings that can be used for clustering in high-dimensional space. This allows us to separate strokes that have the same initial label but belong to different structural components.

The diagram of the proposed method is presented in Figure 2.

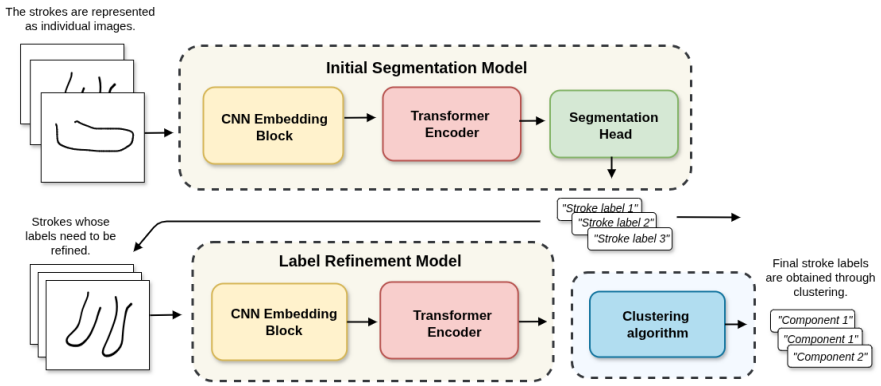

Fig 2. Diagram of the proposed processing pipeline.

2.1 Segmentation Model

To perform the initial segmentation of each stroke within the sketch, we generate two distinct image representations:

1. In the first format, each individual stroke is centered and scaled to occupy the entire image canvas. This normalization enables the model to focus on the stroke's overall shape, while preserving fine-grained details within a compact input size, thus maintaining computational efficiency. This enables efficient representation even for short stroke segments.
2. In the second format, each stroke is rendered at its original position and scale within the image space. When these images are stacked, the original sketch can be reconstructed. This representation allows the model to learn spatial context, including the stroke's position and size within the sketch. Empirically, this approach proved more effective than explicitly providing bounding box coordinates as additional input.

Each pair of stroke images is initially processed through a convolutional module to extract embeddings. The architecture employs a dual-branch design, where two parallel convolutional pipelines independently encode shape and spatial representations of each stroke. Each branch consists of multiple convolutional blocks, each comprising 2D convolutional layers, batch normalization, ReLU activations, and max pooling. These vectors are then fused via a small multi-layer perceptron (MLP) to form a unified embedding that captures both geometric and positional characteristics.

After obtaining stroke embeddings, we fuse them with order embeddings using linear layers to incorporate information about the stroke drawing sequence. The Transformer block employs multi-head self-attention to model long-range dependencies between strokes. After processing the stroke embeddings with the Transformer encoder, they are passed through an MLP that performs stroke-wise segmentation, predicting a label for each stroke.

Since different stroke types occur with varying frequencies, we compensate for this imbalance during training by applying class-specific weights in the cross-entropy loss. To compute these weights, we first calculate the ratio between the median class frequency and each individual class frequency. We then apply a normalization by taking the square root of each value. Finally, the weights are normalized to have a mean of one before being used in the loss function.

The weighted cross-entropy loss is defined as

$$\mathcal{L}_{\text{seg}} = -\sum_{i=1}^{n} w_{s_i}\, s_i \log z_i, \tag{1}$$

where s_i is the true class label of sample i, z_i is the predicted probability of the correct class, and w_{s_i} is the weight associated with class s_i.

The class weights are computed as

$$w_s = \frac{r_s}{\frac{1}{n}\sum_{j=1}^{n} r_j}, \quad r_s = \sqrt{\frac{\text{median}(\{f_k\})}{f_s}}, \tag{2}$$

where f_s is the frequency of class s, median$(\{f_k\})$ is the median of all class frequencies, and n is the total number of classes. This weighting scheme increases the contribution of underrepresented classes while reducing that of more frequent classes.

After segmentation, we obtain initial stroke embeddings. Strokes requiring finer class distinctions undergo additional processing during the Refinement step.

Figure 3 shows a detailed diagram of the segmentation model.

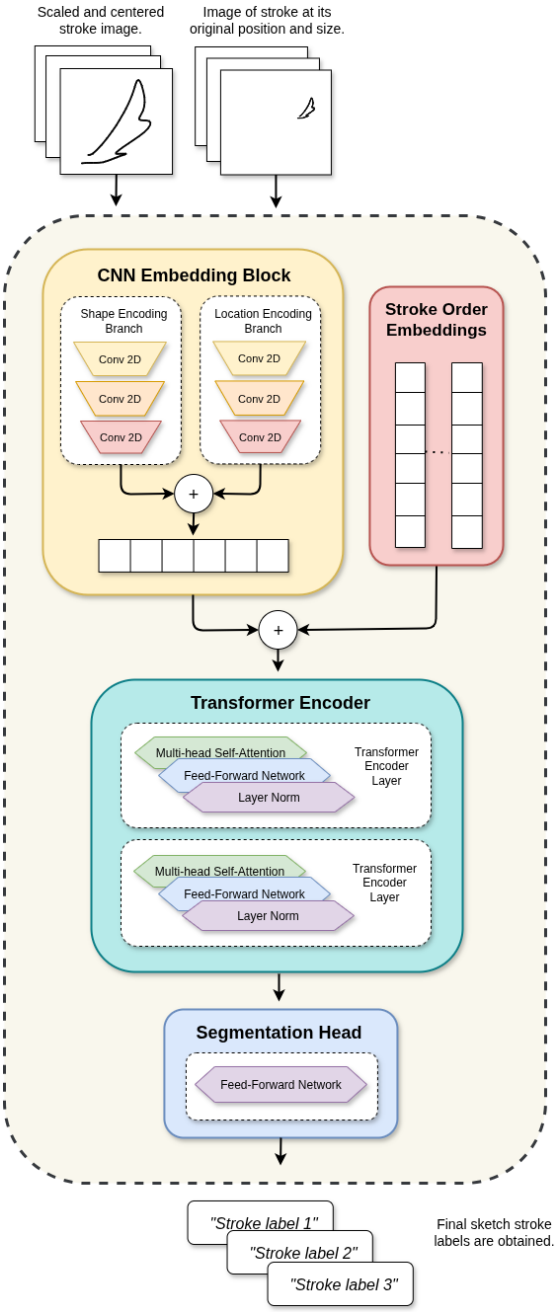

Fig 3. Diagram of the Segmentation Model.

2.2 Refinement Model

While developing a model for stroke segmentation and experimenting with various architectural approaches, we encountered a notable challenge. Although many existing methods perform well when distinguishing strokes that belong to clearly different components, their performance degrades significantly when attempting to classify strokes that are part of similar or symmetric components.

Consider the case of segmenting sketches of animals. Identifying which strokes correspond to limbs in general is relatively straightforward. However, assigning distinct labels to strokes representing different limbs—such as distinguishing the left leg from the right—proves much more difficult. This is because such strokes often appear in highly similar visual and spatial contexts, making them hard to classify directly even for sophisticated models.

Another notable characteristic of sketch data is the high degree of creative freedom and, in some cases, unconventional design. For example, consider a vehicle sketch where the task is to segment all strokes forming the wheels. In certain applications, it may also be necessary to distinguish strokes belonging to individual wheels as separate components. However, due to variability in vehicle types (e.g., a bicycle with two wheels, a car with four, or a fantastical vehicle with N wheels), it becomes impractical to assign labels through direct classification, as the exact number of such components is unknown in advance.

To address this, we introduce a simplifying assumption: instead of assigning semantically specific, predefined labels (e.g., "front left wheel"), we use neutral identifiers such as "wheel 1," "wheel 2," ..., "wheel N." This strategy reduces unnecessary complexity during training and better reflects the inherent ambiguity present in abstract or minimalist sketches.

Instead of direct stroke classification, we decided to employ a clustering approach that would divide strokes into the components they belong to. To produce stroke representations that can be used for clustering, we use a Refinement Model similar to the segmentation step that consists of a convolutional neural network, Transformer encoder, and small multilayer perceptron.

We plot each stroke on a 2D image at its original position and scale within the image space. Unlike the Segmentation Model, no shape representation is needed, as our focus is solely on the relative location of strokes. Stroke images are first processed by a CNN module composed of 2D convolutional layers, ReLU activations, and max pooling operations to extract initial embeddings. Each stroke's initial label obtained during the previous segmentation step is encoded as contextual information and fused with the stroke embeddings and their drawing order embeddings using linear layers. This ensures that the model considers the general component type before attempting to divide it into individual instances. The obtained embeddings are then passed through a Transformer encoder, which employs multi-head self-attention. Finally, a small MLP is used to generate the final stroke representations for clustering tasks.

The model is trained using the triplet loss function (Schroff et al., 2015). This loss function is used to train models to learn meaningful embeddings by comparing three samples at once: an anchor, a positive sample (same class as anchor), and a negative sample (different class). The goal is to minimize the distance between the anchor and

positive while maximizing the distance between the anchor and negative, which encourages the model to produce embeddings for strokes of similar classes closer in high-dimensional space than those from different parts. The loss is computed as:

$$\mathcal{L}_{\text{triplet}} = \frac{1}{n}\sum_{i=1}^{n} \max\left(\|a_i - p_i\|_2^2 - \|a_i - n_i\|_2^2 + m,\ 0\right) \tag{3}$$

where a_i is the anchor sample, p_i is the positive sample (same class as anchor), n_i is the negative sample (different class from anchor), m is the separation margin between clusters, n is the number of triplets, and $\|x - y\|_2$ denotes the Euclidean distance between vectors x and y.

These representations can then be used for clustering using methods like K-Means clustering if the goal is to divide strokes into a specific number of components, or use unsupervised clustering algorithms if we do not know the number of components beforehand.

3 Dataset Details

For training and testing, we gathered a custom dataset. A portion of the images was sourced from the Quick, Draw! dataset (Ha and Eck, 2018), while the rest were manually collected and annotated by our team to ensure a diverse range of drawing styles—from simple sketches to detailed, realistic depictions. This was essential because existing datasets lacked the stroke-level annotations required for our task. The final dataset includes seven classes: four-legged creatures, two-legged creatures, sun, cloud, car, tree, and flower. Examples of images from the collected dataset are shown in Figure 4.

Fig 4. Examples of sketch instances from the dataset.

Each class is annotated with different stroke labels. For example, the four-legged creature class can include up to eight stroke types (e.g., head, body, tail). The actual number of stroke types in an image may vary depending on which components are present.

To enhance model generalization, we apply a range of augmentation techniques during preprocessing. These augmentations are performed directly on the stroke coordinates before rendering, ensuring the strokes remain visible. The applied techniques include:

- **StrokeDropout (Random Stroke Removal):** Selectively removes certain strokes, usually the shortest to preserve important information.
- **Random Scaling:** Alters the sketch's dimensions along one or more axes to introduce shape diversity.
- **Random Rotation:** Rotates sketches by small angles to account for orientation differences.
- **Random Horizontal Flip:** Mirrors sketches along the vertical axis (i.e., flips the sketch left-to-right).
- **Random Stroke Transformations:** Applies stroke-level modifications—including random scaling, rotation, and directional shifts—individually to each stroke, rather than to the entire sketch at once.
- **Gaussian Noise:** Adds random Gaussian noise to the stroke coordinates, followed by smoothing.

These augmentations significantly reduced overfitting during training, demonstrating their effectiveness and improving the model's robustness and its ability to generalize across different sketch styles.

Table 1 provides an overview of the distribution of training and validation samples across different sketch classes. The numbers in the training set reflect the data after balancing and augmentation, while the validation set contains only the original, unaltered images.

Table 1: Number of training and test instances per category in the dataset.

Category	Train Instances	Test Instances
Car	6994	569
Cloud	3186	116
Flower	3798	179
Sun	3702	307
Tree	4179	348
4-Legged Animal	5406	436
2-Legged Animal	3276	181

4 Experiments

4.1 Implementation Details

The model pipeline consists of two models that are trained separately. The training process employs the Adam optimizer, along with an adaptive learning rate scheduler, to promote stable convergence. To mitigate overfitting, dropout regularization is incorporated during training. All models were trained for 150 epochs with a batch size of 128. During training, different model sizes were tested. At the end, we chose parameters that gave the optimal accuracy to model size ratio.

The Segmentation Model employs a CNN-based embedding module that processes both stroke shape and location images, each of size 32×32 with a single channel. These inputs are handled by two separate CNN branches with 3×3 convolutions. The number of feature channels increases from 32 to 64 and then to 128 before applying adaptive average pooling. The resulting features are fused into embeddings of size 128.

In this setup, position embeddings are also 128-dimensional. We use a Transformer encoder with 2 layers, 4 attention heads, and a hidden size of 128—providing a balance between efficiency and model capacity. After encoding, layer normalization is applied, followed by an MLP that maps the refined stroke embeddings to the final stroke labels.

The complete model consists of approximately 665,000 parameters.

The Refinement Model takes as input images that depict strokes in their original spatial positions. Each image is 32×32 pixels with a single channel. These inputs are processed by a CNN with 3×3 convolutional layers. During this process, the number of feature channels increases to 32. The resulting feature maps are then passed through a linear layer to generate embeddings of size 64.

These embeddings are fused with encoded stroke labels and order representations of the same size, and then projected to a 64-dimensional space. Then they are processed by a Transformer encoder module comprising 2 layers, 4 attention heads, and a hidden size of 64. Subsequently, layer normalization is applied. Finally, the normalized embeddings are refined using an MLP, producing the final stroke embeddings of size 32, which can be used for downstream clustering tasks.

The complete model consists of approximately 282,000 parameters.

Our dataset includes detailed stroke-level annotations for four-legged and two-legged creature classes in particular, with a consistent set of limb labels applied across both. In the case of four-legged creatures, each limb is annotated with fine-grained distinction between front and back, as well as left and right sides. This results in eight unique labels:

1. ‘front-right-foot’
2. ‘front-left-foot’
3. ‘front-right-leg’
4. ‘front-left-leg’
5. ‘back-right-foot’
6. ‘back-left-foot’
7. ‘back-right-leg’
8. ‘back-left-leg’

For two-legged creatures, the same label set is used, though with slightly different semantics: the “front” limbs correspond to the upper limbs (arms), and the “back” limbs refer to the lower limbs (legs). In this context, “leg” and “foot” in the upper limbs

refer to the arm and hand, respectively, while in the lower limbs they retain their usual meaning.

While all other labels are classified solely by the Segmentation Model, these specific limb labels are used to train and evaluate our Refinement Model. Initially, our segmentation model classifies each stroke as belonging to either the front or back pair of legs, as well as its type (leg or foot), since this distinction can be made more reliably based on spatial location.

The refinement model then processes the front and back limb stroke groups separately and generates embeddings, which will be used to separate the strokes into distinct legs (left or right).

To assess the separation accuracy, we compare the model's output with the initial annotations. Specifically, we measure how frequently strokes originally labeled with the same orientation (left or right) were correctly grouped into the same limb.

Since here we used the model on leg pairs, as a clustering algorithm we have selected the K-Means algorithm with 2 clusters.

4.2 Performance Metrics

By combining all modules, we obtain a robust and efficient pipeline with approximately 947,000 parameters. After training, we evaluated the pipeline on the validation set using key metrics such as classification and segmentation accuracy and F1-scores.

We also compared our method with other Transformer-based sketch segmentation approaches, including SketchESC (Zhu et al., 2024) and HCTSketch (Svirhunenko et al., 2025). The implementations of SketchESC and HCTSketch were obtained from their respective publicly available GitHub repositories and evaluated without modification using the same validation split. All models were evaluated on the full dataset, considering all sketch categories simultaneously.

To ensure a fair comparison in segmentation accuracy for refined labels, we assess how well each model separates strokes into distinct limbs by considering alternative limb orientations, rather than directly comparing predicted labels to the initial ground truth. Specifically, we evaluate both the original and mirrored versions of the predicted leg labels—where left and right are swapped—and use the better of the two assignments for final evaluation of all models. This approach accounts for directional ambiguity and provides a more robust measure of limb-level separation performance.

Table 2 presents the performance comparison of all three models.

Table 2: Validation performance comparison across models. **InstanceSketch** achieves the highest scores across all metrics.

Model	**Segmentation Accuracy (%)**	**Segmentation F1-Score (%)**
SketchESC	85.43	80.07
HCTSketch	91.17	87.04
InstanceSketch	**93.41**	**90.70**

Our proposed InstanceSketch achieves the highest scores across both metrics, reaching 93.41% segmentation accuracy and 90.70% segmentation F1-score. Compared to HCTSketch, this represents a gain of 2.24 percentage points in accuracy and 3.66 percentage points in F1-score. Relative to SketchESC, the improvement is even larger: 7.98 percentage points in accuracy and 10.63 percentage points in F1-score. These results demonstrate that InstanceSketch not only outperforms both baselines but also provides a substantial improvement in segmentation quality while maintaining a low parameter count.

Inference times for all models—InstanceSketch, HCTSketch (Svirhunenko et al., 2025), and SketchESC (Zhu et al., 2024)—were measured on the validation set using an NVIDIA GeForce RTX 3090 Graphics Processing Unit (GPU) with a batch size of one. The reported times represent the average processing duration per sample across the entire set.

A detailed comparison of the performance of all three models is provided in Table 3.

Table 3: Comparison of model size and inference time for different methods.

Method	**Parameters (Millions)**	**Inference Time (s)**
SketchESC	95.2	0.0118
HCTSketch	7.9	0.0016
InstanceSketch	**0.9**	**0.0029**

Although InstanceSketch has significantly fewer parameters compared to both HCTSketch (Svirhunenko et al., 2025) and SketchESC (Zhu et al., 2024), its inference time is slightly slower than HCTSketch. This is primarily due to its pipeline design, which involves two separate models. Nevertheless, InstanceSketch achieves a much smaller model size while maintaining competitive execution speed.

To assess the contribution of each component in our pipeline, we conducted an ablation study by removing the refinement step and training the segmentation model to classify all strokes jointly. Without the refinement step, InstanceSketch achieved 91.98% segmentation accuracy and 88.39% F1-score. This corresponds to a drop of 1.43 percentage points in accuracy and 2.31 percentage points in F1-score compared to the full model, highlighting the positive impact of the refinement step on segmentation performance.

4.3 Qualitative Analysis

Compared to both HCTSketch (Svirhunenko et al., 2025) and SketchESC (Zhu et al., 2024), our model demonstrates a superior ability to distinguish between elements of different legs, resulting in cleaner and more consistent segmentation. For instance, Figure 5 shows an example of four-legged animal segmentation by different models. Other approaches often confuse the strokes that form the legs, whereas the Refinement step in our approach resolves this problem. Figure 6 presents confusion matrices for leg segmentation in the dataset specifically. InstanceSketch achieves the lowest error rate.

Beyond this improvement, our approach also achieves higher overall segmentation accuracy across a wide range of stroke types.

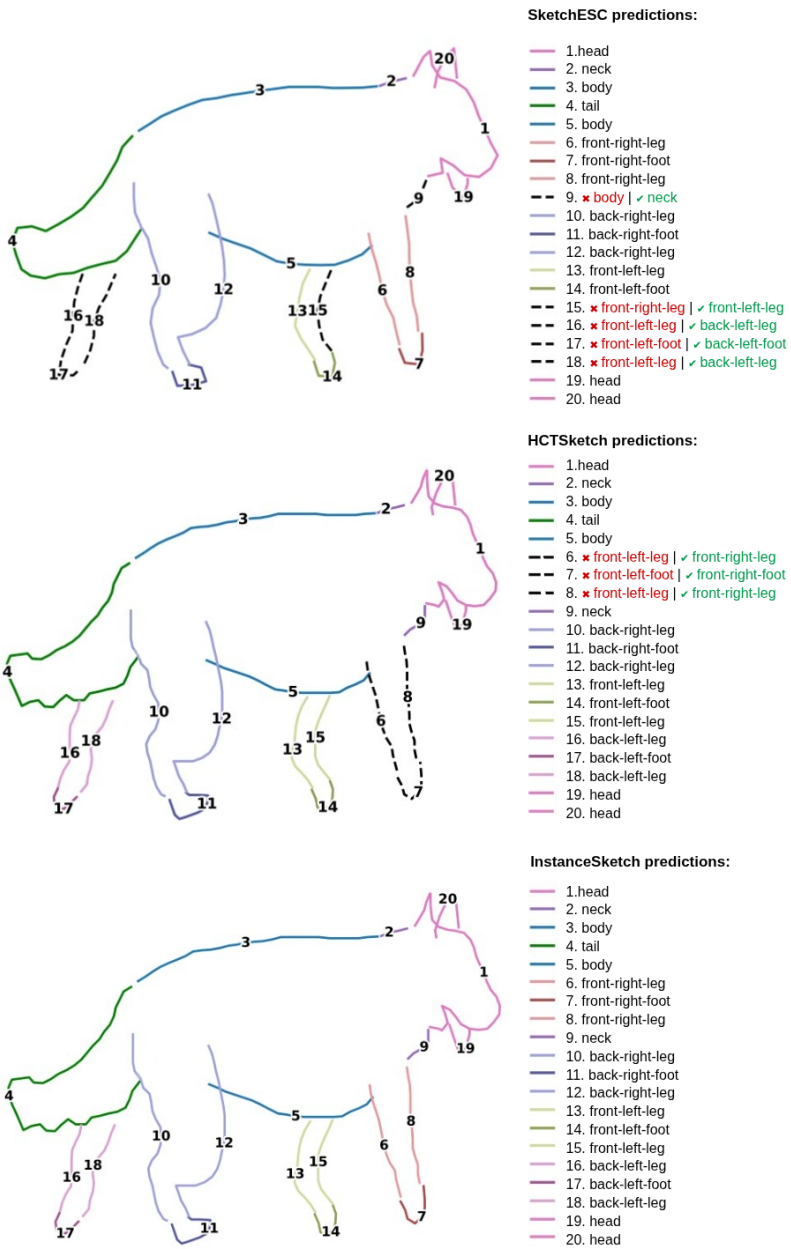

Fig 5. An example of a common error in other stroke segmentation models, where elements of different legs are often confused due to their visual similarity.

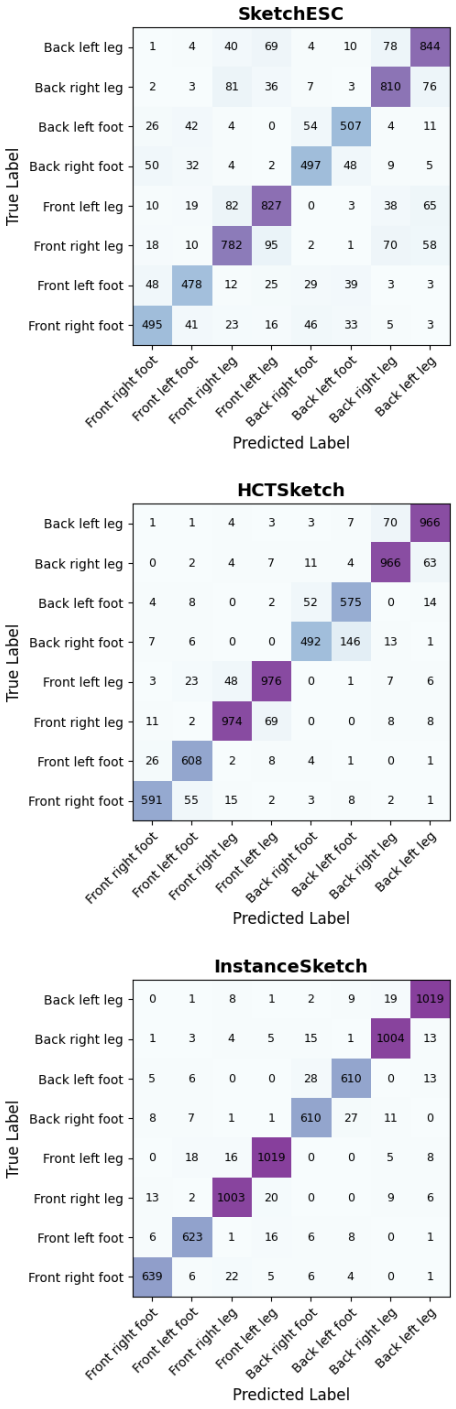

Fig 6. Sections of the confusion matrices showing leg-part segmentation results for different models.

Finally, while the Refinement approach showed strong performance specifically on strokes forming legs, a more comprehensive evaluation would benefit from additional datasets with fine-grained annotations. Such annotations would allow us to better separate and quantitatively assess performance on distinct stroke components, leading to a more detailed understanding of model strengths and limitations.

5 Code Availability

The implementation of InstanceSketch is publicly available at `https://github.com/y-svirhunenko/instance-sketch`

6 Conclusion

In this work, we presented InstanceSketch, a two-step pipeline for segmenting complex sketches, and InstanceSketch-Scene, an extension designed to decompose entire scenes into semantically meaningful components. By integrating convolutional neural networks with Transformers, our approach captures both local stroke-level details and global contextual information, enabling accurate and interpretable segmentation.

A key contribution of our method is the introduction of a flexible label refinement technique, which utilizes clustering techniques to effectively distinguish between strokes belonging to similar components, resulting in improved segmentation of challenging elements.

Experimental results demonstrate that InstanceSketch outperforms state-of-the-art methods in both segmentation accuracy and F1-score, while maintaining a lightweight architecture suitable for on-device deployment. Specifically, InstanceSketch achieves 93.41% segmentation accuracy and 90.70% F1-score, representing gains of 2.24 and 3.66 percentage points over HCTSketch, and 7.98 and 10.63 percentage points over SketchESC, respectively.

For InstanceSketch-Scene, our experiments demonstrate that incorporating sketch class information as contextual input improves stroke segmentation performance. In particular, we observe a +0.5% increase in accuracy and a +0.43% improvement in F1-score (see Appendix, Section 7.1 for detailed results).

Despite these promising results, challenges remain in segmenting rarely occurring decorative details and in handling dense or short strokes. Moreover, additional data is needed to fully evaluate the benefits of the refinement step.

Overall, our framework provides a robust and efficient solution for sketch segmentation, supporting downstream tasks such as recognition, generative modeling, and semantic analysis. Future work will focus on extending the approach to more diverse sketch styles and integrating it into interactive sketch-based applications.

List of Abbreviations

CNN	Convolutional Neural Network
CRF	Conditional Random Field
GNN	Graph Neural Network
GPU	Graphics Processing Unit
LSTM	Long Short-Term Memory
MLP	Multilayer Perceptron
RNN	Recurrent Neural Network
ViT	Vision Transformer
YOLO	You Only Look Once

Acknowledgements

This research received no external funding. All authors contributed to the conception, design, implementation, and writing of this work. All materials and resources used in this study were prepared by the authors.

References

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale, *International Conference on Learning Representations (ICLR)*. arXiv:2010.11929.

Ha, D., Eck, D. (2018). A neural representation of sketch drawings, *International Conference on Learning Representations (ICLR)*. arXiv:1704.03477.

Herold, J., Stahovich, T. F. (2011). Speedseg: A technique for segmenting pen strokes using pen speed, *Computers & Graphics* **35**(2), 250–264. DOI: 10.1016/j.cag.2010.12.003.

Schroff, F., Kalenichenko, D., Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering, *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823. DOI: 10.1109/CVPR.2015.7298682.

Stahovich, T. (2004). Segmentation of pen strokes using pen speed, *AAAI Fall Symposium - Technical Report* .

Sun, Z., Wang, C., Zhang, L., Zhang, L. (2012). Free hand-drawn sketch segmentation, *European Conference on Computer Vision (ECCV)*.

Svirhunenko, Y., Tytarchuk, P., Holovko, Y., Tereshchenko, Y. (2025). Hctsketch: Hybrid cnn-transformer approach for stroke segmentation in sketches, *Computer Science Research Notes* **3501**(1), 165–172. DOI: 10.24132/CSRN.2025-18.

Wang, J., Li, C. (2024). Contextseg: Sketch semantic segmentation by querying the context with attention, *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3679–3688. arXiv:2311.16682.

Wu, X., Qi, Y., Liu, J., Yang, J. (2018). Sketchsegnet: A rnn model for labeling sketch strokes, *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*. DOI: 10.1109/MLSP.2018.8516988.

Yang, L., Zhuang, J., Fu, H., Wei, X., Zhou, K., Zheng, Y. (2021). Sketchgnn: Semantic sketch segmentation with graph neural networks, *ACM Transactions on Graphics* **40**(3), 1–13. DOI: 10.1145/3450284.

Zhang, Z., Deng, X., Li, J., Lai, Y., Ma, C., Liu, Y., Wang, H. (2022). Stroke-based semantic segmentation for scene-level free-hand sketches, *The Visual Computer* **39**, 6309–6321. DOI: 10.1007/s00371-022-02731-8.

Zhu, G., Wang, S., Wu, T., Zhang, L. (2024). Enhance sketch recognition's explainability via semantic component-level parsing, *Proceedings of the AAAI Conference on Artificial Intelligence* **38**(7), 7731–7738. DOI: 10.1609/aaai.v38i7.28607.

Zhu, X., Xiao, Y., Zheng, Y. (2018). Part-level sketch segmentation and labeling using dual-cnn, *Proceedings of ICONIP*. DOI: 10.1007/978-3-030-04167-0_34.

Zhu, X., Xiao, Y., Zheng, Y. (2020). 2d freehand sketch labeling using cnn and crf, *Multimedia Tools and Applications* **79**, 1585–1602. DOI: 10.1007/s11042-019-08158-z.

7 Appendix

7.1 Scene Segmentation

While our pipeline performs well on single-object sketches, people often draw complex multi-object scenes, where many strokes belong to unrelated objects, providing no useful context and often confusing the model.

To address this challenge, we propose InstanceSketch-Scene, a three-step pipeline that first applies a detection model to identify individual objects, and then leverages the InstanceSketch backbone to perform detailed segmentation.

As the detection component, we utilize YOLO 8n; its object classification output is incorporated as contextual information for stroke segmentation. A scene of size 512×512 is fed into the detector, and strokes are assigned to objects based on the bounding box that contains the majority of their length.

After we get grouped strokes and an object label, we process it using the previously described Segmentation model. However, in this implementation, the model also generates embeddings of size 128 for each sketch class, which are then trained to better facilitate stroke segmentation. An appropriate class representation vector is selected and appended to the sequence of stroke embeddings before being fed into a Transformer encoder model. The Transformer block employs multi-head self-attention to model dependencies between strokes while considering the sketch's overall class. This approach yields better results than simply adding the sketch class to individual stroke representations or fusing them using MLPs. After processing the stroke embeddings with the Transformer encoder, further steps are identical to the InstanceSketch pipeline.

If needed, certain strokes can be further processed by the Refinement model, together with their primary labels.

Since the original dataset contains only isolated objects, we manually constructed a scene dataset to evaluate the full pipeline. Using an adaptive grid approach, individual objects are placed into non-overlapping grid cells with slight random shifts and per-object augmentations. Additional realism rules (e.g., a single sun, floating objects above grounded ones) yielded a training set of 18,000 samples and a test set of 2,000 samples.

To assess classification quality, each stroke is assigned the class label of the object it belongs to; strokes outside any detected bounding box are marked as noise.

To evaluate the impact of incorporating sketch class information on stroke segmentation, we also tested a variant of InstanceSketch-Scene (NoCls), which uses the Segmentation model from the standard InstanceSketch pipeline without class conditioning.

Although incorporating sketch class information as context led to slight improvements in segmentation performance (Segmentation accuracy improved by +0.5% and Segmentation F1-Score by +0.43%), these gains are relatively minor. Moreover, achieving such improvements requires a highly accurate classifier with a considerable number of parameters. Therefore, for most practical cases of single-object sketch segmentation, pre-classification may not be worthwhile. In contrast, for scene segmentation, where class prediction is essential, leveraging the classifier output is a meaningful and effective choice.

Examples of scenes segmented by InstanceSketch-Scene are shown in Figure 7. The obtained metrics are summarized in Table 4.

Fig 7. Examples of images processed by InstanceSketch-Scene.

Table 4: Validation performance comparison between InstanceSketch-Scene variants.

Metric	InstanceSketch-Scene (NoCls)	InstanceSketch-Scene
Class. Accuracy (%)	99.07	**99.07**
Class. F1-Score (%)	98.31	**98.31**
Segm. Accuracy (%)	92.98	**93.48**
Segm. F1-Score (%)	90.60	**91.03**

7.2 Additional Segmentation Examples

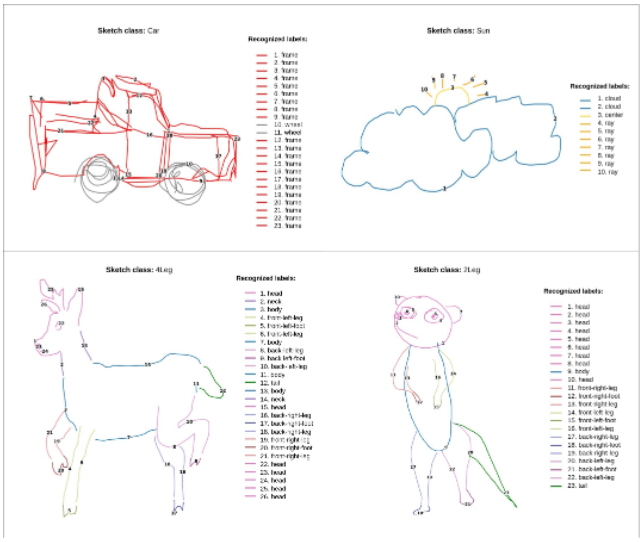

Fig 8. Examples of sketches correctly segmented by InstanceSketch.

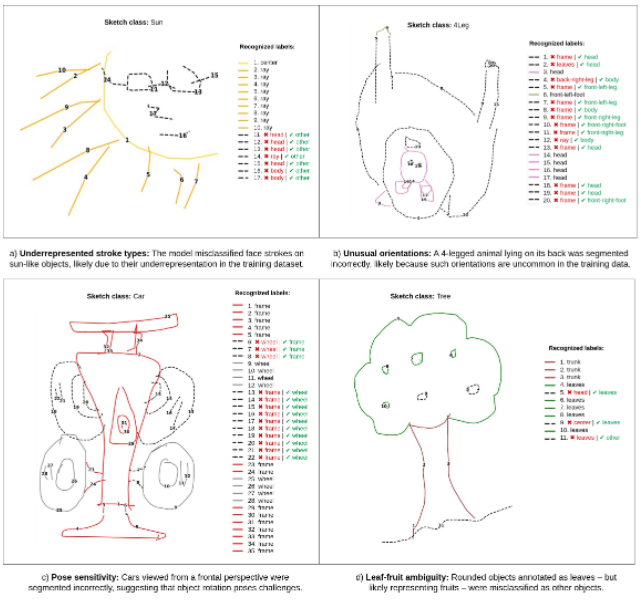

Fig 9. Examples of sketches incorrectly segmented by InstanceSketch.

Received December 28, 2025 , revised April 10, 2026, accepted April 13, 2026