Comput. Aided Geom. Des. 128 (2026) 102568

Contents lists available at ScienceDirect

Computer Aided Geometric Design

journal homepage: www.elsevier.com/locate/cagd

SmartTracer: Interactive tracing-based stroke extraction for complex line art

Zhengyu Huang [a], Zhongyue Guan [b], Zeyu Wang [b,*]

[a] Guangzhou Civil Aviation College, Guangzhou, 510403, China
[b] The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, 511453, China

HIGHLIGHTS

- An AI-assisted framework converts inaccurate inputs into high-quality strokes, addressing the ill-posed nature via prompts.
- We develop SmartTracer, an interactive system to easily extract user-specific stroke patterns in complex line drawings.
- Evaluations demonstrate that SmartTracer delivers more customizable, efficient stroke extraction than existing methods.

ARTICLE INFO

Keywords:
Interactive modeling
Human-in-loop system
Stroke extraction

ABSTRACT

Strokes are fundamental for understanding and editing drawings, as they constitute the basic units artists use to create line art. However, because different users may interpret stroke segmentation patterns differently within the same high-quality line drawing with complex topology, existing methods cannot efficiently support interactive editing that aligns with user intent. In this paper, we address the task of stroke segmentation, aiming to generate masks for natural and high-quality strokes from raster line drawings with complex topologies. Given a stroke prompt, such as a point or a short mark, our method employs convolutional neural networks (CNNs) to predict the corresponding stroke with reasonable connectivity. Building on this single-stroke extraction model, we further propose a graph-based algorithm to segment all strokes in a drawing while preserving user control in topologically complex regions. We also develop an interactive system, SmartTracer, to demonstrate the effectiveness of both our single-stroke and multi-stroke segmentation methods. Quantitative comparisons and user studies show that SmartTracer achieves superior stroke segmentation quality compared with existing methods, while also providing more convenient and efficient stroke extraction operations than commercial software.

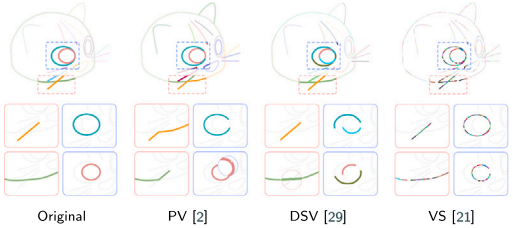
1. Introduction

Line drawing is an expressive and flexible visual medium widely used in illustration, animation, architecture, and product design. Although strokes are the fundamental units artists use to create line drawings, most drawings are still represented in raster format and therefore lack explicit vector-stroke information. Extracting natural and high-quality strokes from raster line drawings is highly valuable, as such strokes are essential for understanding and editing the drawing. However, traditional stroke extraction methods (e.g., curve parameterization), which typically require manual tracing, are labor-intensive and cannot guarantee consistent segmentation

* Corresponding author.
Email addresses: huangzhengyu@gcac.edu.cn (Z. Huang), zeyuwang@ust.hk (Z. Wang).
URL: https://cislab.hkust-gz.edu.cn/members/zeyu-wang/ (Z. Wang).

https://doi.org/10.1016/j.cagd.2026.102568
Received 9 March 2026; Received in revised form 16 April 2026; Accepted 26 April 2026
Available online 5 May 2026

(a) User-specific extraction patterns. Given the same line drawing, different users may intend different stroke segmentations (green) from their rough inputs (red). Note that the two petals overlap partially; therefore, directly deleting one petal in pixel-level editing would compromise the completeness of the other.



(b) Topology complexity. The overlapping strokes in the original line drawing lead to a complex topology after binarization, causing existing vectorization methods to fail in recovering the correct strokes and topology.

Fig. 1. Two major challenges in stroke extraction: pattern ambiguity (a) and topological complexity (b).

quality. Although these methods provide fine-grained control over details, the required time and effort make them impractical for large-scale or topologically complex drawings, highlighting the need for more effective solutions.

Despite its importance, stroke extraction remains an ill-posed problem for graphics research with two major challenges. First, stroke extraction is inherently ambiguous and often depends on user intent: for the same line drawing, different users may prefer different ways of grouping lines into strokes, leading to multiple plausible extraction results. Second, the topological complexity of line drawings makes stroke reconstruction from binary images particularly difficult. This difficulty mainly stems from the unpredictable intersections, overlaps, and connectivity patterns that frequently occur in complex drawings. As illustrated in Fig. 1(a), an effective stroke extraction method should therefore not only recover natural stroke structures, but also support multiple user-specified outcomes.

The most closely related lines of research are stroke segmentation and line vectorization. Several automatic stroke segmentation methods have been developed primarily for Chinese characters (Kim et al., 2018), but they often suffer from long processing times due to the intricate structure of character strokes. In addition, these methods are typically sensitive to image resolution, which limits their applicability to more detailed and topologically complex line drawings (Ito et al., 2022). Moreover, the segmented strokes they produce are often difficult to edit, thereby restricting their usefulness in downstream stroke-based applications.

Line vectorization methods face a different but closely related challenge. Because predicting correct stroke connectivity from raster input is inherently ill-posed, these methods often prioritize overall reconstruction quality over the segmentation quality of individual strokes. For example, Egiazarian et al. (2020) propose a deep learning-based vectorization method for technical line drawings, while Mo et al. (2021) use a neural network to fit a raster image within a moving window to segments of continuous parametric curves until no pixels remain. Despite these efforts, direct vectorization often results in missing details and unreasonable stroke connectivity due to the ambiguity of stroke structure and the complex relationships among strokes. Fig. 1(b) also illustrates several typical vectorization errors.

For interactive extraction, vector graphics software such as Adobe Illustrator allows users to manually trace strokes using parametric curves. Although this approach can bypass the ambiguity of stroke segmentation patterns and the challenges posed by topological complexity, it depends heavily on precise user input. As a result, even slight hand shakiness (or any other factors) during tracing can introduce inaccuracies and errors, leading to what we refer to as the scribble tracing problem (Fig. 2a).

Motivated by these limitations, we propose an interactive stroke extraction method that focuses on segmenting complete individual strokes and recovering the correct topological structure which allows user adjustment, thereby serving as a preprocessing step to improve vectorization quality.

This paper presents a bottom-up solution that narrows the gap between a user's scribble input (or prompt) and the high-quality stroke the user intends to extract (Fig. 2b), by combining the advantages of manual interactivity and automatic segmentation. As illustrated in Fig. 3, we first train two neural networks, ClickNet and TraceNet, for single-stroke segmentation, which predict a stroke mask from either a point or a short-mark prompt. We then design a prompt simulation algorithm and iteratively apply these networks on a graph constructed from the input line drawing to achieve multi-stroke segmentation. Building on these stroke segmentation methods, we develop an interactive system called SmartTracer, which allows users to conveniently refine the automatic segmentation results and obtain masks for their preferred strokes. Together, our stroke segmentation framework and the SmartTracer system enable

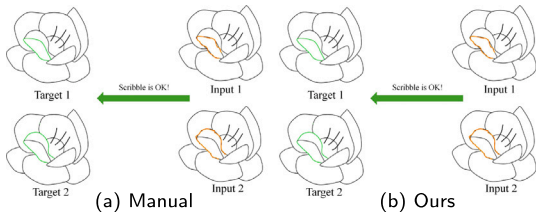


Fig. 2. Scribble tracing problem and our solution.Instead of directly treating the user scribble tracing as the extracted stroke (a), our method interprets the scribble input and maps it to the appropriate stroke mask in the line drawing (b).

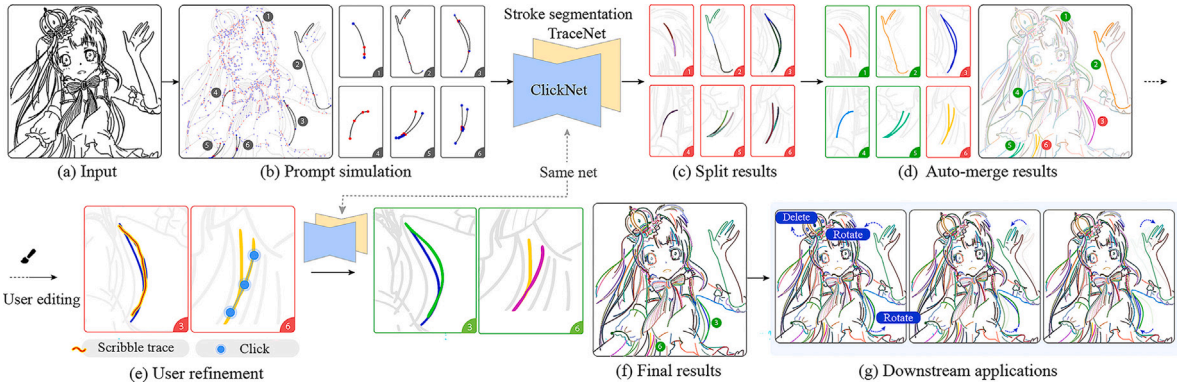


Fig. 3. Overview of SmartTracer. Given a raster line drawing (a), our method generates prompts (b) and automatically segments local strokes (c) with our neural networks. After our auto-merging (d), users can then interactively refine stroke segmentation via a click or scribble in real time (e) with the same networks as their expectation. SmartTracer enables high-quality extraction of overlapping strokes with user-preferred segmentation and topology, and our results (f) can be applied to downstream tasks such as stroke-level editing and animation (g).

the extraction of natural, high-quality strokes that better align with user intent, thereby facilitating downstream applications such as vectorization and stroke-level image editing.

In summary, we make the following contributions:

- We introduce an AI-assisted interactive framework that transforms imprecise, hand-shaken inputs into high-quality strokes, addressing the ill-posed nature of stroke extraction through a prompt-based formulation.
- We develop SmartTracer, an interactive system that enables users to conveniently extract strokes from prompts or refine automatic results, ultimately obtaining masks that match their preferred stroke patterns in complex line drawings.
- Quantitative and qualitative evaluations demonstrate that SmartTracer provides more customizable and efficient stroke extraction, while achieving superior stroke recovery compared to existing methods and commercial software.

2. Related work

This section reviews prior research related to stroke-level line drawing analysis, stroke segmentation, and vectorization.

2.1. Line drawing analysis

Understanding line drawings is a fundamental research problem that has long attracted the attention of the computer graphics and perception communities (Cole et al., 2009; Eitz et al., 2012). Early research concentrated on understanding line drawings at the pixel level (Cole et al., 2008), using the pixel distance between drawings to measure similarity. Since strokes are the fundamental unit of a drawing rather than pixels, subsequent studies leverage richer spatiotemporal information in the vector format (Gryaditskaya et al., 2019; Wang et al., 2021; Xiao et al., 2022), enabling a deeper understanding at the stroke level. The disparity between raster and vector formats highlights the importance of stroke segmentation, which aims to extract the masks of individual strokes from a raster drawing. Stroke segmentation methods are essential for supporting further stroke-level analysis and applications.

2.2. Stroke extraction

Stroke extraction aims to extract masks of individual strokes from a raster drawing, which differs significantly from general image segmentation. The input line drawing is typically a bitmap image that lacks color and texture information, making traditional segmentation methods ineffective. Moreover, the individual strokes have no clear semantics compared to natural images, since they do not correspond to distinct objects with recognizable features. Overlapping strokes also make it challenging to obtain natural

strokes with reasonable connectivity. Although there has been extensive research on image segmentation, relatively few studies have addressed stroke segmentation. Previous works have explored segmentation in Chinese characters (Liu et al., 2022; Zhang et al., 2022) and Latin characters (Berio et al., 2022), but they do not easily generalize to our area of concern.

Kim et al. (2018) employed a deep neural network to detect pixel-pair similarity, which estimates the probability that two pixels belong to the same stroke. It then segments strokes from the original raster line drawing by labeling all foreground pixels based on this similarity. A heuristic algorithm finds the minimum cut of the stream network to assign the labels. However, the processing time of this method increases exponentially with the number of foreground pixels, making it impractical to scale up to high-resolution line drawings. Ito et al. (2022) extended this method by calculating similarity and performing semantic segmentation on stroke segments separately after removing intersections. Their processing time depends on the number of detected intersections rather than the number of pixels, which is around 2 s on average for a 400×400 image, while the method by Kim et al. takes 18 minutes on average. However, the lack of precision in intersection detection leads to a decrease in the segmentation accuracy of the method by Ito et al. Despite these efforts, handling more complex topologies remains challenging for these methods when applied to high-quality drawings. Their networks with small kernel sizes can effectively segment symbolic drawings with few strokes, as seen in the QuickDraw Ha and Eck (2017) dataset, but they struggle with images containing significantly more strokes. Therefore, efficient and accurate stroke segmentation on complex line drawings remains an open problem.

2.3. Line drawing vectorization

Vectorization refers to the process of converting raster graphics into vector graphics. This task is related to stroke segmentation, but does not always prioritize the extraction of natural strokes. Line drawing vectorization is mainly categorized into two types: optimization-based methods (Noris et al., 2013; Favreau et al., 2016; Bessmeltsev and Solomon, 2019; Gutan et al., 2023) and learning-based methods (Guo et al., 2019; Mo et al., 2021; Puhachov et al., 2021; Yan et al., 2024). Below, we discuss this line of work, focusing on their segmentation and generalization performance on complex line drawings.

Optimization-based methods. Noris et al. (2013) proposed a method to extract stroke topology and centerlines using image gradients. However, the gradient field alone provides insufficient local information, resulting in unreliable vector strokes. Bessmeltsev and Solomon (2019) effectively vectorized simple line drawings using a graph-based approach with PolyVector fields, but this method can only support X- and T-junctions, limiting its capability to handle more complex drawings.

Gutan et al. (2023) introduced a frame field optimization framework that aligns singularity-free fields to line drawings to enhance vectorization quality. While previous approaches focused on clean line drawings, Favreau et al. (2016) developed a method capable of handling thick or sketchy lines. Their technique uses a minimal number of curves with few control points to reconstruct the drawing, in contrast to methods that generate excessively short curves and control points. However, this simplification may omit important details. The long runtime required by optimization-based methods is also a major concern.

Deep learning-based methods. Guo et al. (2019) employed neural networks to automatically infer the most likely node connectivity, focusing on the topology at junctions. However, their training dataset is limited and can only support vectorizing simple drawings. Mo et al. (2021) introduced a dynamic window to iteratively search and vectorize undrawn areas. Due to its random initialization, the method produces different outputs with each run, often requiring multiple attempts to achieve optimal results. It also tends to generate short, consecutive strokes and may miss details in highly complex drawings. Puhachov et al. (2021) argued that previous methods neglected correct drawing topology reconstruction. Their algorithm, PolyVector Field (PVF), designed to vectorize complex line drawings with correct topology, uses keypoint detection to guide graph construction and then optimizes it with PolyVector fields. However, its success depends heavily on keypoint detection accuracy. Yan et al. (2024) adopts a pairwise training strategy, mapping a line drawing to only one set of correct vector lines. This approach struggles to determine the “correct” topology in ambiguous cases. We believe that finding the “correct” topology in complex line drawings is an ill-posed problem, as even human tracings of the same drawing vary in stroke sets, and each could be considered a valid solution. In contrast to generating only one ground truth, we approach the problem by finding the most likely stroke segment given a rough stroke query, such as a point or short trace. Single-line drawing (Magne and Sorkine-Hornung, 2025), VecFusion (Xiao et al., 2018), and DeepSVG (Carlier et al., 2020) are also attempting to address the issue of ambiguity, such as stroke intersections and overlaps, but they belong to different scenarios.

In summary, our work is the first to tackle the segmentation problem in complex line drawings. Unlike previous approaches, our method can efficiently segment line drawings at a resolution of up to 1024×1024 with a relatively short runtime. The training data, which includes complex line drawings we collected from the artists, allows our algorithm to adapt to intricate topologies. Additionally, our method supports real-time extraction of individual strokes. The interactive tools we developed in our segmentation system enable convenient and customized user editing.

3. Single-stroke segmentation

3.1. Problem definition

Mathematically, we treat the stroke segmentation problem on a raster drawing as finding a parametric solution to a function. Consider a clean line drawing $I = \cup_{i=1}^{m} s_i$, where s_i represents the mask of each stroke and m is the total number of strokes. The goal of stroke segmentation is to obtain a stroke sequence using a function $\mathcal{F}_\theta$, which is defined as:

$$\mathcal{F}_\theta(I) = \{\hat{s}_1, \hat{s}_2, \dots, \hat{s}_n\}, \tag{1}$$

where $\hat{s}_i$ represents the mask of each segmented stroke and n is the total number of segmented strokes. The optimization objective is:

$$\theta = arg\ \min \mathcal{L}(I, \cup_{i=1}^{n} \hat{s}_i), \tag{2}$$

where θ represents the learnable parameters of the segmentation function $\mathcal{F}$ and $\mathcal{L}$ is a loss function measuring the pixel similarity between the extracted strokes and the ground truth. Since the function $\mathcal{F}$ outputs a binary image (a two-dimensional matrix), $\mathcal{F}$ in Eq. (1) can be expressed as the union of strokes in Eq. (2), i.e., $\mathcal{F}_\theta = \cup_{i=1}^{n} \hat{s}_i$. The solution space is vast due to ambiguities in stroke extraction and stroke order. To address this ill-posed problem, we introduce user-interactivity with the stroke prompt, i.e., a point or a rough partial trace on a stroke, narrowing the task to single-stroke segmentation. This approach is inspired by the fact that humans can perceive and extract corresponding strokes from arbitrary prompts. Given a stroke prompt x_i for the i-th stroke s_i, the extracted stroke is defined as $\hat{s}_i = \mathcal{F}_\theta(I, x_i)$, The optimization problem is thus reduced to:

$$\theta = \arg\min \mathcal{L}(\mathcal{F}_\theta(I, x_i), s_i) \tag{3}$$

This reduces the solution space from extracting all ordered strokes to a single stroke. And once the θ is learned by our model, the user can easily access the intended strokes by inputting the stroke prompts. Our approach disregards the stroke order but focuses on extracting natural single strokes. Once a single stroke is successfully extracted, user-specific stroke segmentation can be achieved with the following objective:

$$\gamma = \arg\min \Sigma_{i \in m} \mathcal{L}(s_i, \mathcal{F}_\theta(I, U_\gamma(\hat{x}_i))) \tag{4}$$

Therefore, we designed a human-in-loop interface U_γ, allowing users to achieve personalized stroke extraction that matches their expectation (i.e., pattern γ). Note that γ is the user-specified segmentation pattern (Fig. 1a). As we introduce "prompts," different segmentation patterns correspond to different prompt patterns. When a specific stroke s_i in Eq. (3) is identified, users can extract it with various different patterns of prompts $\hat{x}_i$ and the interface function $U_\gamma(\hat{x}_i)$ make the results obtained by $\hat{x}_i \in s_i$ approximate the output of the original prompt x_i. This makes users efficiently achieve the complete process of user-specific stroke extraction from a complex line art. Furthermore, we selected a human-like pattern $\hat{\gamma}$ to achieve automatic multi-stroke extraction. This problem can be reformulated as a prompt simulation task, which will be explained further in Section 5.

$$\hat{\gamma} = \arg\min \Sigma_{i \in m} \mathcal{L}(s_i, \mathcal{F}_\theta(I, U_{\hat{\gamma}}(\hat{x}_i))) \tag{5}$$

For users, compared to manually inputting prompts for all strokes, it is far more efficient to make minor editing operations to obtain their expected pattern γ output from initial pattern $\hat{\gamma}$ results.

3.2. Model architecture

Inspired by the clicking and tracing operation in the manual stroke extraction from commercial software (such as Illustrator), we adopt ClickNet and TraceNet to ensure that user input is more flexible and aligns with the habits of professional users. Our model comprises two identical fully convolutional encoder-decoder CNN cascades, as shown in Fig. 4. The first CNN, called ClickNet, takes as input a point on a stroke as the stroke prompt and outputs the corresponding segmented stroke. The primary function of ClickNet is to accurately extract initial local strokes with clear directionality from point prompts generated by clicks (as shown in Stage I in Fig. 4), even when the prompt is near a junction (as in Fig. 6a). The second CNN, called TraceNet, is designed to refine the segmentation of full strokes. Using the guide strokes generated by ClickNet as input, TraceNet enhances segmentation by accurately identifying and segmenting longer strokes in high-resolution line drawings, shown as Stage II in Fig. 4.

Each CNN in our framework comprises 24 layers, primarily based on Conv-BatchNorm-ReLU blocks (He et al., 2016)). Note that He et al. (2016) introduced ResNet-34 for feature extraction, which means in this architecture, its output dimension is lower than its input dimension, whereas our network is designed to generate stroke masks (the shapes of the input and output are the same). Our network downsamples the input three times using convolutions and restores it to its original size through sub-pixel convolutions, allowing the input image size to be any integer multiple of 8. To avoid the error-prone nature of zero-padding, the first layer employs a 9×9 pixel kernel with 4×4 reflective padding.

3.3. Loss function

To specify the loss function $\mathcal{L}$ in Eq. (5), we must account for the imbalance between the segmented strokes and the background, where negative samples in the background typically dominate. To address this, we adopt a loss function that combines the Dice loss Milletari et al. (2016) with the mean squared error (MSE) loss:

$$\mathcal{L} = \alpha \mathcal{L}_{\text{MSE}}(s_i, \mathcal{F}_\theta(I, x_i))) + \beta \mathcal{L}_{\text{Dice}}(s_i, \mathcal{F}_\theta(I, x_i))), \tag{6}$$

where $\alpha = 0.1$ and $\beta = 1$ in our experiments, and the notations are consistent with those in Eq. (5). The MSE loss is a natural choice as it encourages the strokes to maintain their original shapes, while the Dice loss is advantageous due to its robustness against class imbalance. It can effectively handle the disparity in the number of positive and negative samples, allowing the model to focus more on accurately segmenting the smaller strokes. Otherwise, due to the imbalance of positive and negative samples, training with the MSE or Dice loss alone tends to be unstable, ultimately resulting in pure white images regardless of the input prompt.

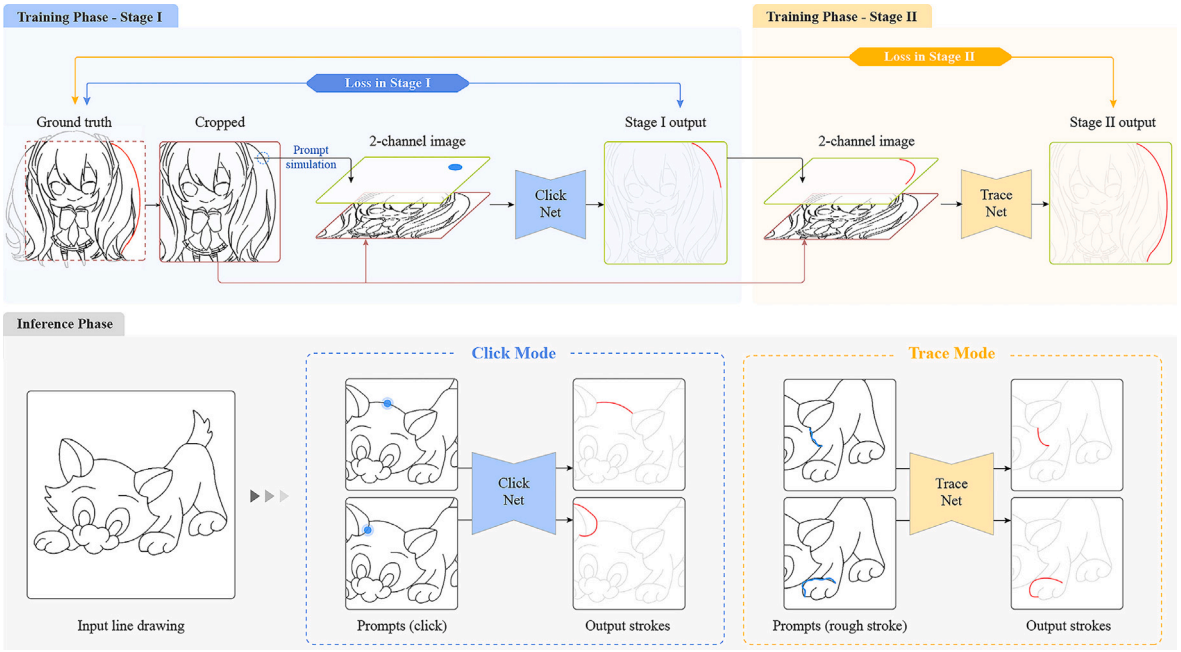


Fig. 4. Our training process. In the training phase, the randomly generated prompts and the original image are merged into a 2-channel image as input, and the output of ClickNet's stroke segmentation results in 1-channel data, which must be merged with the original image again and fed to TraceNet for training. In the test phase, our ClickNet and TraceNet correspond to the click mode and trace mode in our UI, respectively, and users can achieve satisfactory results by performing manual segmentation of strokes or fine-tuning the results based on our automatic segmentation results.

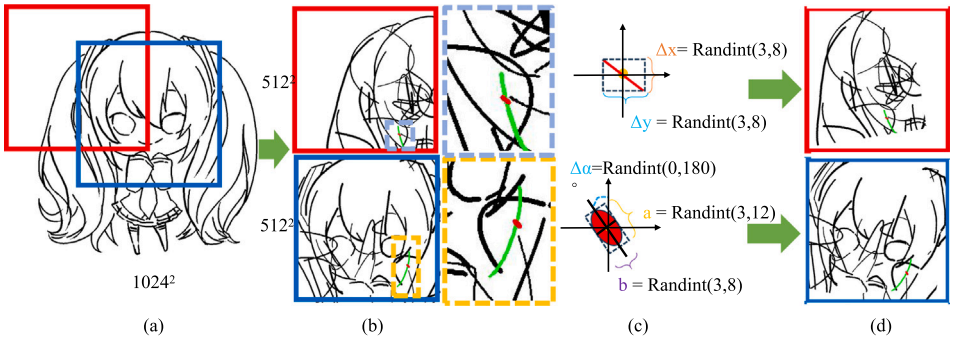


Fig. 5. Details for prompt-stroke pair generation. The resolution of the original image (a) in the training set is 1024^2, which is processed through random cropping and random Bézier curve replacement of strokes to obtain 512^2 input patches (b). Based on (b), random strokes (green) are selected as the output, while the input prompts (red) are generated by randomly selecting a center point near these strokes (dashed box), drawing short line segments (top) or ellipses (bottom) according to the rules in (c). Finally, random translation and scaling operations are applied to obtain the prompt-stroke pairs (d). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

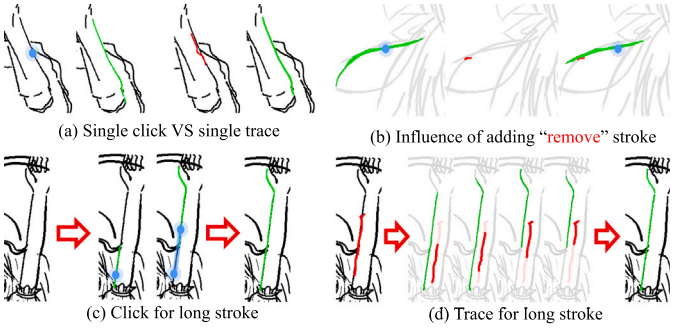


Fig. 6. Click mode and trace mode. Both modes can produce similar results with different inputs. The red "remove" stroke added in manual mode can change the path of the extracted stroke in (b), and click mode merges results from multiple user-specified points, whereas trace mode automatically extracts strokes and performs merging after segmenting the input curve into sub-line segments, as described in Section 5. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3.4. Implementation details

Dataset collection. In order to train our networks to learn the visual features of strokes drawn with a stylus, we collected 179 high-quality vector line drawings in an anime-style from eight artists by recreating the color illustrations in the Pixiv Pixiv (2025) dataset. To ensure consistency in network learning despite individual variations in brush strokes, we selected two artists with visually similar stroke styles. Fourteen of their drawings were used for training, while the remaining works constituted the test set (VW165–165 line drawings with variable width). The user interface for data collection closely resembles the manual mode in SmartTracer, as shown in Fig. 7, with the addition of a layer selection feature and an optional mode supporting pressure-sensitive stroke widths. All strokes, including vertices, stroke widths, and corresponding timestamps, are recorded as the artists draw.

Training details. We trained our networks using an NVIDIA RTX 4090 GPU. The size of the input patches and output images was 512×512. This patch size is used solely for extracting individual strokes; our method applies to inputs of arbitrary size. In Stage I, we trained the networks for over 15,100 iterations with a batch size of 16, followed by refinement in Stage II for 500 iterations, maintaining the same batch size. Before Stage II, we shared the parameters learned by ClickNet in Stage I with TraceNet for joint training. Since the training data were randomly cropped from the original line drawings, we used iterations rather than epochs to describe the training process. We used the Adam optimizer Kingma et al. (2015) with an initial learning rate of 0.001. A learning rate decay schedule was applied every 1000 iterations, with a decay factor of $\lambda = 0.5$.

Prompt-Stroke Pair Generation. There are only 14 images in the training set (averaging more than 200 raw strokes per image), which constitutes a few-shot learning problem. However, note that our training is conducted in terms of prompt-stroke data pairs. After our data augmentation operations, variations of effective prompt-stroke pairs increase dramatically, and few duplicate prompt-stroke data pairs appear in each iteration. After randomly selecting a stroke, as shown in Fig. 5(c), we select a random point on the stroke and draw a short line segment or a random ellipse as a prompt for this stroke. To prevent overfitting, in addition to adding random prompts and rotating and scaling (Fig. 5d), we include synthetic images in the training: all strokes in the crop box where the selected stroke is located randomly replace the original stroke with a 50% probability, making it a quadratic Bézier curve or replacing the stroke width in the range of 1px to 5px. At the same time, we also allow a certain probability (10%) that the random prompt point and the stroke do not intersect, and then the single-channel output is a pure white image in this case. In Stage II, the trace prompts are the output strokes from Stage I, which contain direction information. That's why our TraceNet is sensitive to direction, while there is no direction factor in Eq. (3).

4. Interactive system

Based on our single-stroke segmentation methods, we develop an interactive system called SmartTracer for user refinement using Python and Qt. To design the interface function $U_\gamma(\hat{x}_i)$ Eq. (3), our UI supports the following four modes for stroke segmentation, as Fig. 7 shows.

Manual mode (M). Our stroke extraction methods do not take effect. The freehand sketch drawn by the user will be converted into stroke masks faithfully.

Trace mode (T). Trace mode allows the user to extract the corresponding stroke using TraceNet by drawing a partial trace as the stroke prompt. This mode can also extract long strokes by merging results from multiple stroke prompts, as shown in Fig. 6. From the training results, it also validates our motivation for using two networks: TraceNet is more sensitive to the direction of line segment

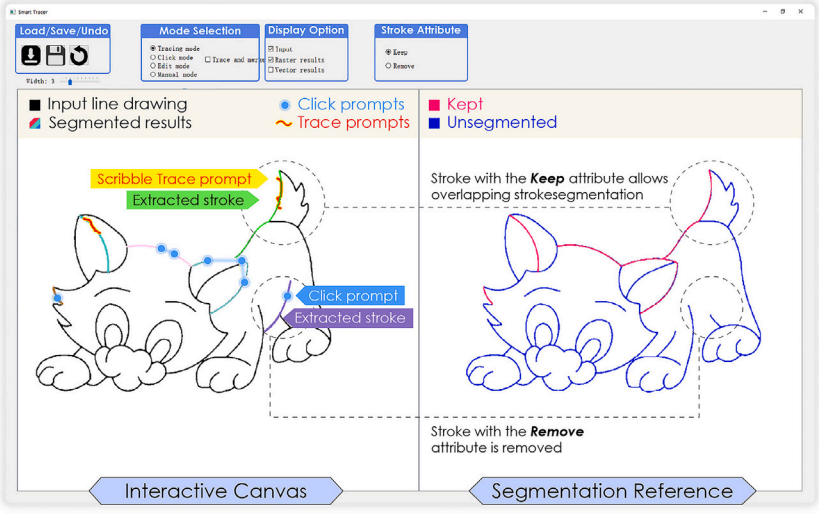


Fig. 7. Stroke segmentation interface. The left canvas enables users to generate new segmented strokes using trace and click modes or to modify existing strokes in edit mode. The right canvas dynamically reflects the results from the left canvas and uses three color indicators: blue for unsegmented regions, pink for segmented strokes marked with the "keep" attribute, and green for the currently selected stroke in edit mode. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

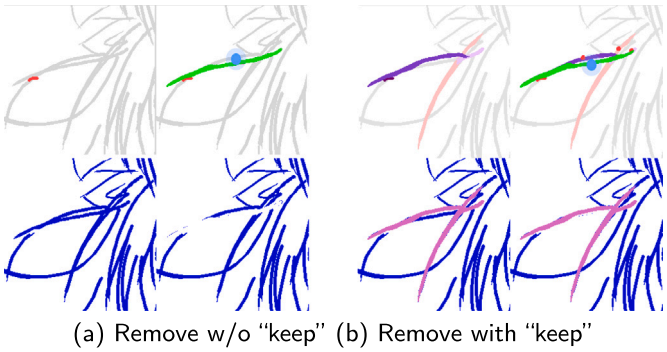


Fig. 8. An example of avoiding overlapping segmentation by switching the stroke attribute from "keep" to "remove". Given a line drawing and a stroke labeled as "remove" (in red), cases (a) and (b) attempt to extract the same target stroke using the "remove" attribute. The bottom row shows the corresponding segmentation results in the right canvas, as illustrated in Fig. 7. Since (b) has "kept" two strokes (from the red point prompts), it preserves the overlapping region that would otherwise be removed in case (a), thereby preventing undesired segmentation loss. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

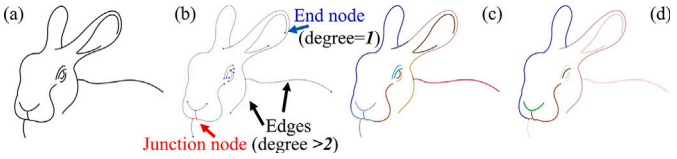


Fig. 9. Automatic stroke segmentation algorithm for the input line drawing (a). Using the edges in the simplified graph (b) composed by the skeleton algorithm as input, our CNNs can obtain the initial stroke segmentation (c). Through optimization, we merge the fragmented edges and generate the final segmentation (d).

prompts, but if given the point prompt on the left of Fig. 6(a), TraceNet cannot recognize it – even though its initial parameters come from the trained Stage I and were only fine-tuned for 500 iterations. On the other hand, due to the locality of CNN, ClickNet cannot capture long lines from point prompts (as shown in Stage I of Fig. 4).

Click mode (C). Click mode allows the user to extract the corresponding stroke using ClickNet by selecting a point as the stroke prompt. Similar to trace mode, click mode also supports extracting longer strokes by merging multiple local strokes.

Edit mode (E). Edit mode allows the user to delete, change vertex positions, and toggle attributes for extracted strokes.

The extracted strokes have two attributes, "keep" and "remove," meaning that the extracted stroke does not affect the original drawing or is removed from the original drawing, so it will not be extracted again. The stroke attributes are mainly used to avoid over-segmentation so the user can generate an intended result. Fig. 8 shows the effect of the attributes on stroke segmentation.

The purpose of the function U is to reduce the burden of accuracy requirements for user input prompts. The four modes correspond to four types of operations, so the function $U_\gamma(\hat{x}_i)$ is:

$$U_\gamma(\hat{x}_i) = u_0 \cdot M(\hat{x}_i) + u_1 \cdot T_{\text{TraceNet}}(\hat{x}_i) + u_2 \cdot C_{\text{ClickNet}}(\hat{x}_i) + u_3 \cdot E(\hat{x}_i) \tag{7}$$

where u_0 to u_3 are controlled by users, and their values are either 0 or 1. This gives users sufficient freedom to customize the strokes and achieve their desired pattern γ.

5. Multi-stroke segmentation

As requiring users to input prompts for every single stroke remains a time-consuming and laborious process, we design an algorithm for stroke prompt simulation mimicking humans' observation to extract all strokes from a raster drawing with a fixed Pattern $\hat{\gamma}$. As Fig. 9 shows, we first extract the skeleton from the input drawing as a graph using the method by SK (Zhang and Suen, 1984). The graph $G = (V, E)$ is built by traversing the eight-connected neighbors of pixels iteratively, starting from selected points in the thinned image, while merging vertices with a degree of two. Fig. 9(b) presents an example graph, where blue end nodes have a degree of one and red junction nodes have a degree greater than two.

Graph-based prompt simulation. The goal of prompt simulation is to obtain a set of points and partial traces as the stroke prompt, so we can iteratively use the single-stroke segmentation models to extract all strokes. Given an edge $e \in E$, on which the pixels are represented by $\{p_0, p_1, \dots, p_{N-1}\}$, our algorithm simulates the stroke prompt based on the number of pixels N as follows:

- **Denoising.** If $N \leq 3$, we discard the edge and do not generate any stroke prompt, as it is too short and can be noise caused by skeletonization.
- **Near-junction perception.** If $3 < N \leq 20$, we create a point prompt by drawing a circle with a radius of three pixels centered at the middle pixel $p_{\lfloor N/2}$ and use ClickNet for stroke extraction; If $20 < N \leq 80$, we create a trace prompt by drawing a three-pixel-width polyline using all pixels on the edge and use TraceNet for stroke extraction.
- **Long stroke tracing.** If $N > 80$, we create multiple trace prompts using an offset $k = 20$ and an interval of $3k$, i.e., $\{p_0, \dots, p_{3k}\}, \{p_k, \dots, p_{4k}\}, \dots$, and $\{p_{\lambda k}, \dots, p_N\}$ if $N > (\lambda+2)k$ in a similar manner and feed them to TraceNet, where $\lambda = \rfloor N/k - 2$.

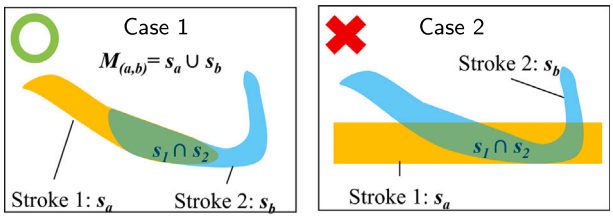


Fig. 10. Stroke merge condition. Strokes in Case 2 are not allowed to merge.

Automatic merging. Our single-stroke segmentation method generates a stroke mask, called a sub-stroke, every time we feed a stroke prompt simulated using the algorithm above. We aim to merge the sub-strokes to obtain more complete strokes. Since our prompt simulation algorithm allows overlapping strokes, we compute the Maximum Intersection Ratio (MIR) of the sub-strokes on adjacent edges and set a threshold $\tau = 20\%$, which serves as the criterion for merging the sub-strokes. Given two strokes s_a and s_b, the MIR is:

$$\text{MIR} = \max\left(\frac{s_a \cap s_b}{s_a}, \frac{s_a \cap s_b}{s_b}\right) \tag{8}$$

Moreover, our algorithm also ensures that merged strokes have a single-path topology with no more than two end nodes. For example, both cases in Fig. 10 satisfy the MIR condition, but only Case 1 allows merging, whereas Case 2 does not, as merging would increase stroke complexity. The highlighted part in Fig. 9(d) shows the merged strokes, and the overall segmentation results are more concise compared to Fig. 9(c) before merging.

6. Results and evaluation

In this section, we evaluate the performance of our automatic stroke segmentation methods and the user experience with our interactive system, SmartTracer. Our evaluation comprises the following components: Section 6.1 evaluates the quality of segmentation results, comparing the performance of the multi-stroke segmentation algorithm and the vectorization algorithm guided by automatically generated cues; Section 6.2 presents a subjective user evaluation of SmartTracer in scenarios involving user input.

6.1. Stroke segmentation algorithm

6.1.1. Dataset

We evaluate our automated approach on two datasets: the drawings we collected "VW165", as described in Section 3.4, and the dataset "UW369" used in Yan et al. (2024). In comparison, VW165 is more complex and includes variable-width strokes, which present greater challenges for extraction, while the strokes in UW369 have a uniform width. We rasterize the original vector graphics in SVG format into 1024×1024 images, which serve as the high-resolution ground-truth input. To assess the segmentation performance, we re-render the output vector graphics into raster images and compute several metrics, including Intersection over Union (IoU), Chamfer Distance (CD), Hausdorff Distance (HD), and Frechet Inception Distance (FID), comparing these results with the ground truth. Given that no existing stroke segmentation method specifically targets high-quality line drawings, we use PV (Bessmeltsev and Solomon, 2019), PVF (Puhachov et al., 2021), VS (Mo et al., 2021), and DSV (Yan et al., 2024), which can handle high-resolution line drawings, as baseline methods. For evaluating the vectorization capability enabled by our segmentation, strokes extracted by our automatic methods are vectorized in SVG by Potrace (Selinger, 2003), as it is well-suited for reconstructing variable-width strokes. Note that both Potrace and SK do not generate any strokes without stroke segmentation; instead, they outline the entire connected component (connected pixel region). All methods are run on an i5-13400 CPU with an NVIDIA GeForce RTX 4090 GPU. Since all methods, except VS, produced lines of fixed width, we selected the width that resulted in the maximum IoU for each drawing to ensure a fair comparison.

6.1.2. Results

Table 1 presents the quantitative results on the two datasets, respectively. In each sub-table, the upper section reports the comparison of all rasterized outputs, while the lower section focuses on the comparison of all vectorized outputs. We referred to the results before the automatic merging as "Ours-split" and to those after the merging as "Ours-merge." None of them contains user-refinement in this section. We analyze the results in detail below.

Pixel-level stroke extraction incurs less loss. All methods exhibit performance degradation when rendered with a fixed 2-pixel stroke width compared to optimal-IoU rendering (e.g., PVF-w2 vs. PVF in Table 1). In contrast, our automatic stroke extraction method (Our-raw) directly outputs raster strokes with varying widths and achieves the best performance across most evaluation metrics. This demonstrates its superior alignment with the input and underscores the advantage of low-loss, pixel-level stroke extraction for editing.

Stroke extraction improves vectorization. As illustrated in Fig. 11, the stroke reconstruction results of applying VS to our segmented results (Ours + VS) preserve finer details better than applying VS alone, consistent with our quantitative findings. In Table 1 for VW165, Ours + VS improves all metrics compared to VS alone and also outperforms other vectorization methods. Table 1 for UW369 shows that while CD and HD improve with Ours + VS, IoU and FID exhibit slight declines. This performance drop can be attributed to the characteristics of the UW369 dataset, which features thin, fixed-width line drawings where the additional vectorization loss introduced by consecutive vectorization steps outweighs the benefits of segmentation.

Table 1
Quantitative comparisons on the VW165 and UW369 dataset. Ours-raw: our method before vectorization, Potrace-only: vectorization with Potrace without our stroke extraction, SK: skeleton algorithm (Zhang and Suen, 1984), -w2: 2-pixel width, r512: of 512 × 512 resolution. Unit of metrics: CD (10^{-5}), IoU (10^{-2}), and HD (10^{-3}).

	VW165				UW369			
Method	IoU ↑	CD ↓	HD ↓	FID ↓	IoU ↑	CD ↓	HD ↓	FID ↓
Potrace-only	91.51	4.37	**3.39**	4.08	85.92	**7.46**	**2.58**	4.66
SK	23.89	58.56	9.40	69.15	49.37	25.55	5.65	21.70
Ours-raw	**99.02**	**2.00**	21.28	**2.12**	**98.97**	15.61	11.24	**1.48**
PV-w2	22.61	64.56	19.60	70.15	44.13	32.21	9.52	24.98
PV	56.46	33.61	18.77	30.67	63.55	24.36	9.17	11.08
PVF-w2	26.11	60.85	21.10	81.21	50.84	29.53	13.89	29.14
PVF	60.99	29.62	20.27	47.95	63.97	23.50	13.61	20.02
VS	75.30	20.39	30.03	15.96	74.71	24.00	14.79	11.44
DSV-w2	27.27	114.17	54.96	82.99	51.07	25.82	8.29	22.14
DSV	57.45	88.36	54.28	59.39	59.38	22.59	8.14	16.41
DSV-r512	56.30	44.11	36.31	35.85	57.63	28.03	7.00	11.52
Ours-split + Potrace	88.15	7.62	21.49	8.29	79.79	25.68	11.98	10.29
Ours-merge + Potrace	**89.56**	**6.96**	21.53	**7.29**	**82.67**	24.38	12.09	**8.76**
Ours + VS	77.24	13.38	**7.33**	13.05	71.46	**15.40**	**3.98**	12.86

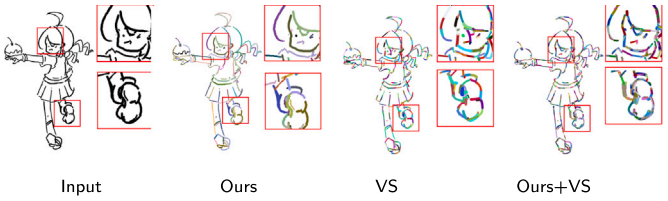


Fig. 11. An example of vectorization improvement. VS applied in our stroke extraction results (Ours + VS) can reconstruct the correct topologies near the eyes and shoes in the red boxes. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Correlation analysis. CC: Correlation Coefficient.

	VW165		UW369	
Metric Pair	CC	p-value	CC	p-value
IoU & CD	−0.83	0.0002	−0.71	0.0043
IoU & HD	−0.26	0.3646	0.06	0.8263
IoU & FID	−0.96	0.0000	−0.90	0.0000
CD & HD	0.70	0.0057	0.50	0.0656
CD & FID	0.91	0.0000	0.69	0.0067
HD & FID	0.42	0.1366	0.18	0.5387

Table 3
Computation time and success rate (SR).

	VW165		UW369	
Method	Time	SR	Time	SR
PV	430.53	100.00	89.53	100.00
PVF	541.28	59.39	188.97	81.84
VS	54.06	100.00	34.71	100.00
DSV	220.24	97.58	165.37	99.18
Ours-split	**35.55**	100.00	**29.69**	100.00
Ours-merge	38.46	100.00	30.88	100.00

Our results most closely resemble the input style. In terms of FID, which measures the distributional similarity between the input clean line drawings and the outputs, our segmentation achieves a higher score than other methods across both test sets. This indicates that our method more accurately captures the drawing style of the artist.

Our results demonstrate superior visual quality. The qualitative comparisons are shown in Fig. 22. In complex line drawings, our method yields more reasonable stroke segmentation in the zoomed-in regions compared to other approaches. Furthermore, the segmentation results can be interactively refined based on user input, offering additional flexibility. Correlation analysis in Table 2 reveals strong and statistically significant correlations ($p < 0.05$) among IoU and FID, IoU and CD, as well as CD and FID. Given that CD has been shown to better align with human perception (Yan et al., 2020), our results, characterized by higher IoU and lower FID and CD, further highlight the superior visual quality of our method.

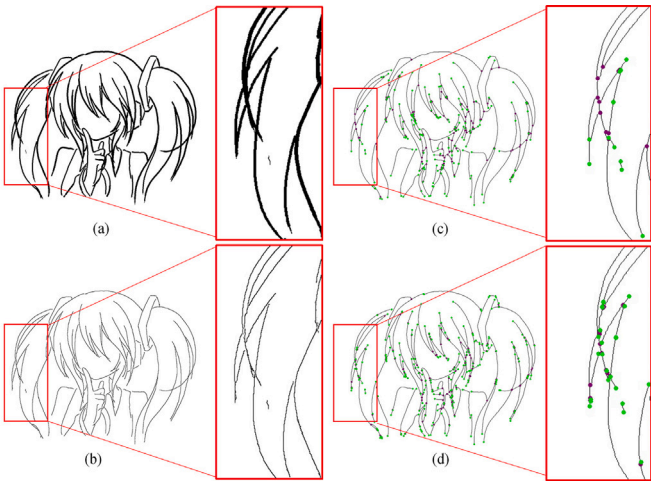


Fig. 12. Traditional skeletonization (SK) VS deep learning based method. For a given input (a), if we first use a deep learning method, LineNormalizer (Simo-Serra et al., 2018), to simplify the strokes and obtain (b), it will actually introduce additional errors (red box). This leads to more inaccurate prompts (d) compared to directly using the SK-generated image (c). Green points in (c) and (d) are end nodes, and purple points are junction nodes as defined in Fig. 9. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Our method requires less computation time. As shown in Table 3, the increased complexity of line drawings in the VW165 dataset, compared to UW369, leads to longer average computation times for other methods and a reduced vectorization success rate for PVF. In contrast, our method consistently remains the most efficient. For single-stroke extraction within our user interface, the process takes approximately 0.30 s, allowing users to obtain results in real time. For the automatic algorithm (Ours-merge), the average computation times are 38.46 s for VW165 and 30.88 s for UW369. The computation time of our system is proportional to the number of prompts extracted from the image. Since our algorithm is heuristic, it requires extracting strokes one by one during traversal. The more complex the topology (with more prompts), the longer it takes.

SK vs. LineNormalizer (Simo-Serra et al., 2018). As shown in Fig. 12, deep learning skeletonization may not achieve a better topological structure for prompt simulation. On the other hand, the low success rate in PVF is also mainly caused by the mismatch of junction point detection in their preprocessing with a deep learning-based method, indicating that deep learning-based methods are not stable for images with complex topologies. The main reason is that current deep learning methods still struggle to effectively resolve stroke ambiguity in complex topological structures. That's why we do not adopt such methods. Instead, our prompt simulation adopted SK Zhang and Suen (1984), which can achieve a complete topological structure for complete reconstruction in our automatic part to reduce the workload of user refinement.

Our method remains stable for varying-width strokes with noise. As noted in DSV (Yan et al., 2024), their method tends to fail on thick strokes, leading to improved performance at a lower input resolution of 512×512 (DSV-r512) compared to 1024×1024. In contrast, our interactive method exhibits greater stability in stroke extraction for noisy binary images, as demonstrated in Fig. 16. In such scenarios, the quality of the results depends heavily on user input, which represents a limitation of our approach in fully automatic settings.

Style diversity. Although both our training and test images are in anime style, our data augmentation method expands the diversity of styles. As shown in Fig. 13, the results on UW369 demonstrate that we can also achieve good reconstruction effects in complex scenarios such as car interiors and architectural styles.

Limitations. Our automatic method relies on the accuracy of the prompts. As Fig. 14 shows, when the input image is not a clean line drawing, unnatural strokes may be generated due to incorrect topological prompts from SK.

Ablation study. We report the performance of our method without the merging step (Ours-split + Potrace) in Table 1, which yields slightly lower metric scores compared to the full algorithm. The time difference between this variant and our complete method is shown in Table 3 reflects the runtime of the merging algorithm, which remains relatively efficient. Additionally, our approach achieves a substantial improvement over the simple skeletonization algorithm (SK), which is used for prompt generation in our system.

Discussion for MIR. As shown in Fig. 15, a lower MIR indicates more lenient merging conditions, resulting in a smaller total number of output strokes. Choosing 20% will allow users to ultimately process fewer strokes. However, note that this does not necessarily mean the automatic segmentation results align with the user's intent. Therefore, we need to evaluate interactive factors in advance based on user input.

6.2. Stroke segmentation interface

To evaluate the user experience with SmartTracer, we invited 18 participants to use our stroke segmentation system on a Wacom tablet together with a stylus and mouse. We let the participants manually segment two simple drawings and one more complex

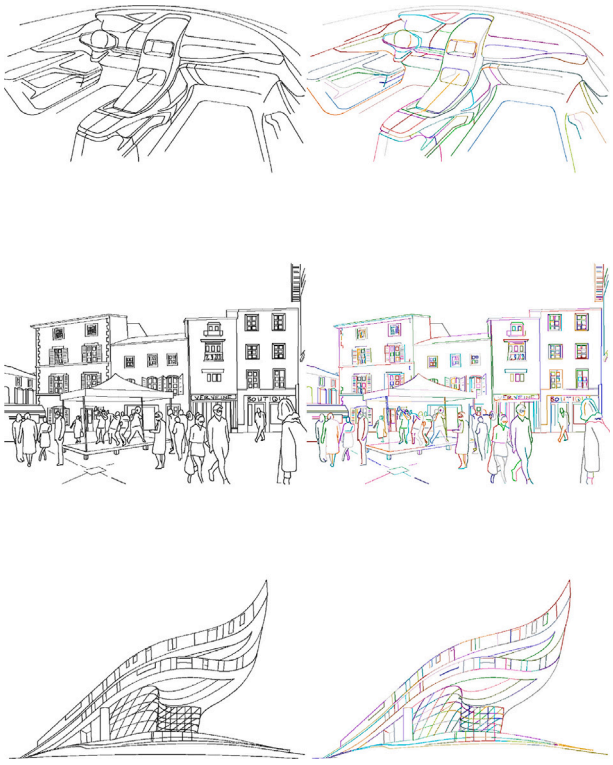


Fig. 13. In addition to the anime styles in our training set, our method is also applicable to complex scenarios such as architectural styles, all of which are derived from UW369.

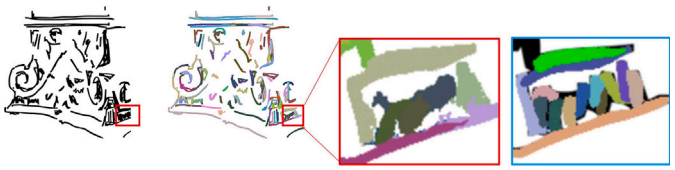


Fig. 14. Limitations of our automatic method. For a heavily shaded rough line drawing (a), when SK fails to capture the correct topological structure, unnatural strokes with excessive overlapping may appear (b, red box). The zoomed-in view of (b) is shown in (c). In such cases, user refinement (d) with reasonable prompts is required for correction. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

drawing. Then, the automatic segmentation results of several complicated drawings were loaded while the participants were asked to check the segmentation quality and edit the undesired results.

For the segmentation results and segmentation tools, we also provide 16 statements for the participants to rate on a scale from 1 to 5, where 1 corresponds to "strongly disagree" and 5 corresponds to "strongly agree." We report the results in Fig. 19.

Participants in our user study were asked to familiarize themselves with the system alone for 5 minutes after completing the tutorial and to perform the segmentation experiments as we requested. We conducted 3 experiments to verify the effectiveness of our UI in stroke segmentation and extraction.

Experiment 1: UI Comparison. This UI comparison experiment is specifically designed for single-stroke segmentation, and our automatic method is not applied in this part. In this experiment, we aim to understand the advantages and disadvantages of our single-stroke segmentation and extraction functions, compared to the tracking function in commercial software. Our system extracts strokes in two modes, click and trace, which are operationally similar to path fitting in Adobe Illustrator and drawing fitting in Photoshop, respectively. It is therefore necessary to make a comparison between these two existing functions. We set up four types of UI: 1) trace + edit, 2) click + edit, 3) manual + edit (trace w/o segmentation), and 4) Adobe Illustrator (click w/o segmentation). Given two line drawings, users segment the strokes in a random order with the four UIs and answered a questionnaire in Fig. 19 comparing UI1 with UI3 (Qa1–4), UI2 and UI4 (Qb1–4).

Since we implemented Bézier curve-based vectorization in manual mode, we have replaced the drawing adjustment in Photoshop with pure manual mode. Tracing based on drawing input is more sensitive to input strokes than point-and-click operations, and replacing the Photoshop UI with a manual mode ensures that the accuracy of the user's system input remains consistent when tracing

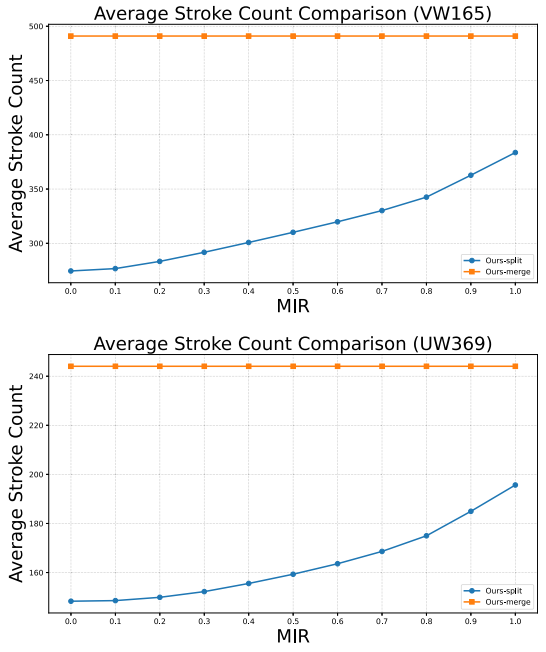


Fig. 15. The relationship between MIR and the average number of strokes. MIR can effectively reduce the average number of strokes for both VW165 and UW369.

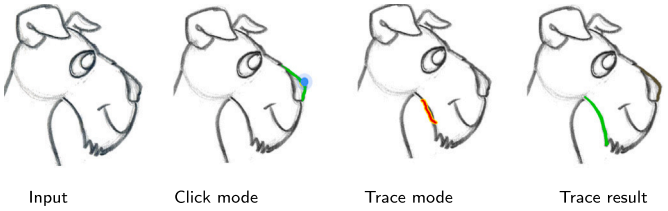


Fig. 16. Stability for varying width strokes with noise. Our method can extract varying-width strokes in noisy images interactively with both modes.

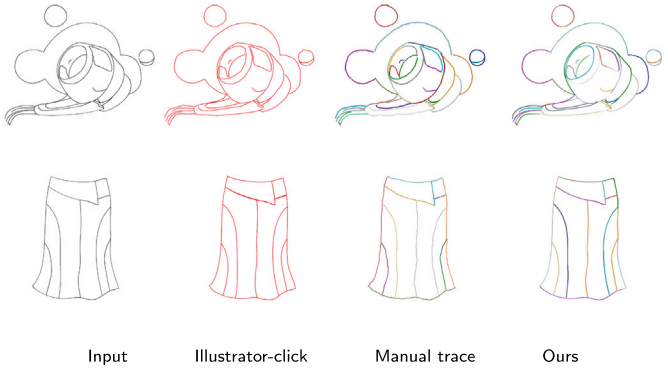


Fig. 17. Two examples for Experiment 1. Our method eliminates the "hand-shake" artifacts inherent in manual trace mode and the inaccuracies present in Illustrator-click mode. The results displayed in the Illustrator's UI inherently lack visualization of distinct strokes, whereas our UI can dynamically render and differentiate strokes in real-time using varying colors. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

with the drawing tablet. In each UI, the user can modify and delete strokes using edit mode, but can only trace strokes in one way (click or trace). In UIs 1–3, the user can switch views to view segmentation and vectorization results and make corrections.

Results. Participants generally agreed that our intelligent stroke extraction was more convenient than the traditional method, which converts imprecise input into precise strokes. Fig. 18 also supports this statement. On the two selected images, UIs 1 and 2, our intelligent stroke extraction algorithm, achieves segmentation and vectorization results that provide higher accuracy in less time, as Fig. 17 shows. Our video demonstrates more details of the experimental results.

Experiment 2: Segmentation Process Experience. To comprehensively evaluate the usefulness of the overall system, we had users perform stroke segmentation of complex sketches. We selected 20 complex line drawings, divided them into 4 groups, and

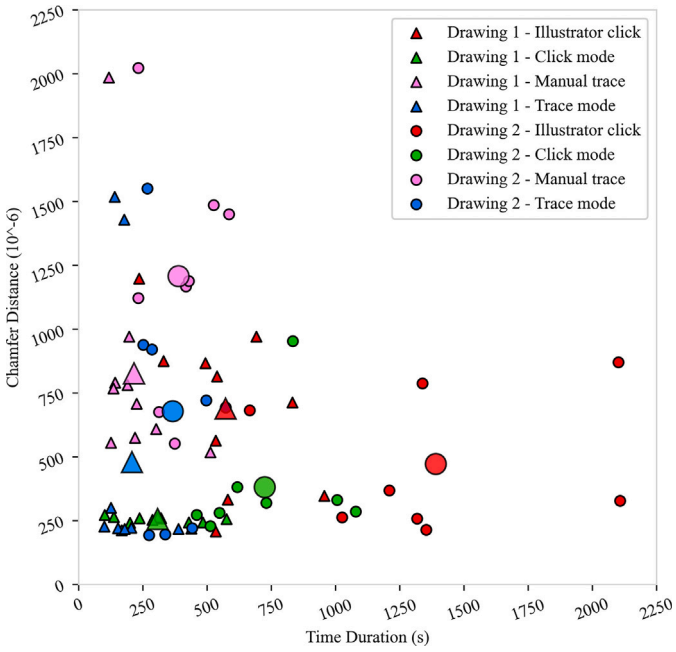


Fig. 18. Evaluation of time efficiency and reconstruction accuracy for each drawing by tool and participant. Lower values for time and Chamfer distance indicate better performance. Colors represent different interactive tools, while shapes correspond to distinct sample drawings. The larger mark represents the center of gravity for each label. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

each user segmented one of the groups of line drawings in random order, ensuring that at least 4 users worked on each group. Pre-segmentation of the strokes was performed using our automatic segmentation algorithm, and the results were automatically loaded after the user selected a line drawing. We asked the users to refine the stroke segmentation based on the segmentation according to their understanding and using all the features of our UI, followed by a questionnaire.

Results. Based on Qc1–8 in Fig. 19 shown, we discussed the following aspects: According to Qc1–4, the average scores from novices and experts are consistently high, which means our system is useful and easy to access (system usability). Qc3 and Qc7 show our automatic segmentation results and how edit mode helps both novices and experts.

According to Qc5 and Qc6, most users consider our single-stroke segmentation results to be of high quality. Meanwhile, more people tend to prefer strokes in trace mode compared to click mode. Qc8 shows that our overall user satisfaction is generally high.

At the end of the two experiments described above, we invited users to a semi-structured interview where they were asked to talk about the reasons for their ratings and the strengths and weaknesses of the system. Both experts and novices gave good ratings for the extraction of single strokes, mainly because they thought our system was more intelligent and that both click and trace modes were able to obtain the desired strokes by entering short and imprecise input (scribble or point). The main negatives are related to technical issues and the editing process: users who have used professional software would like to see our system implement features such as scaling or integrate with professional software as they have done; some users would like to see previews and single point extraction added to the click mode.

Experiment 3: Expert Interviews. To further validate the effectiveness and hidden potential of the system, we invited two additional experts to conduct Experiments 1 and 2 followed by the subsequent interviews. Expert E1 is an expert in 3D game scene design, proficient in Photoshop, and completed the tasks in Experiment 2 through remote operation. Expert E2 is an expert in vector animation, proficient in Adobe Illustrator, and visited our experimental site to complete Experiment 1 and Experiment 2. Both of them also provided positive feedback on our system. E1 remarked that our system was intelligent and could extract strokes easily, but noted that in scene design, it was more focused on illustration splicing rather than line drawing, and there was no need for 2D vectorization. However, he believes that our approach is better suited to the process of creating illustrations, and that there is potential for efficient stroke-level editing and animation on clean line drawings.

E2 said that the purpose of vectorization after stroke segmentation is to form a closed region for coloring, and our system can segment the correct strokes well and conveniently, and no longer needs tedious fitting and fine-tuning, which greatly improves the efficiency of segmentation. As he said, “I don’t need to spend much time segmenting strokes. It’s great that only minor adjustments are needed.” In addition, for complex line drawings, the automated method in our approach achieves a more reasonable stroke segmentation, and the combination of strokes after segmentation can be further customized.

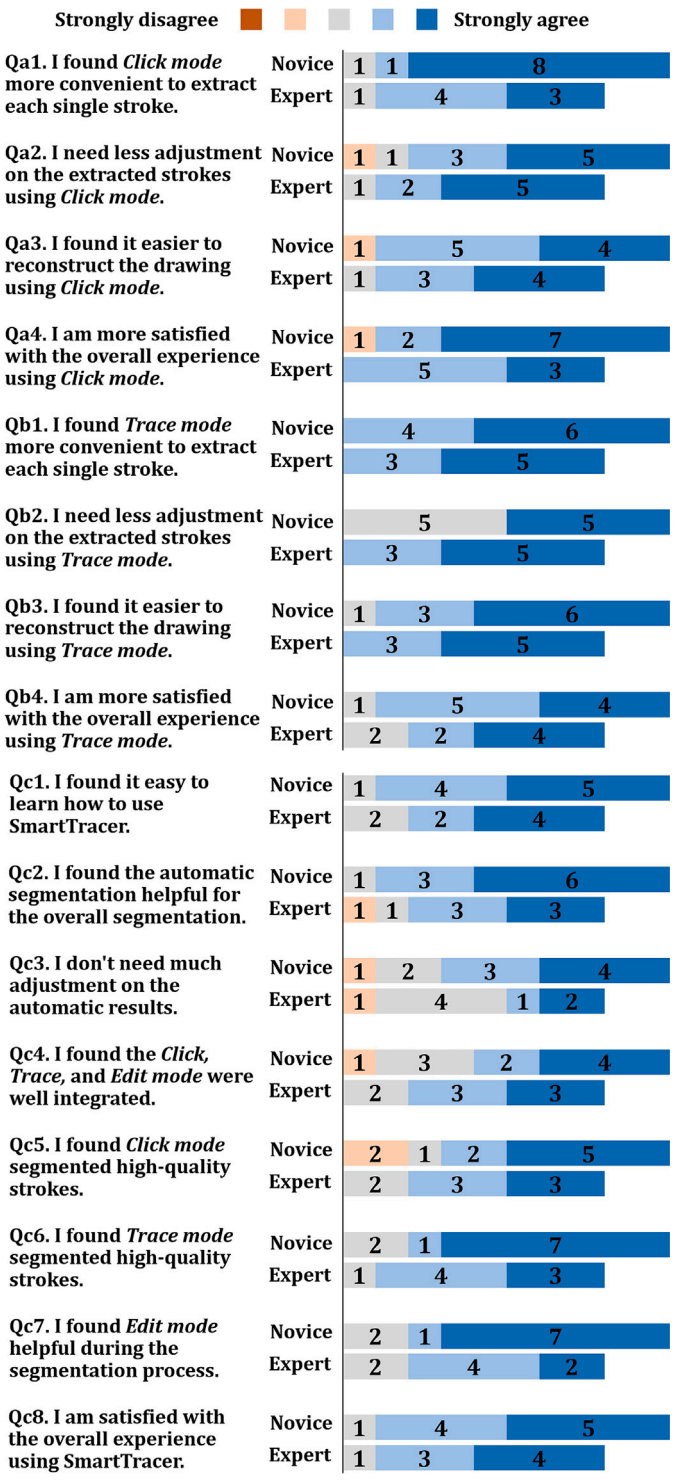


Fig. 19. Results from Experiment 2 collecting user feedback on our segmentation interface.

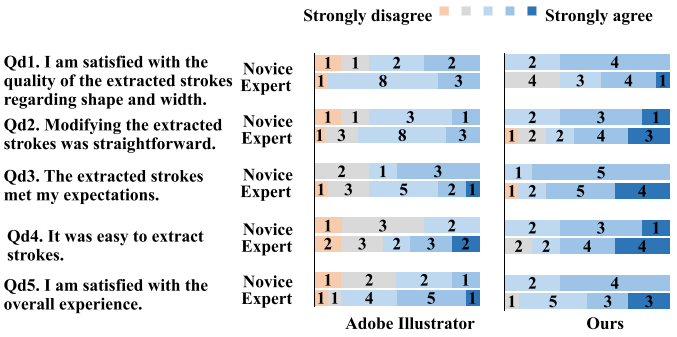


Fig. 20. Pipeline comparison results from Experiment 4. Our method outperforms Adobe Illustrator in all five aspects corresponding to Q1-Q5.

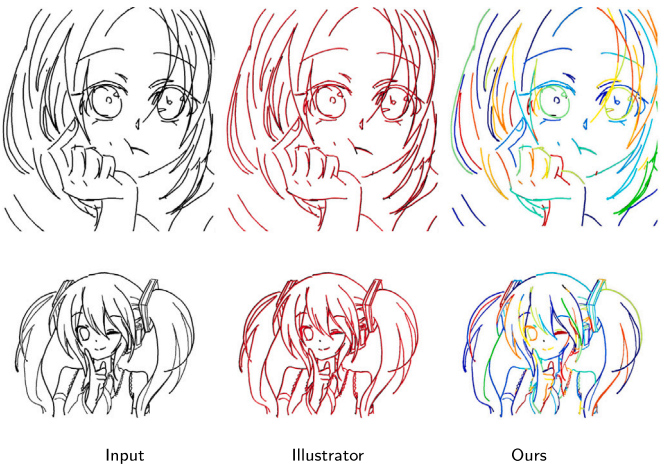


Fig. 21. Two examples for Experiment 4. The Illustrator's LineArt mode produces results lacking (1) stroke width information and (2) proper topological structure, making complete correction impossible within the allotted time. Our method enables efficient user-customized stroke segmentation under identical time constraints.

Experiment 4: Pipeline Comparison (Ours vs. Adobe Illustrator). To further validate our combined automated + interactive stroke processing framework, a comparative evaluation was executed between A) Our integrated solution ("Auto + Interactive") versus B) Adobe Illustrator's comprehensive toolkit —- users can use any tool for stroke refinement and extraction, specifically leveraging its LineArt image trace algorithm (LineArt mode) within the Image-Trace module for automatic stroke extraction. We conducted a randomized A/B evaluation by recruiting eighteen new participants, including twelve self-certified Adobe Illustrator experts who rated their software proficiency at ≥ 3 on a validated five-point Likert scale (where maximum = expertise mastery). Within each iteration of our A/B test, participants were required to perform maximal stroke segmentation refinements within strict five-minute intervals based on their respective auto-algorithm outputs. Before formal experimentation commencement, all participants underwent a mandatory five-minute orientation session designed for comprehensive familiarization with the complete suite of the stroke segmentation pipeline with both UIs.

Results. The results in Fig. 20 showed that ours was better in five aspects and were consistent with findings from Fig. 19 Qc1-8. During semi-structured interviews, experts noted: "LineArt mode is ineffective; I would like to manually trace each stroke rather than fix them," and observed that "The mode A (ours) lowers barriers for stroke extraction," which aligns with our original motivation and the visual results in Fig. 21.

7. Conclusion

This paper presents a novel approach to stroke extraction of clean line drawings with complex topologies using prompt-based neural networks and graph-based prompt simulation. By addressing the challenges of stroke extraction, particularly the user-specific pattern ambiguity and topological complexity of line drawings, our method offers a robust solution to single-stroke extraction. Our framework leverages high-resolution input and an interactive user interface, enabling personalized and precise segmentation. Our contributions include an efficient and stable CNN-based automatic segmentation method, a user-friendly interface for customizable segmentation, and extensive validation through experiments and user studies. Our findings underscore the potential for further innovations in intelligent stroke extraction and segmentation, paving the way for future research and development in stroke-based drawing analysis, stroke-level semantic labeling, and rendering. For limitations, since our automatic segmentation algorithm is based

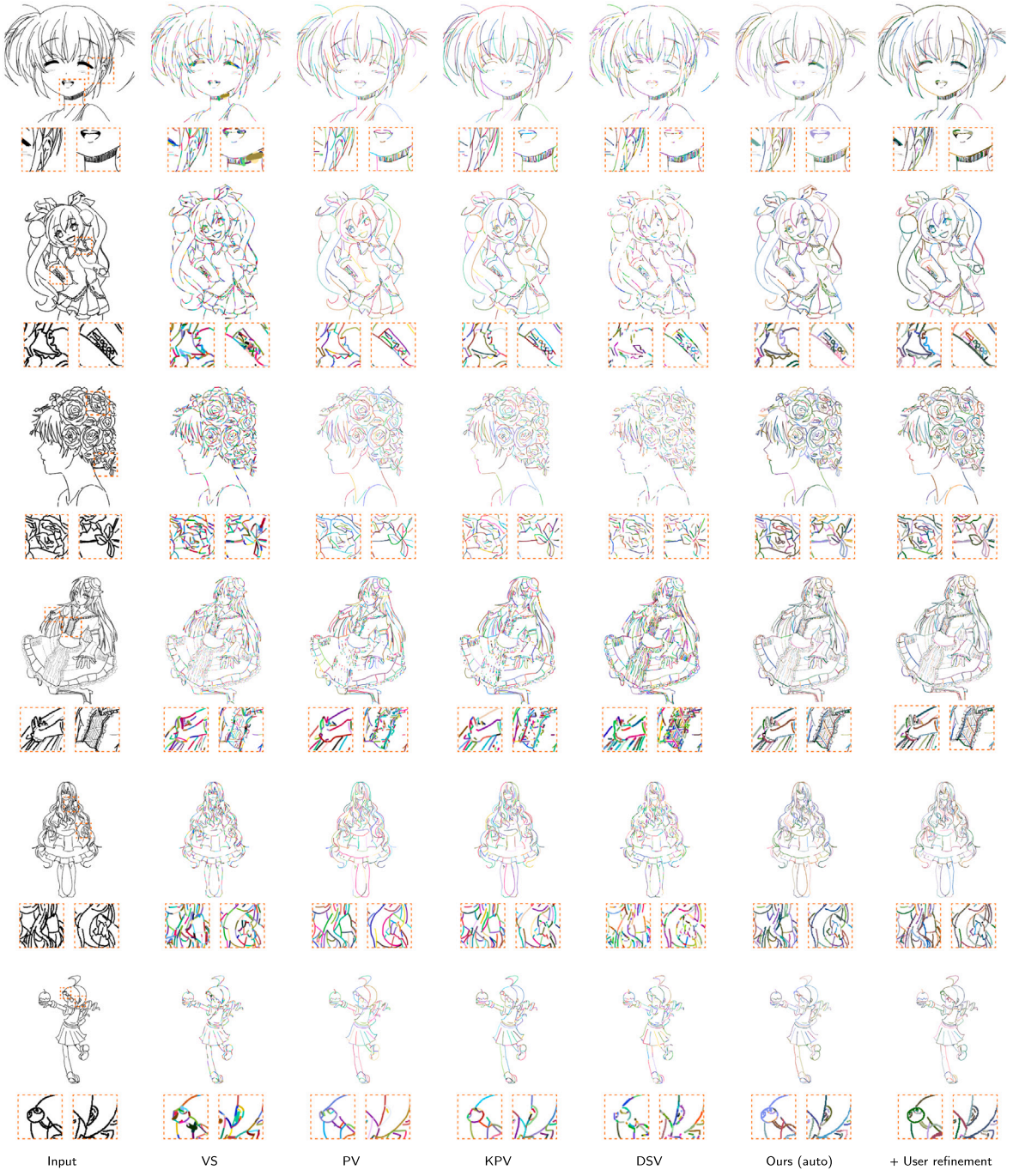


Fig. 22. Qualitative comparisons. Zoomed-in views show that VS, KPV, PV, and DSV produce inconsistent and unnatural line segments or missing fittings, whereas the segmented strokes generated by our methods maintain visual consistency and exhibit more reasonable results. Specifically speaking, VS fits thick strokes with repeated short curves, and other methods extract thin and uniformly wide strokes (shown as fixed-width bold) while ignoring stroke widening and overlapping; in contrast, our method splits the widening and overlapping strokes and allows for user refinement.

on the quality of the extracted skeleton before the graph construction, the skeleton extraction algorithm may generate erroneous topological prompts in certain complex cases. In such instances, manual input of correct prompts by the user is required to correct the results. In the future, we will further analyze the recorded stroke segmentation behaviors and examine user preferences and the sequence in which they perform tasks. Gaining deeper insights into user intentions will also inform more intuitive and user-friendly stroke segmentation methods.

CRediT authorship contribution statement

Zhengyu Huang: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Methodology, Investigation. **Zhongyue Guan:** Visualization, Investigation, Data curation. **Zeyu Wang:** Writing – review & editing, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (No. 62502410). We thank all the editors, reviewers, and participants for their contributions to this article.

Appendix A. Supplementary data

Supplementary data for this article can be found online at doi:10.1016/j.cagd.2026.102568.

Data availability

Data will be made available on request.

References

Berio, D., Leymarie, F.F., Asente, P., Echevarria, J., 2022. Strokestyles: stroke-based segmentation and stylization of fonts. ACM Trans. Graph. 41, https://doi.org/10.1145/3505246

Bessmeltsev, M., Solomon, J., 2019. Vectorization of line drawings via polyvector fields. ACM Trans. Graph. 38, https://doi.org/10.1145/3202661

Carlier, A., Danelljan, M., Alahi, A., Timofte, R., 2020. Deepsvg: A hierarchical generative network for vector graphics animation. arXiv:2007.11301.

Cole, F., Golovinskiy, A., Limpaecher, A., Barros, H.S., Finkelstein, A., Funkhouser, T., Rusinkiewicz, S., 2008. Where do people draw lines? ACM Trans. Graph. 27, https://doi.org/10.1145/1360612.1360687

Cole, F., Sanik, K., DeCarlo, D., Finkelstein, A., Funkhouser, T., Rusinkiewicz, S., Singh, M., 2009. How well do line drawings depict shape? In: ACM SIGGRAPH 2009 Papers. Association for Computing Machinery, New York, NY, USA, https://doi.org/10.1145/1576246.1531334

Egiazarian, V., Voynov, O., Artemov, A., Volkhonskiy, D., Safin, A., Taktasheva, M., Zorin, D., Burnaev, E., 2020. Deep vectorization of technical drawings. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J. (Eds.), Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII. Springer, pp. 582–598, https://doi.org/10.1007/978-3-030-58601-0_35

Eitz, M., Hays, J., Alexa, M., 2012. How do humans sketch objects? ACM Trans. Graph. 31, https://doi.org/10.1145/2185520.2185540

Favreau, J.-D., Lafarge, F., Bousseau, A., 2016. Fidelity VS. Simplicity: a global approach to line drawing vectorization. ACM Trans. Graph. 35, https://doi.org/10.1145/2897824.2925946

Gryaditskaya, Y., Sypesteyn, M., Hoftijzer, J.W., Pont, S., Durand, F., Bousseau, A., 2019. Opensketch: a richly-annotated dataset of product design sketches. ACM Trans. Graph. 38, https://doi.org/10.1145/3355089.3356533

Guo, Y., Zhang, Z., Han, C., Hu, W., Li, C., Wong, T., 2019. Deep line drawing vectorization via line subdivision and topology reconstruction. Comput. Graph. Forum 38, 81–90. https://doi.org/10.1111/CGF.13818

Gutan, O., Hegde, S., Berumen, E.J., Bessmeltsev, M., Chien, E., 2023. Singularity-free frame fields for line drawing vectorization. Comput. Graph. Forum 42, i–viii. https://doi.org/10.1111/CGF.14901

Ha, D., Eck, D., 2017. A neural representation of sketch drawings. CoRR arXiv:1704.03477.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016. IEEE Computer Society, pp. 770–778, https://doi.org/10.1109/CVPR.2016.90

Ito, S., Takagi, N., Sawai, K., Masuta, H., Motoyoshi, T., 2022. Fast semantic segmentation for vectorization of line drawings based on deep neural networks. In: International Conference on Machine Learning and Cybernetics, ICMLC 2022, Japan, September 9–11, 2022. IEEE, pp. 231–236, https://doi.org/10.1109/ICMLC56445.2022.9941326

Kim, B., Wang, O., Öztireli, A.C., Gross, M., 2018. Semantic segmentation for line drawing vectorization using neural networks. Comput. Graph. Forum. https://doi.org/10.1111/cgf.13365

Kingma, D.P., Ba, J., 2015. Adam: a method for stochastic optimization. In: Bengio, Y., LeCun, Y. (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings, http://arxiv.org/abs/1412.6980.

Liu, L., Lin, K., Huang, S., Li, Z., Li, C., Cao, Y., Zhou, Q., 2022. Instance segmentation for chinese character stroke extraction, datasets and benchmarks. arXiv preprint arXiv:2210.13826.

Magne, T., Sorkine-Hornung, O., 2025. Single-line drawing vectorization. Comput. Graph. Forum 44, 14.

Milletari, F., Navab, N., Ahmadi, S., 2016. V-net: fully convolutional neural networks for volumetric medical image segmentation. In: Fourth International Conference on 3D Vision, 3DV 2016, Stanford, CA, USA, October 25–28, 2016. IEEE Computer Society, pp. 565–571, https://doi.org/10.1109/3DV.2016.79

Mo, H., Simo-Serra, E., Gao, C., Zou, C., Wang, R., 2021. General virtual sketching framework for vector line art. ACM Trans. Graph. 40, https://doi.org/10.1145/3450626.3459833

Noris, G., Hornung, A., Sumner, R.W., Simmons, M., Gross, M., 2013. Topology-driven vectorization of clean line drawings. ACM Trans. Graph. 32, https://doi.org/10.1145/2421636.2421640

Pixiv, 2025. Pixiv: illustration communication service. https://www.pixiv.net/ (accessed: 7 April 2025).

Puhachov, I., Neveu, W., Chien, E., Bessmeltsev, M., 2021. Keypoint-driven line drawing vectorization via polyvector flow. ACM Trans. Graph. 40, https://doi.org/10.1145/3478513.3480529

Selinger, P., 2003. Potrace: a polygon-based tracing algorithm. http://potrace.sourceforge.net/potrace.pdf.

Simo-Serra, E., Iizuka, S., Ishikawa, H., 2018. Real-time data-driven interactive rough sketch inking. ACM Trans. Graph. 37, https://doi.org/10.1145/3197517.3201370

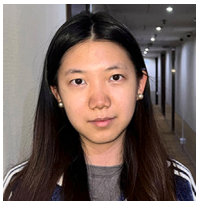
Wang, Z., Qiu, S., Feng, N., Rushmeier, H., McMillan, L., Dorsey, J., 2021. Tracing versus freehand for evaluating computer-generated drawings. ACM Trans. Graph. 40, https://doi.org/10.1145/3450626.3459819

Xiao, C., Zhang, C., Zheng, C., 2018. Fontcode: embedding information in text documents using glyph perturbation. ACM Trans. Graph. 37, :15:1–:15:16. https://doi.org/10.1145/3152823

Xiao, C., Su, W., Liao, J., Lian, Z., Song, Y.-Z., Fu, H., 2022. Differsketching: how differently do people sketch 3d objects? ACM Trans. Graph. 41, https://doi.org/10.1145/3550454.3555493
Yan, C., Vanderhaeghe, D., Gingold, Y., 2020. A benchmark for rough sketch cleanup. ACM Trans. Graph. 39, https://doi.org/10.1145/3414685.3417784
Yan, C., Li, Y., Aneja, D., Fisher, M., Simo-Serra, E., Gingold, Y., 2024. Deep sketch vectorization via implicit surface extraction. ACM Trans. Graph. 43, https://doi.org/10.1145/3658197
Zhang, T.Y., Suen, C.Y., 1984. A fast parallel algorithm for thinning digital patterns. Commun. ACM 27, 236–239. https://doi.org/10.1145/357994.358023
Zhang, Z., Liu, X., Li, C., Wu, H., Wen, Z., 2022. Shape-Aware Stroke Segmentation for Calligraphic Characters, https://api.semanticscholar.org/CorpusID:249634967.

Author biography

Zhengyu Huang is a lecturer in the Civil Aviation Electronic Information Engineering College at Guangzhou Civil Aviation College. He received a PhD from the Japan Advanced Institute of Science and Technology, advised by Prof. Kazunori Miyata, an MS from Northwest A&F University, advised by Prof. Shaojun Hu, and a BS from Jinan University, advised by Prof. Jieyuan Tang. His research interests include computer graphics, human-computer interaction, artificial intelligence, and AIGC. He has published several papers in international journals and conferences, including SIGGRAPH, CVM, and CAG.



Zhongyue Guan is a postgraduate student supervised by Professor Zeyu Wang. She is interested in computer graphics, particularly in non-photorealistic rendering, vector graphics, and animation. She obtained a Bachelor's degree in Integrative Systems and Design with an additional major in Computer Engineering from The Hong Kong University of Science and Technology.

Zeyu Wang is an Assistant Professor in the Thrusts of Computational Media and Arts (CMA) and Artificial Intelligence at The Hong Kong University of Science and Technology (Guangzhou), leading the Creative Intelligence and Synergy (CIS) Lab. He received a PhD from the Department of Computer Science at Yale University, advised by Profs. Julie Dorsey and Holly Rushmeier, and a BS (summa cum laude) from the Department of Machine Intelligence at Peking University, advised by Profs. Hongbin Zha and Katsushi Ikeuchi. His research interests include computer graphics, human-computer interaction, artificial intelligence, and digital cultural heritage. He has published over 60 papers in top international journals and conferences. He serves on the program committees of SIGGRAPH Asia, Eurographics, Pacific Graphics, etc. His research has been recognized with a CCF-Tencent Rhino-Bird Fellowship, an Adobe Research Fellowship, a Best Paper Award, and three Honorable Mention Awards.