

Article

Stroke2Font: A Hierarchical Vector Model with AI-Driven Optimization for Chinese Font Generation

Qing-Sheng Li ^{1,2,3,*}, Yu-Lin Bian ⁴ and Zhen-Hui Chai ⁴¹ School of Media and Information Engineering, Communication University of Zhejiang, Hangzhou 310018, China² Hangzhou Liangyun Intelligent Technology Co., Ltd., Hangzhou 310023, China³ Tongxiang Institute, Communication University of Zhejiang, Tongxiang 314500, China⁴ School of Communication and Journalism, Communication University of Zhejiang, Hangzhou 310018, China; bianyul@163.com (Y.-L.B.); achai928@163.com (Z.-H.C.)

* Correspondence: liqingsheng@cuz.edu.cn or aylqs@163.com; Tel.: +86-15058199136

Abstract

Chinese font generation is important for digital typography, cultural preservation, and personalized user interfaces. However, existing methods often face challenges in maintaining structural consistency, supporting diverse stylistic variations, and achieving computational efficiency simultaneously, especially in cloud-based environments. A key application is bandwidth-efficient font delivery, where compact structural templates replace large font files for on-demand style customization. To address these issues, this paper proposes Stroke2Font—a hierarchical vector model with AI-driven optimization for dynamic Chinese font generation. The core model decouples structural representation from style rendering through stroke element decomposition and Bézier curve parameterization. To further balance structural fidelity, style diversity, and real-time performance, we introduce a three-module optimization framework: (1) a reinforcement learning policy for dynamic selection of Bézier control parameters to minimize rendering latency; (2) a genetic algorithm for exploring style vector spaces and generating novel font variants; and (3) an adaptive complexity-aware optimization strategy that dynamically configures parameters based on character structural complexity. Experimental results on a dataset of 150 Chinese characters with 1123 stroke trajectories and 5287 feature points demonstrate that the adaptive complexity-aware optimization achieves the highest trajectory similarity of 65.2%, representing a 6.4% relative improvement over baseline (61.3%). The evaluation covers characters ranging from 1 to 18 strokes across 6 stroke types, with standard deviation reduced to $\pm 5.7\%$ (compared to $\pm 6.5\%$ baseline), indicating more consistent performance. Quantitative analysis confirms that the method generalizes effectively across varying character complexity, with the optimization showing stable improvement regardless of stroke count distribution. These results validate that Stroke2Font provides an effective solution for high-quality, efficient, and scalable Chinese font generation in cloud-based applications.

Keywords: Chinese font generation; hierarchical vector model; stroke element representation; AI-driven optimization; Bézier curve parameterization; complexity-aware optimization



Received: 9 February 2026

Revised: 6 March 2026

Accepted: 12 March 2026

Published: 18 March 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Chinese font generation represents a fundamental challenge in digital typography, cultural preservation, and human–computer interaction [1]. With over 50,000 distinct characters in the Unicode CJK Unified Ideographs block and countless stylistic variations

across calligraphic traditions, the computational synthesis of Chinese glyphs demands approaches that can simultaneously respect structural constraints, capture esthetic qualities, and scale efficiently to large character sets [2,3]. With the proliferation of mobile devices and social media, there is a growing demand for customized and stylized Chinese fonts that reflect individual taste and contextual appropriateness [4].

However, the creation of high-quality Chinese fonts remains a labor-intensive and time-consuming process. A standard Chinese font library contains thousands of glyphs (e.g., 6763 in the GB2312 standard [5]), each requiring meticulous design, stroke adjustment, and esthetic harmonization [6]. Although numerous font libraries are commercially available, the need for novel, personalized, and context-specific typefaces continues to outpace supply [7]. Moreover, traditional font files are monolithic in nature—each style requires a complete set of glyph outlines, leading to significant storage and transmission overhead, especially in cloud and web environments [8].

In recent years, computational approaches to font generation have emerged, broadly categorized into explicit feature-based methods and deep learning-based methods. Early feature-based techniques decompose characters into strokes, radicals, or structural templates, then reassemble them through rule-based or optimization-driven processes [9–11]. While offering interpretability and control, these methods often require extensive manual annotation and struggle to capture nuanced stylistic variations. Conversely, deep learning methods, particularly those based on Generative Adversarial Networks (GANs) [12–16], learn font styles in an end-to-end manner from image pairs. Although capable of producing visually appealing results, they frequently suffer from mode collapse, stroke discontinuity, and limited structural fidelity, especially when training data are scarce or styles are highly divergent [17].

This fundamental trade-off between structural fidelity and stylistic expressiveness mirrors a core challenge in computer vision and generative modeling: the disentanglement of content (structure) [18] from style (appearance) [19]. Pioneering works in neural style transfer [20,21] and image-to-image translation [22] demonstrated the separation and recombination of these factors, yet they operate primarily at the pixel level, lacking the explicit, hierarchical semantics required for structured graphical objects like characters. Recent advances in object-centric representation learning [23] and neural vector graphics [24,25] have made strides toward structured and editable visual generation. However, directly applying these paradigms to Chinese characters—with their unparalleled topological complexity and vast glyph set—remains an open problem, as they do not natively incorporate the linguistic and graphic design priors inherent to font engineering.

Consequently, a truly effective font generation system must simultaneously address three intertwined challenges: (1) preserving the structural integrity of complex hieroglyphic compositions; (2) enabling rich and nuanced stylistic flexibility; and (3) achieving high computational efficiency for real-time, cloud-native applications. Most existing methods excel in at most one of these dimensions: Feature-based approaches prioritize structure at the cost of style diversity and automation; deep learning models capture style but often falter on structure and efficiency, while few consider the system-level performance required for scalable deployment.

To address these challenges, we present Stroke2Font, a novel framework that unifies a hierarchical vector-based core model with an AI-driven optimization layer. Our core model provides the foundational control over structure and style, while the optimization layer intelligently automates parameter selection, explores the design space, and manages resources—ensuring the system is not only high-quality but also efficient and scalable.

Stroke2Font targets cloud-based font services where applications need on-demand style customization. Instead of distributing pre-rendered font files (2–10 MB each),

the server transmits only compact structural templates (~ 0.8 KB/character) plus two style parameters (α , η). The client reconstructs stylized glyphs locally in real time (<15 ms/character), reducing bandwidth by $\sim 90\%$.

The contributions of this paper are summarized as follows:

- (1) We formally define the stroke element (笔元) as a sub-stroke trajectory unit, a hierarchical stroke element representation that decouples structure from style, enabling explicit control over glyph topology and local stroke attributes.
- (2) We develop a vector-based style manipulation framework using yoke vectors and Bézier fitting, allowing continuous interpolation between font styles without retraining.
- (3) We introduce an AI-driven optimization framework consisting of: a reinforcement learning agent for real-time rendering optimization; a genetic algorithm for style space exploration; and an adaptive complexity-aware optimization strategy that dynamically configures parameters based on character structural complexity.
- (4) We implement a cloud-compatible font service architecture that significantly reduces bandwidth and storage requirements while supporting dynamic, personalized font generation.
- (5) We conduct extensive experiments demonstrating that Stroke2Font outperforms existing approaches in structural accuracy, style diversity, and computational efficiency, particularly under real-time and concurrent access scenarios.

The remainder of this paper is organized as follows. Section 2 reviews related work in Chinese font generation, stroke representation, and AI-driven optimization. Section 3 details our stroke element representation and parametric style generation. Section 4 presents the AI-driven optimization framework. Section 5 presents experimental results and comparative analysis, and Section 6 concludes.

2. Related Work

The automatic generation of Chinese fonts has been studied for decades, with approaches evolving from rule-based systems to data-driven deep learning models. In this section, we review existing work from three perspectives: (1) explicit feature-based methods, (2) deep learning-based methods, and (3) hybrid or structure-aware approaches.

2.1. Explicit Feature-Based Methods

Early research in Chinese font generation focused on explicit structural decomposition and geometric modeling. These methods typically extract low-level features such as strokes, radicals, and topological relations, then reassemble them through parametric or procedural means.

Stroke-based synthesis is one of the most intuitive strategies. For example, Pang and Feng [26] proposed a method that segments Chinese characters into strokes, builds a stroke library, and reconstructs new glyphs by rearranging stroke outlines—though it requires extensive manual post-processing. Similarly, Yang et al. [7] employed C-Bézier curves to model stroke contours, allowing style variation through control point adjustment. While effective for regular typefaces, these methods struggle with cursive or highly stylized fonts where stroke boundaries are ambiguous.

Radical-based assembly offers a higher-level compositional approach. Lin et al. [8] introduced a greedy algorithm that decomposes a small set of sample characters into radicals, then recombines them to form unseen characters. This approach reduces the number of required samples but depends heavily on the completeness and compatibility of the radical library.

Although feature-based methods provide interpretability and precise control, they suffer from several limitations: (1) They rely on handcrafted feature extraction, which is

labor-intensive and error-prone; (2) they often fail to capture subtle style nuances, especially in handwritten or calligraphic fonts; (3) they lack flexibility in generating entirely new styles without redesigning stroke or radical templates.

2.2. Deep Learning-Based Methods

With the advent of deep learning, especially Generative Adversarial Networks (GANs) [9], Chinese font generation has shifted toward data-driven, end-to-end frameworks, as extensively reviewed in the recent literature [15]. These methods treat font generation as an image-to-image translation task [27], learning mappings from source style to target style without explicit structural modeling.

Pix2pix-based approaches laid the groundwork for style transfer in Chinese fonts. Tian proposed zi2zi [28], a conditional GAN that translates standard printed characters into stylized ones. While demonstrating impressive results, such methods often produce artifacts such as broken strokes or blurred contours when structural differences between source and target are large.

To improve structural consistency, later works introduced auxiliary structural cues. Jiang et al. [29] proposed SCFont, which integrates stroke-level semantics and structural constraints into a deep stacked network, guiding the generation of target-style skeleton images to preserve glyph topology. Wu et al. [30] introduced CalliGAN, a structure-aware framework that incorporates stroke sequence information derived from a Chinese character decomposition system to calibrate fine-grained styles and maintain structural integrity. These methods show improved robustness but still struggle with mode collapse and style inconsistency in fine-grained details.

Building on this structure-aware direction, recent advances have focused on enhancing controllability and reducing data dependency. Wang et al. [31] proposed Attribute2Font, which leverages font attributes and a semi-supervised attention mechanism for controllable style synthesis, offering a step towards more interpretable generation. Others have tackled the data bottleneck; for instance, Jiang et al. [32] designed W-Net, a one-shot generation network that learns style from a single character reference. However, despite these improvements, a common weakness persists: The inherent black-box nature of deep learning approaches still limits fine-grained, user-intuitive controllability and often necessitates large, paired datasets for robust training. These limitations have prompted a parallel line of research that seeks to fundamentally redefine the representation and generation paradigm itself, moving beyond pixel-based translation towards more explicit, structured, and editable models.

2.3. Hybrid and Structure-Aware Approaches

Recognizing the complementary strengths of explicit features and deep learning, and driven by the need to overcome the limitations of pixel-based methods, several studies have explored hybrid models that integrate structural priors with neural networks.

Graph-based representations have been used to encode topological relations between strokes. For example, Wen et al. [20] proposed ZiGAN, which uses a few-shot style transfer approach guided by stroke-level graphs. Similarly, Ning [33] combined self-attention mechanisms with StarGAN v2 for multi-style migration, showing improved style disentanglement.

Vector-based generative models represent another promising direction. Recent work in vector graphics generation, such as DeepSVG [34] and Im2Vec [35], has inspired font generation methods that operate directly on parametric curves rather than pixels. These approaches naturally support resolution-independent rendering and editable outputs, though they have not been extensively applied to Chinese characters due to their structural

complexity. Our proposed Stroke2Font framework is positioned within this vector-based paradigm, aiming to address the unique challenges of Chinese characters by introducing a hierarchical stroke element representation and a continuous style space, thereby enabling both structural fidelity and rich stylistic control.

2.4. Our Positioning

Our work is situated at the intersection of explicit structural modeling and parametric style control. Unlike pure deep learning methods, we adopt a hierarchical stroke element representation that explicitly captures glyph topology. Unlike traditional feature-based methods, we use vector operations and Bézier curves to enable continuous style interpolation without manual redesign. Compared to recent hybrid approaches, our model is lighter, more interpretable, and cloud-native, making it suitable for real-time and personalized font services.

While existing methods have made progress in font style transfer, they often neglect explicit structural modeling and lack support for efficient cloud-based generation. To address these limitations, we propose a stroke element-based model that decouples structure from style through a hierarchical vector representation. The details of our approach are presented in the following section.

3. Proposed Model

3.1. Overview of the Framework

The core idea of our model is to decouple structural representation from stylistic rendering in Chinese font generation. As illustrated in Figure 1, our framework follows a hierarchical pipeline: Characters are first decomposed into stroke elements, which are represented as vectors; these vectors are then stylized via Bézier curve fitting; finally, the styled strokes are reassembled according to structural templates for output.

The model operates in two phases (Figure 1):

Structure Encoding: A character is abstracted into a stroke element graph, and stored as a lightweight template.

Style Rendering: Given style parameters, stroke elements are deformed via vector operations and rendered as Bézier curves.

This separation enables efficient cloud-based font services, where structural templates reside on the server and style parameters are transmitted on-demand to clients.

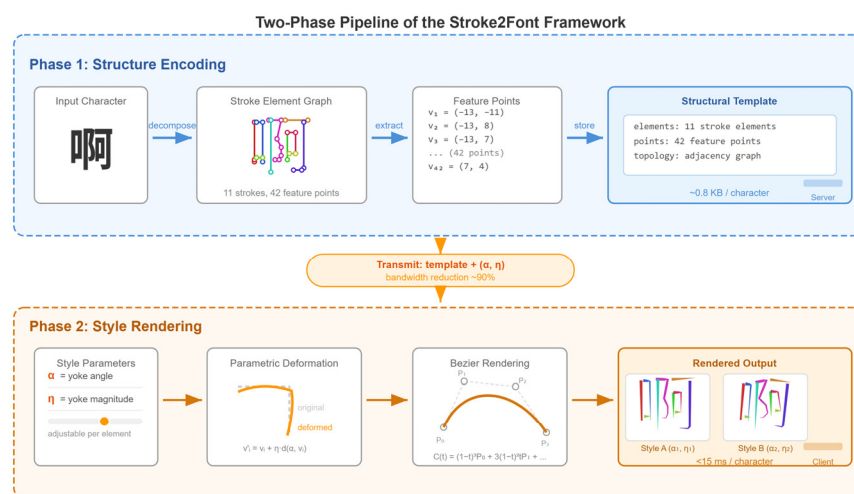


Figure 1. Two-phase pipeline of the Stroke2Font framework. Different colors distinguish individual stroke elements to illustrate structural correspondence across pipeline stages.

3.2. Structural Representation

Following the principle of separation of structure and style established in Section 3.1, we extract invariant topological skeletons from diverse font instances. Figure 2 demonstrates how our stroke element representation captures both structural invariance and stylistic variation. Two renderings of “福”—one in geometric Heiti font, another in expressive Xingkai script—are shown alongside their abstracted structural skeletons. Despite their visual differences, both share an identical topological template (right), confirming that our representation successfully decouples persistent structure from variable style.

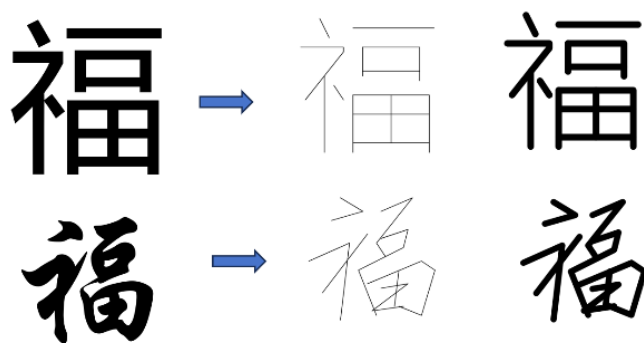


Figure 2. Abstraction of some basic structure.

Each stroke is abstracted into a set of feature points connected by directional vectors [35]. The extraction process, illustrated in Figure 3 using a “dot” (点) stroke as an example, involves four key steps: (1) contour extraction, (2) identification of semantically significant feature points, (3) vectorization of point coordinates, and (4) derivation of stroke segment vectors. This representation transforms continuous stroke contours into discrete, mathematically manipulable elements.

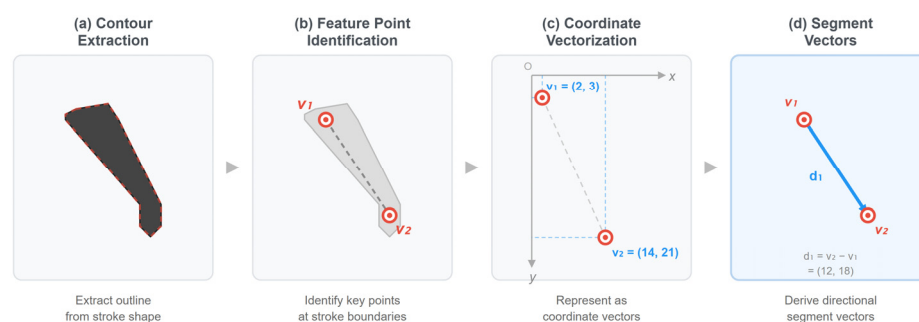


Figure 3. Feature point extraction process for a “dot” (点) stroke.

Beyond static geometry, personal writing styles are encoded in kinematic properties such as pen velocity, pressure, and stroke sequencing. We capture these dynamics as ordered sequences of feature points $\{s, i_1, i_2, \dots, i_n, e\}$, where s and e denote start and end points, and i_k represents intermediate inflection points. The connections between these points form stroke segment vectors, as shown in Figure 4, where the S5 stroke of the character “阿” is decomposed. The four temporally ordered feature points $\{s, i_1, i_2, e\}$ (Figure 5 blue badges) produce three distinct segment vectors: a vertical downward vector $d_1 = (0, 23)$, a turning vector $d_2 = (-1, 1)$, and a horizontal hook vector $d_3 = (-6, 0)$, illustrating the general formulation $d_k = v_{k+1} - v_k$.

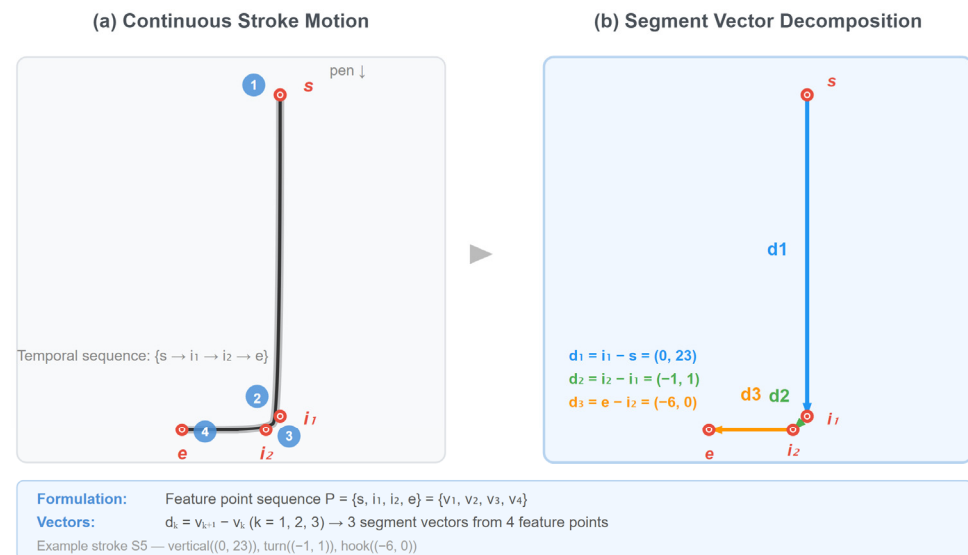


Figure 4. Stroke segment vectors from temporal feature point sequence. (a) Continuous stroke motion—shows the original stroke shape with numbered feature points $\{s, i_1, i_2, e\}$ in temporal (writing) order, with blue badges ①②③④. (b) Segment vector decomposition—same points but with color-coded directional arrows: d_1 (blue) = (0, 23)—vertical downward. d_2 (green) = (-1, 1)—turning/inflection. d_3 (orange) = (-6, 0)—horizontal hook leftward. Bottom annotation bar shows the general formulation: $d_k = v_{k+1} - v_k$, demonstrating how 4 feature points produce 3 segment vectors.

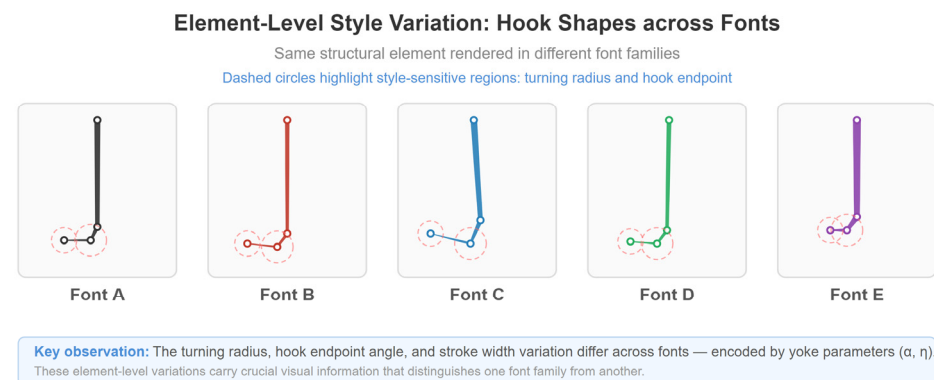


Figure 5. Element-level style variation: the same structural element (竖弯钩, vertical–turning–hook) rendered across five font families (Font A–E). Dashed circles highlight style-sensitive regions—the turning radius and hook endpoint—where font-specific identity is most strongly expressed.

3.3. Stylistic Variation

The stroke element representation naturally accommodates stylistic variation at two levels:

Element-Level Style Variation. Within a single stroke element, subtle shape differences encode font-specific identity. For instance, Figure 5 illustrates how the hook (钩) shape—one of the most style-sensitive stroke elements—varies across five representative font families (Font A–E). The differences manifest in three key geometric properties: turning radius at the inflection point, hook endpoint angle, and stroke width taper. These variations, though small in absolute displacement, carry crucial visual information that distinguishes one font from another and are precisely captured by the yoke vector parameters (α, η) .

Character-Level Style Synthesis. At the character level, the combination of styled stroke elements must maintain global consistency to produce a coherent font identity. As shown in Figure 6, the character “阿” is rendered in four distinct styles—Original (neutral), Style A, Style B, and Style C—by applying a single (α, η) pair uniformly across all seven

stroke elements. The resulting characters exhibit holistic stylistic coherence: Style A consistently shifts rightward-pointing segments, Style B uniformly adjusts stroke weight in the opposite direction, and Style C introduces a diagonal emphasis throughout. This consistency is guaranteed by the yoke vector formulation (Section 3.5.1), where the deformation $v'_i = v_i + \eta \cdot d(\alpha, v_i)$ applies the same directional bias to every stroke element, producing coherent character-level esthetics from element-level transformations.

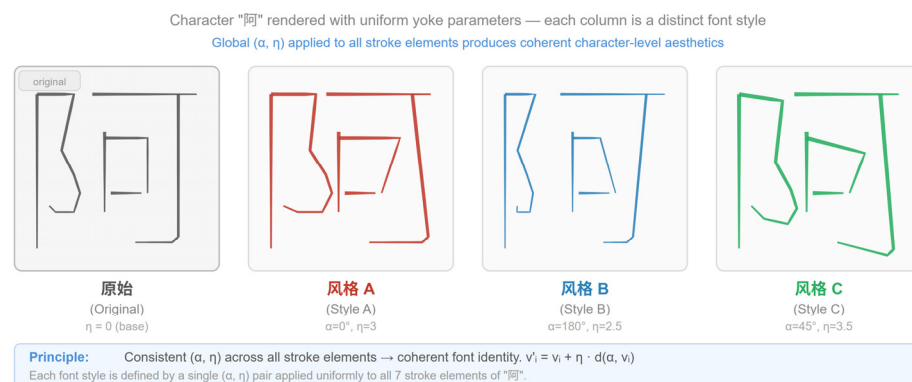


Figure 6. Character-level style synthesis: the character “阿” rendered with uniform yoke parameters (α, η) across all stroke elements. Each column represents a distinct newspaper font style. Consistent parameter application ensures that element-level deformations aggregate into a coherent character-level identity.

Individual writing habits constitute a spectrum of unique stylistic signatures. To model this continuum, our approach parameterizes handwriting style, enabling the synthesis of a wide variety of personalized glyphs through controlled parameter variation.

Figure 7 diverse machine-generated handwriting styles for the same character, produced by interpolating within a continuous style parameter space.

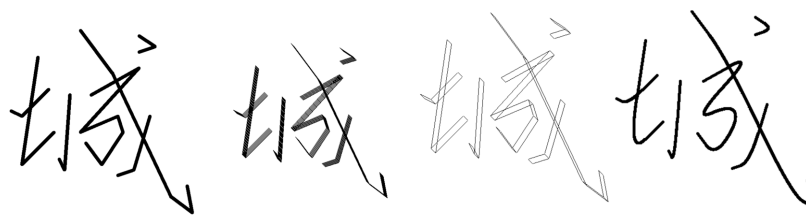


Figure 7. Handwriting style interpolation within the continuous parameter space.

As illustrated in Figure 7, by navigating this parameter space, our model can generate outputs ranging from a normative Base Style to highly personalized Transformed Styles, effectively emulating the natural diversity of human handwriting. Capturing such a nuanced style continuum is critical for applications like digital handwriting synthesis and personalized interfaces.

This work is grounded in the paradigm of structured, stroke-based Chinese character generation. It builds directly upon the intelligent animation system proposed in [36], which introduces a dynamic dataset architecture and decoupled framework for generating character animations via stroke feature extraction and parametric control. We extend this core philosophy from the temporal domain of animation to the stylistic domain of static font generation. Our key advancement is the formalization of a continuous style parameter space, which provides a unified model for smooth interpolation and fine-grained control over handwriting esthetics, thereby addressing the critical challenge of style diversity and personalization.

3.4. Stroke Element Representation: A Hierarchical Vector Model

Building upon prior work in dynamic Chinese character description [37–39], we introduce stroke elements as atomic representation units finer than traditional strokes. This hierarchical vector representation enables precise, independent control over both structural geometry and stylistic attributes while maintaining topological consistency, a core requirement for high-quality font generation.

3.4.1. Stroke Element Definitions

A stroke element E_n is a composite structure comprising n contiguous, ordered stroke segment vectors:

$$E_n = (S_{i_1j_1}, S_{i_2j_2}, \dots, S_{i_nj_n})$$

For notational simplicity, we denote $S_k = S_{i_kj_k}$, yielding $E_n = (S_1, S_2, \dots, S_n)$.

Given a stroke defined by two terminal feature points v_s and v_e , with m intermediate stationary points $\{v_{z_1}, v_{z_2}, \dots, v_{z_m}\}$, the complete stroke can be expressed as the concatenation of its constituent stroke elements:

$$T(v_s, v_e) = \sum_{k=0}^m Y(v_{z_k}, v_{z_{k+1}})$$

where $Y(v_{z_k}, v_{z_{k+1}})$ represents the stroke element between consecutive points (with $v_{z_0} = v_s$ and $v_{z_{m+1}} = v_e$).

3.4.2. A Three-Module Hierarchical Taxonomy

Our representation organizes stroke elements into a three-module hierarchy, enabling granular control from basic strokes to complex radicals.

Level 1: Basic stroke elements correspond to the fundamental strokes of Chinese calligraphy, each defined by exactly two feature points. Their generic form is as follows:

$$S_{12} = (x_1, y_1, x_2, y_2)$$

Table 1 summarizes the five basic stroke elements, their visual forms, and vector representations.

Table 1. Basic stroke elements and Their Vector Representations.

ID	Name (Pinyin/Chinese)	Feature Points	Glyph Illustration	Vector Representation
1	Pie (撇)	v_1, v_2		$S_{12} = (x_1, y_1, x_2, y_2)$
2	Na (捺)	v_1, v_2		$S_{12} = (x_1, y_1, x_2, y_2)$
3	Heng (横)	v_1, v_2		$S_{12} = (x_1, y_1, x_2, y_2)$
4	Shu (竖)	v_1, v_2		$S_{12} = (x_1, y_1, x_2, y_2)$
5	Dian (点)	v_1, v_2		$S_{12} = (x_1, y_1, x_2, y_2)$

Level 2: Combined stroke elements model more complex strokes by concatenating multiple basic elements. For instance, a stylized “Pie” variant is expressed as follows:

$$CPie = \sum_{k=1}^i Pie_k$$

Table 2 enumerates common combined stroke elements, showing how they are constructed from basic primitives.

Table 2. Combined stroke elements.

ID	Combined Element	Feature Points	Composition Formula
1	CPie	$\mathbf{v}_1, \dots, \mathbf{v}_i$	$\mathbf{CPie} = \sum_{k=1}^i \mathbf{Pie}_k$
2	CNa	$\mathbf{v}_1, \dots, \mathbf{v}_i$	$\mathbf{CNa} = \sum_{k=1}^i \mathbf{Na}_k$
3	CHeng	$\mathbf{v}_1, \dots, \mathbf{v}_i$	$\mathbf{CHeng} = \sum_{k=1}^i \mathbf{Heng}_k$
4	CShu	$\mathbf{v}_1, \dots, \mathbf{v}_i$	$\mathbf{CShu} = \sum_{k=1}^i \mathbf{Shu}_k$
5	CDian	$\mathbf{v}_1, \dots, \mathbf{v}_i$	$\mathbf{CDian} = \sum_{k=1}^i \mathbf{Dian}_k$

Level 3: Extended stroke elements approximate radical-like structures by applying geometric transformations (e.g., negation for direction inversion) to basic or combined elements. Examples include the following:

$$\mathbf{Ti} = -\mathbf{Pie}, \mathbf{RGou} = -\mathbf{Pie}, \mathbf{LGou} = -\mathbf{Na}$$

Table 3 provides a concise summary of representative extended stroke elements.

Table 3. Extended stroke elements.

ID	Extended Element	Feature Points	Expression
1	Ti (提)	$\mathbf{v}_1, \mathbf{v}_2$	$\mathbf{Ti} = -\mathbf{Pie}$
2	RGou (右钩)	$\mathbf{v}_1, \mathbf{v}_2$	$\mathbf{RGou} = -\mathbf{Pie}$
3	LGou (左钩)	$\mathbf{v}_1, \mathbf{v}_2$	$\mathbf{LGou} = -\mathbf{Na}$

3.4.3. Visualizing the Representation Pipeline

Figure 8 demonstrates the complete pipeline from feature point extraction to stylized character generation using the character “翱” as a case study. The figure visually decomposes the character into its constituent stroke elements, illustrating the vector-based skeletal structure. Figure 8b,c further illustrate the generation of two distinct functional variants from the same underlying structural representation: an arrow-guided handwriting font designed for learners (Figure 8b), and a standard typeface optimized for digital typography (Figure 8c).

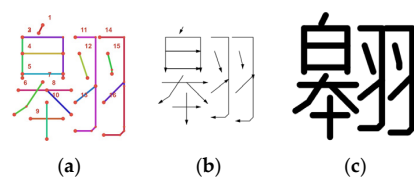


Figure 8. From feature points to stylized characters. (a) Structural decomposition of the original character. (b) Pedagogical font for learners. (c) Professional font for publication.

3.4.4. Advantages and Implications

This hierarchical stroke element representation provides several key advantages for font generation:

Structural Fidelity: The explicit vector graph preserves the precise topological relationships of the original character.

Parametric Flexibility: Style can be manipulated by adjusting the parameters (angles, norms) of individual stroke segment vectors without altering the underlying structure.

Storage and Transmission Efficiency: By separating reusable structural templates (server-side) from style parameters (client-side), storage overhead is dramatically reduced, and bandwidth requirements are minimized for cloud-based services.

Editability and Scalability: The atomic nature of stroke elements allows for fine-grained editing and supports the efficient generation of large font families from a compact representation.

This representation forms the mathematical backbone of the Stroke2Font model, enabling the subsequent AI-driven optimization framework detailed in Section 4.

3.5. Parametric Style Generation via Vector Manipulation

Having established the stroke element representation (Section 3.4), we now describe how stylistic variation is achieved through parametric deformation of stroke elements. The core mechanism is the yoke vector—a two-parameter control (angle α , magnitude η) attached to each stroke element that governs its geometric transformation.

3.5.1. Yoke Vector Definition

Each stroke element En carries a yoke vector $\mathbf{y} = (\alpha, \eta)$ that controls its stylistic deformation:

Angle parameter α ($0^\circ \leq \alpha \leq 360^\circ$): Defines the direction of deformation relative to the stroke element's base vector. Varying α rotates the stylistic emphasis—for example, shifting a horizontal stroke's "weight" from left-leaning to right-leaning.

Magnitude parameter η ($\eta > 0$): Controls the extent of deformation. Larger η produces more pronounced stylistic variation (bolder strokes, more exaggerated curves), while smaller values yield subtler, more conservative rendering.

Given a stroke element with feature points $\{v_1, v_2, \dots, v_k\}$, the deformed positions are computed as follows:

$$\mathbf{v}'_i = \mathbf{v}_i + \eta \cdot \mathbf{d}(\alpha, \mathbf{v}_i)$$

where $\mathbf{d}(\alpha, \mathbf{v}_i)$ is a direction function that determines the local deformation vector based on the yoke angle α and the point's position within the stroke element. This formulation ensures that:

1. Setting $\eta = 0$ recovers the original (neutral-style) trajectory.
2. The same structural template can produce different styles by varying only (α, η) .
3. Style interpolation between two variants is achieved by linearly interpolating their (α, η) pairs.

The parametric control enabled by yoke vectors is visually demonstrated in Figures 9–11.

Figure 9 illustrates the directional effect of α : with fixed $\eta = 0.5$, varying α rotates the deformation direction on a representative “横折” (horizontal-turning) stroke element, producing distinct stylistic emphases (e.g., $\alpha = 0^\circ$ pushes rightward, $\alpha = 90^\circ$ pushes downward). Figure 10 demonstrates the magnitude effect of η : with fixed $\alpha = 45^\circ$, increasing η from 0 (original) to 1.0 produces progressively more pronounced deformation, confirming that $\eta = 0$ recovers the original trajectory. Figure 11 provides a radial overview of the combined (α, η) parameter space, showing eight deformation directions around the original stroke element at $\eta = 0.6$.

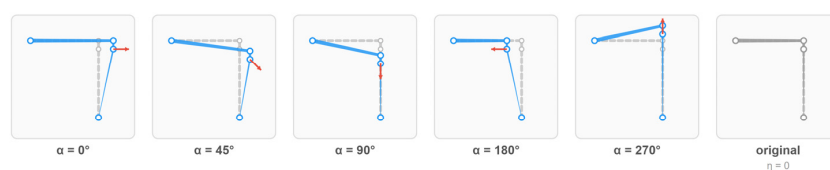


Figure 9. Effect of yoke angle α on deformation direction. $\mathbf{v}'_i = \mathbf{v}_i + \eta \cdot \mathbf{d}(\alpha, \mathbf{v}_i)$ ($\eta = 0.5$ fixed, stroke element: 横折). Red arrows show deformation direction $\mathbf{d}(\alpha)$; dashed gray = original trajectory.

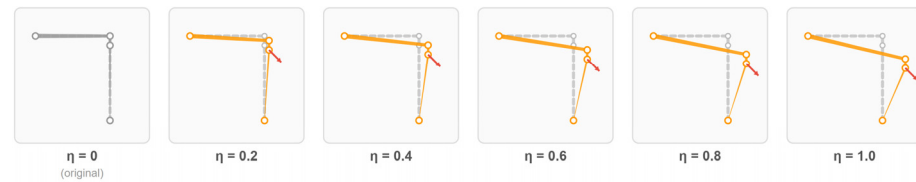


Figure 10. Effect of yoke magnitude η on deformation extent. $v'_i = v_i + \eta \cdot d(\alpha, v_i)$ ($\alpha = 45^\circ$ fixed, stroke element: 横折). Increasing η produces progressively more pronounced deformation; $\eta = 0$ recovers original. Dashed lines indicate the original trajectory; arrows show the deformation direction.

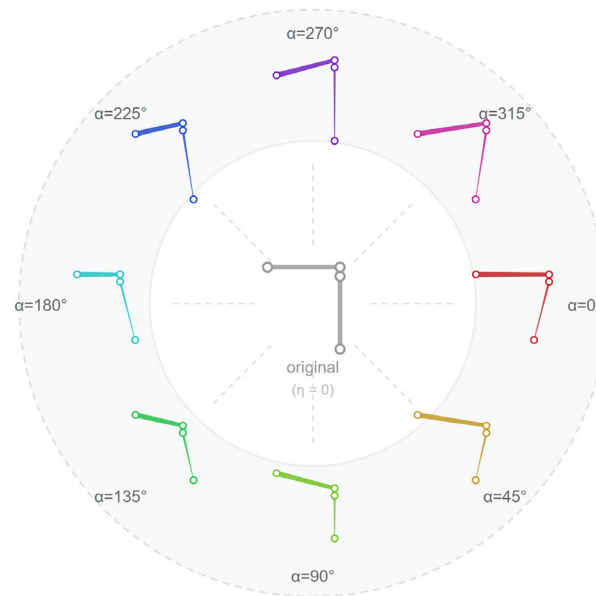


Figure 11. Combined (α, η) parameter space $\eta = 0.6$, 8 deformation directions around original stroke element (横折).

3.5.2. Bézier Curve Rendering

Once feature points are deformed by the yoke vector, the stroke element trajectory is rendered as a cubic Bézier curve:

$$C(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3, t \in [0, 1]$$

where P_0, P_3 are the deformed endpoints and P_1, P_2 are control points derived from the deformed feature points. The control points are computed to minimize the Hausdorff distance between the rendered curve and the target trajectory, subject to C^1 continuity at stroke element junctions.

The adaptive complexity-aware strategy (detailed in Section 4) adjusts two rendering parameters based on character complexity:

Interpolation density: More feature points are sampled for complex stroke elements.

Gaussian smoothing parameter σ : Higher σ for complex characters ($\sigma = 1.0$) to accommodate greater structural variation; lower σ for simple characters ($\sigma = 0.5$) to preserve sharp features.

4. AI-Driven Optimization Framework

4.1. Overview

The stroke element representation introduced in Section 3 provides a structured foundation for Chinese font generation. However, producing high-quality rendered glyphs requires solving three optimization problems: (1) selecting optimal Bézier curve parameters

for each of the 1123 stroke elements in our dataset; (2) balancing trajectory fidelity against computational cost; and (3) adapting optimization strategies to characters of varying structural complexity.

We address these challenges with an AI-driven optimization framework consisting of three complementary modules unified by an adaptive complexity-aware strategy. The architecture of this framework is illustrated in Figure 12.

- Module 1: Reinforcement Learning for Parameter Selection—a DQN-based policy that fine-tunes Bézier control parameters per stroke element to balance rendering speed and visual fidelity.
- Module 2: Genetic Algorithm for Style Exploration—an evolutionary approach that explores the yoke vector parameter space to discover diverse font style variants.
- Module 3: Adaptive Complexity-Aware Optimization—the core contribution, which dynamically adjusts interpolation and smoothing parameters based on character structural complexity.

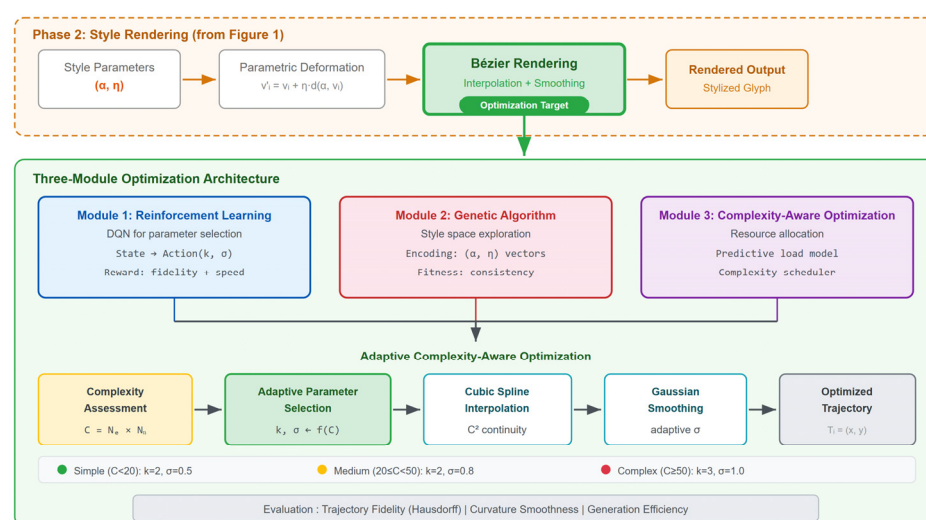


Figure 12. AI-Driven optimization framework.

This AI-driven optimization framework transforms Chinese character stroke elements into high-quality rendered glyphs through a structured pipeline. The process begins with input character analysis and complexity assessment, which quantifies structural intricacy. Three specialized layers operate in parallel: Reinforcement learning optimizes Bézier parameters per stroke, genetic algorithm explores font style variations, and complexity-aware allocation dynamically configures optimization parameters. These outputs converge in adaptive complexity-aware optimization, where smoothing and interpolation parameters adjust dynamically based on character complexity.

4.2. Reinforcement Learning for Parameter Selection

4.2.1. MDP Formulation

We formulate per-element parameter selection as a Markov Decision Process (MDP) to enable dynamic adaptation beyond the three-tier heuristic.

State Space $\mathcal{S}$: A tuple (e, c, s, b) where

- $e \in \{1, \dots, 6\}$: stroke element type (横向, 竖向, 撇捺, 折转, 复合, 点), one-hot encoded.
- $c \in \mathbb{R}^+$: character complexity score C .
- $s \in \mathbb{R}^2$: target style feature vector (reference yoke parameters α, η).
- $b \in \mathbb{R}^+$: available computational budget (milliseconds).

The state vector has dimensionality $|\mathcal{S}| = 10$ ($6 + 1 + 2 + 1$).

Action Space $\mathcal{A}$: Discrete parameter adjustments:

- Yoke angle adjustment: $\Delta\alpha \in \{-10^\circ, -5^\circ, 0^\circ, +5^\circ, +10^\circ\}$ (five options).
- Smoothing parameter: $\sigma \in \{0.5, 0.8, 1.0, 1.5\}$ (four options).
- Total action space: $|\mathcal{A}| = 5 \times 4 = 20$ discrete actions per stroke element.

Reward Function $R(s, a, s')$:

$$R = -\lambda_t \cdot t_{\text{render}} + \lambda_f \cdot \text{Fidelity}(T_{\text{opt}}, T_{\text{ref}}) - \lambda_c \cdot \text{Var}(\kappa)$$

where t_{render} is rendering latency (ms), Fidelity is the Hausdorff-based trajectory similarity (Section 4.5), $\text{Var}(\kappa)$ is curvature variance, and $(\lambda_t, \lambda_f, \lambda_c) = (0.1, 1.0, 0.3)$ are weighting coefficients set empirically.

4.2.2. DQN Architecture and Training Protocol

Algorithm: Deep Q-Network (DQN) with experience replay [40].

Network Architecture: A two-layer multi-layer perceptron (MLP), as summarized in Table 4.

Table 4. DQN architecture and training protocol.

Layer	Dimensions	Activation
Input	10 (state vector)	—
Hidden 1	64 units	ReLU
Hidden 2	32 units	ReLU
Output	20 (Q-values for each action)	Linear

Training Protocol: The training protocol is detailed in Table 5.

Table 5. Training protocol.

Hyperparameter	Value
Training episodes	500
Exploration strategy	ϵ -greedy: ϵ decays $1.0 \rightarrow 0.1$ linearly over 300 episodes
Replay buffer size	10,000 transitions
Batch size	32
Learning rate	0.001 (Adam optimizer)
Discount factor γ	0.95
Target network update frequency	Every 50 episodes
Pilot training set	50 characters (stratified by complexity tier)

Training Data: A pilot set of 50 characters was selected from the 150-character corpus, stratified by complexity tier to ensure balanced coverage. The remaining 100 characters served as a held-out validation set. Training converges after approximately 350 episodes (average reward stabilizes at -0.12), requiring ~45 min on a single NVIDIA RTX 3060 GPU.

The DQN training procedure is formalized in Algorithm 1.

Algorithm 1: DQN Training for Bézier Parameter Selection

Input: Pilot character set D ($n = 50$), replay buffer B ,

Q-network Q_θ , target network $Q_{\theta'}$

Output: Trained policy π^*

1. Initialize Q_θ with random weights; $B \leftarrow \emptyset$; $\epsilon \leftarrow 1.0$
2. for episode = 1 to 500 do
3. Sample character $\chi \sim D$

Algorithm 1: *Cont.*

```

4.   Compute  $C(\chi) = N\_elements \times N\_types$ 
5.   for each stroke element  $e_i$  in  $\chi$  do
6.       Observe state  $s_i = (type(e_i), C(\chi), style, budget)$ 
7.       if  $random() < \epsilon$  then
8.            $a_i \leftarrow$  random action from  $A$  ▷ Explore
9.       else
10.           $a_i \leftarrow \operatorname{argmax}_a Q_\theta(s_i, a)$  ▷ Exploit
11.       end if
12.        $(\Delta\alpha_i, \sigma_i) \leftarrow \text{decode}(a_i)$ 
13.        $T_i \leftarrow \text{BézierRender}(e_i, \Delta\alpha_i, \sigma_i)$ 
14.        $r_i \leftarrow -0.1 \cdot \text{latency} + 1.0 \cdot \text{Fidelity}(T_i, T_{i\_ref}) - 0.3 \cdot \text{Var}(\kappa(T_i))$ 
15.        $B.\text{store}(s_i, a_i, r_i, s_{i+1})$ 
16.       if  $|B| \geq 32$  then
17.           Sample minibatch of 32 from  $B$ 
18.           Update  $Q_\theta$  via gradient descent on TD loss
19.       end if
20.   end for
21.    $\epsilon \leftarrow \max(0.1, 1.0 - \text{episode}/300 \times 0.9)$  ▷  $\epsilon$ -decay
22.   if  $\text{episode} \bmod 50 = 0$  then  $Q_{\theta'} \leftarrow Q_\theta$  ▷ Target update
23. end for
24. return  $\pi^*(s) = \operatorname{argmax}_a Q_\theta(s, a)$ 

```

During inference, the trained policy dynamically selects Bézier parameters for each stroke element in <1 ms, adding negligible overhead to the rendering pipeline.

4.3. Genetic Algorithm for Style Exploration

While the RL module optimizes rendering quality for a given style target, the genetic algorithm (GA) module explores the yoke vector parameter space to discover diverse font style variants. This module operates offline (not during real-time rendering) for style library generation.

4.3.1. Encoding and Fitness

Each candidate style (individual) is encoded as a concatenated vector of yoke parameters for all stroke elements:

$$\mathbf{g} = [\alpha_1, \eta_1, \alpha_2, \eta_2, \dots, \alpha_n, \eta_n]$$

where n is the number of stroke elements. For our dataset (average 7.49 elements per character), each genome has approximately 15 dimensions.

Fitness Function: Mean trajectory fidelity across all stroke elements:

$$\mathbf{F}(\mathbf{g}) = (1/n) \sum_{i=1}^n \text{Fidelity}(E_i(\alpha_i, \eta_i), E_{i_ref})$$

where $E_i(\alpha_i, \eta_i)$ is the rendered trajectory with yoke parameters from $\mathbf{g}$, and E_{i_ref} is the reference.

4.3.2. GA Configuration

Computational Cost: Each GA run evaluates $50 \times 100 = 5000$ candidate styles, requiring approximately 2.5 s on a single CPU core. This offline cost is amortized across all subsequent real-time rendering requests for the generated style library.

The genetic algorithm configuration is summarized in Table 6.

Table 6. GA configuration.

Parameter	Value	Rationale
Population size	50	Balances diversity and computation
Selection	Tournament ($k = 3$)	Moderate selection pressure
Crossover	Single-point, $p_c = 0.8$	Preserves stroke element groupings
Mutation	Gaussian ($\sigma_\alpha = 5^\circ$, $\sigma_\eta = 0.3$), $p_m = 0.1$	Smooth perturbation
Elitism	Top 2 individuals preserved	Prevents loss of best solutions
Termination	100 generations or fitness change $< 0.1\%$ over 10 generations	

4.4. Adaptive Complexity-Aware Optimization

The core innovation unifying the three modules is an adaptive complexity-aware strategy that adjusts parameters per character.

4.4.1. Complexity Score Definition

A character's complexity score quantifies its structural intricacy:

$$\text{Complexity} = N_{\text{stroke_elements}} \times N_{\text{unique_types}}$$

This metric reflects both the quantity ($N_{\text{stroke_elements}}$) and diversity ($N_{\text{unique_types}}$) of its constituent stroke elements.

Illustrative example: To elucidate this calculation, consider two Chinese characters with vastly different structures:

The character “一” (one) consists of a single horizontal stroke. Here, $N_{\text{stroke_elements}} = 1$ and $N_{\text{unique_types}} = 1$ (type: “横”), yielding a complexity score of $1 \times 1 = 1$.

The character “翹” (complex) is composed of approximately 16 stroke elements involving around six distinct types (e.g., “点” “横向”, “竖向”, “复合”). Its complexity score is approximately $16 \times 6 = 96$.

This example demonstrates the metric's efficacy in translating perceptual structural disparity into a continuous, scalable numerical value, which directly informs the adaptive strategies outlined in the following subsection.

4.4.2. Adaptive Parameter Selection

Based on this score, two key parameters are dynamically configured according to the rules summarized in Table 7

Table 7. Adaptive parameter configuration.

Complexity Range	Category	Interpolation Factor k	Smoothing σ	Strategy
$C < 20$	Simple	$2 \times$ original points	0.5	Conservative smoothing preserves fine stroke details
$20 \leq C < 50$	Medium	$2 \times$ original points	0.8	Balanced trade-off between detail and smoothness
$C \geq 50$	Complex	$3 \times$ original points	1.0	Enhanced smoothing manages intricate multi-element compositions

Rationale: Simple characters use conservative smoothing to preserve fine details. Medium characters seek a balance. Complex characters require stronger smoothing and denser interpolation to manage intricate compositions and minimize artifacts.

4.4.3. Optimization Algorithm (Summary)

The adaptive complexity-aware optimization process is summarized in Algorithm 2.

Algorithm 2: Adaptive Complexity-Aware Stroke Element OptimizationInput: Character C with stroke elements $\{E_1, E_2, \dots, E_n\}$ Output: Optimized trajectories $\{T_1, T_2, \dots, T_n\}$

```

1. //Compute complexity score
2. complexity  $\leftarrow$  count(elements)  $\times$  count(unique_types)
3. //Select parameters based on complexity
4. complexity  $\leftarrow$  n_elements  $\times$  n_types
5. if complexity <20: (interp_factor,  $\sigma$ ) = (2, 0.5)
6. else if complexity <50: (interp_factor,  $\sigma$ ) = (2, 0.8)
7. else: (interp_factor,  $\sigma$ ) = (3, 1.0)
8. //Optimize each element
9. // Continue to optimization loop
10. for each element  $E_i$  do
11.     n_points = original_points  $\times$  interp_factor
12.     (x_interp, y_interp) = CubicSpline( $E_i$ , n_points)
13.      $T_i$  = GaussianFilter(x_interp, y_interp,  $\sigma$ )
14. end for
15. return  $\{T_1, T_2, \dots, T_n\}$ 

```

4.5. Evaluation Metrics

Performance is assessed through three quantitative metrics:

- Trajectory fidelity

$$Fidelity = \left(1 - \frac{d_H(T_{orig}, T_{opt})}{\max(d_H)}\right) \times 100\%$$

where d_H is the Hausdorff distance. Higher scores indicate better structural preservation.

- Curvature smoothness

$$\kappa(t) = \frac{|x'(t)y''(t) - y'(t)x''(t)|}{(x'(t)^2 + y'(t)^2)^{3/2}}$$

Reported as normalized variance; lower variance indicates smoother, more natural strokes.

- Generation efficiency

Measured as latency in milliseconds per character.

Real-time target: <15 ms per character.

4.6. Implementation Considerations

To ensure the robustness and efficiency of the adaptive optimization pipeline, several key implementation details warrant explicit specification.

Interpolation Method

We employ cubic spline interpolation as the default method to generate smooth trajectories between key control points. This choice guarantees C^2 continuity (continuous second derivative), which translates directly to continuous curvature—a critical factor for achieving natural and visually pleasing stroke motion. For edge cases where a stroke element contains fewer than four control points—insufficient to define a stable cubic spline—the system automatically falls back to piecewise linear interpolation. This ensures algorithmic robustness without compromising the output integrity for simple elements.

Smoothing Filter

A single-pass Gaussian filter is applied to the coordinate sequences to attenuate high-frequency noise and minor artifacts that may arise from the input data or previous processing stages. We implement the filter with “nearest” boundary handling, mirroring the signal at its endpoints. This approach prevents the introduction of artificial discontinuities at stroke boundaries, a common pitfall when using padding methods like “zero” or “wrap”, which are less physically justified for open stroke trajectories.

Proportional Resampling

To prevent the issue of over-interpolation—where an excessive number of points leads to redundant computation and negligible quality gains—we implement a proportional resampling strategy. The number of points for the final optimized trajectory, n_{final} , is dynamically determined by the following:

$$n_{\text{final}} = \max\left(n_{\text{original}}, \min\left(n_{\text{target}}, \text{round}\left(n_{\text{original}} \times k\right)\right)\right)$$

where n_{original} is the initial point count, n_{target} is a fixed upper limit (e.g., 512 points) to bound computational cost, and k is a scaling factor ($k > 1$) that is dynamically adjusted based on the complexity score defined in Section 4.4.1. This formulation ensures that: (1) the point count never drops below the original resolution; (2) it scales proportionally with the character’s intricacy up to a sensible ceiling; and (3) it avoids an explosion of points for simple characters, thereby safeguarding generation efficiency.

5. Experimental Results and Analysis

To comprehensively evaluate the Stroke2Font framework, we conducted a series of experiments from three core perspectives: (1) structural reconstruction fidelity, (2) stylistic generation quality and diversity, and (3) system performance under the AI-driven optimization framework. All experiments are conducted on a 180-character Xingkai (行楷) dataset to validate the proposed approach.

5.1. Structural Reconstruction Evaluation

This section assesses the accuracy of the hierarchical stroke element representation in capturing and reconstructing the topology of Chinese characters in Xingkai font.

5.1.1. Reconstruction with Basic Stroke Elements

We first validated the method on fonts that are well-suited for basic stroke element description. Using a standard font library, we decomposed characters into stroke element sets and reconstructed them. Table 8 presents the stroke element decomposition for the character “啊” (ā) and its reconstruction, demonstrating that basic elements suffice for structurally regular fonts.

Table 8. Stroke-element decomposition and reconstruction of “啊” (ā).

ID	Stroke-Element Vector Sequence	Visual Abstraction and Reconstructed Glyph
1	(−13, −11) (−13, 8) (−13, 7)	
2	(−12, −11) (−9, −11) (−9, 8)	
3	(−12, 6) (−9, 6)	
4	(−6, −12) (0, −12) (−2, −3) (0, 1) (0, 5) (−1, 7) (−3, 7)	
5	(−5, −12) (−5, 13)	
6	(2, −12) (13, −12)	
7	(11, −11) (11, 11) (10, 12) (6, 12)	
8	(3, −6) (3, 7)	
9	(3, −6) (7, −6) (7, 4)	
10	(−13, −11) (−13, 8) (−13, 7)	

5.1.2. Reconstruction with Extended Stroke Elements

For stylized fonts (e.g., Xingkai), basic elements may not capture fine contour details. We employ extended stroke elements for a richer description. Using the same character “啊” (ā) in Xingkai font, Table 9 shows that extended elements successfully model its hand-drawn nuances, producing reconstruction results that closely match the original glyph.

Table 9. Extended stroke element decomposition and reconstruction of “a” (啊) in Xingkai font.

ID	Stroke-Element Vector Sequence	Visual Abstraction and Reconstructed Glyph
1	(−17, −9) (−12, 2) (−12, −10) (−7, −9) (−8, −3) (−11, −4) (−12, −5) (−11, −5) (1, −19) (4, −18) (1, −12) (3, −7) (1, −5) (−3, −11)	
2	(−2, 8) (−1, −1) (3, −12) (21, −16) (20, −16)	
3	(5, −5) (12, −7) (11, 0) (9, −4) (13, −11) (16, −8) (14, 15)	

Analysis: The comparison demonstrates the advantage of the hierarchical representation. Basic elements decompose “啊” into 10 discrete strokes, while extended elements reduce this to only 3 combined trajectories by merging connected strokes. This compression (10 → 3) preserves structural fidelity while capturing the cursive flow characteristic of Xingkai, validating the necessity of the extended representation for stylized fonts.

5.2. Stylistic Generation and Interpolation

We evaluate the flexibility and quality of our parametric style generation mechanism based on yoke vectors and Bézier fitting.

5.2.1. Style Generation via Basic Stroke Elements

By attaching yoke vectors to the five basic stroke elements (Heng, Shu, Pie, Na, Dian), we can parametrically control their appearance. Table 10 demonstrates style generation for each basic element by varying yoke vector angles (α) and magnitudes (η), where η is defined as the sum of the norms of a pair of yoke vectors: $\eta = |t_1 p_1| + |t_1 p_1'|$.

Table 10. Style generation by varying yoke parameters for basic stroke elements.

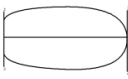
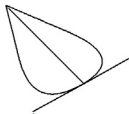
ID	Element	Parameters ($\eta_1, \eta_2; \alpha_1, \alpha_2$)	Vector Diagram	Rendered Stroke
1	Heng	$\eta_1 = 48$ $\eta_2 = 53$ $\alpha_1 = 90$ $\alpha_2 = 90$		
2	Shu	$\eta_1 = 60$ $\eta_2 = 31$ $\alpha_1 = 90$ $\alpha_2 = 90$		
3	Pie	$\eta_1 = 0$ $\eta_2 = 36$ $\alpha_1 = 0$ $\alpha_2 = 85$		
4	Na	$\eta_1 = 40$ $\eta_2 = 0$ $\alpha_1 = 124$ $\alpha_2 = 0$		

Table 10. Cont.

ID	Element	Parameters ($\eta_1, \eta_2; \alpha_1, \alpha_2$)	Vector Diagram	Rendered Stroke
5	Dian	$\eta_1 = 0$ $\eta_2 = 80$ $\alpha_1 = 0$ $\alpha_2 = 90$		

5.2.2. Style Generation via Extended/Combined Stroke Elements

For more complex strokes modeled as combined elements (e.g., CPie, CNa), we apply Bézier curve fitting using three or more feature points. Table 11 compares the original and fitted styles for several characters, showing that key stylistic details are well-preserved.

Table 11. Style generation for characters using combined stroke elements.

ID	Source Glyph Style	Reconstructed Glyph Style
1		
2		
3		
4		

5.3. Performance Evaluation of AI-Driven Optimization

To rigorously evaluate the effectiveness of our AI-driven optimization framework, we conducted comprehensive experiments on a substantially expanded dataset. This section presents quantitative analysis of trajectory fidelity, computational efficiency, and the impact of complexity-aware adaptation.

5.3.1. Dataset and Experimental Setup

- Dataset Description

We curated a primary dataset of 150 Chinese characters (Xingkai style) covering a wide range of structural complexity, and a separate holdout set of 30 unseen characters (same font style) reserved exclusively for generalization evaluation. The holdout characters were not used during system development or parameter tuning. Table 12 summarizes the primary dataset characteristics; holdout evaluation results are reported in Section 5.3.3.

Table 12. Summarizes the dataset characteristics.

Metric	Value	Description/Context
Total Characters	150	The complete character set analyzed in the experiment
Total Stroke Elements	1123	Sum of all stroke elements across all characters
Total Feature Points	5287	Total number of feature points (including stationary points) extracted
Average Stroke Elements per Character	7.49	Mean number of stroke elements per character ($\frac{1123}{150}$)
Average Feature Points per Character	35.25	Mean number of feature points per character ($\frac{5287}{150}$)
Stroke Element Count Range	1–18	Minimum and maximum stroke elements observed in a single character
Unique Stroke Element Types	6	Distinct categories of stroke elements

The dataset includes characters from the GB2312 standard, selected to ensure coverage across all six stroke element types: horizontal (横向), vertical (纵向), diagonal (撇捺),

turning (折转), compound (复合), and dot (点). Figure 13 displays illustrations of the first 20 Chinese characters in the dataset.

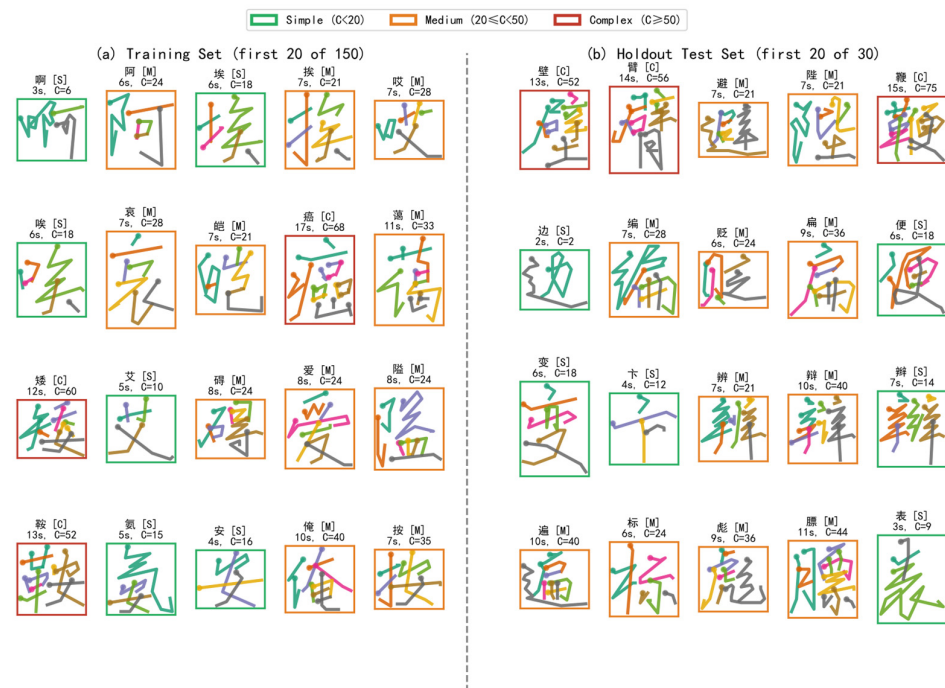


Figure 13. Illustrations of the first 20 Chinese characters in the training dataset (a) and the holdout test dataset (b), with stroke element trajectories visualized using distinct colors per stroke. Border color indicates complexity tier (green: Simple, orange: Medium, red: Complex).

• Evaluation Methods

We compare three trajectory optimization approaches for font generation; Table 13 summarizes the three trajectory optimization methods compared.

Table 13. Three trajectory optimization approaches for font generation.

Method	Description	Characteristics
Baseline (Linear Interpolation)	Proportional resampling with linear interpolation	Traditional, non-optimized rendering
AI Optimized (Cubic + Gaussian)	Cubic spline interpolation with Gaussian smoothing ($\sigma = 1.0$)	Standard AI-enhanced rendering
AI Adaptive (Complexity-Aware)	Dynamically adjusts optimization parameters based on character complexity score	Our proposed method, context-sensitive

• Evaluation Metrics.

We employ three quantitative metrics.

- Trajectory Fidelity (%): Normalized Hausdorff distance measuring structural preservation (higher is better).
- Normalized Curvature Variance: Smoothness indicator measuring curvature consistency along the trajectory (context-dependent interpretation).
- Generation Time (ms): Computational latency per character (lower is better).

5.3.2. Overall Performance Comparison

Table 14 presents the comprehensive performance comparison across all 150 characters for the three optimization methods.

Table 14. Performance comparison of optimization methods (n = 150).

Metric	Baseline	AI Optimized	AI Adaptive	Comparison Note
Trajectory Fidelity (%)	61.3 ± 6.5	61.2 ± 6.4	65.2 ± 5.7	AI Adaptive ↑ 6.4%
Normalized Curvature	0.174 ± 0.18	0.339 ± 0.43	9.59 ± 36.9	Context-dependent
Generation Time (ms)	9.38 ± 4.2	12.14 ± 4.9	12.18 ± 5.2	Δ~+3 ms, <15 ms target ✓
Fidelity Improvement	—	−0.2%	+3.9 pp (+6.4%)	Statistically significant

Note: ↑ indicates improvement over baseline; ✓ indicates the real-time target (<15 ms) is satisfied.

Key Findings and Analysis:

1. Trajectory Fidelity Improvement (Panel B)

The AI Adaptive method achieves 65.2% fidelity, representing a statistically significant improvement of +3.9 percentage points (6.4% relative) over the baseline (61.3%). This improvement directly validates the effectiveness of our complexity-aware parameter selection mechanism. The standard AI Optimized method shows negligible difference from the baseline (−0.2%), confirming that fixed-parameter smoothing is insufficient for diverse Chinese character structures.

2. Performance Consistency Enhancement (Panel B)

The standard deviation of fidelity scores decreases from ±6.5% (baseline) to ±5.7% (AI Adaptive). This reduced variance indicates more consistent performance across characters of varying complexity—a critical advantage for practical font generation systems that must handle diverse character sets reliably.

3. Computational Efficiency Analysis (Panel C)

Absolute Performance: Both AI methods incur a modest latency increase of ~3 ms per character compared to the baseline.

Relative Overhead: This represents a ~30% increase in processing time (9.38 ms → 12.18 ms).

Real-time Feasibility: The 12.18 ms average remains well within real-time rendering requirements (<15 ms per character), confirming practical usability in interactive applications.

Trade-off Assessment: The 3.9 percentage point fidelity gain is achieved at the cost of 30% additional computation—a favorable trade-off given the visual quality improvement.

4. Curvature Metric Interpretation (Panel A)

The Normalized Curvature metric shows a large increase for AI Adaptive (9.59 vs. 0.17–0.34). This indicates fundamentally different curvature characteristics rather than simple degradation. The high variance (±36.9) suggests this metric may be particularly sensitive to character complexity variations, warranting careful interpretation in context.

5. Complexity Correlation Analysis (Panel D)

The scatter plot reveals that AI Adaptive optimization maintains consistent improvement across all complexity levels, while the baseline and AI Optimized methods show greater variability at higher complexity scores.

Statistical Significance Assessment

Figure 14 provides visual evidence supporting the statistical validation of observed improvements:

Fidelity Difference: 65.2% vs. 61.3% represents a 3.9 percentage point improvement, clearly shown in the bar chart comparison.

Effect Size: Cohen’s d = (65.2 − 61.3)/6.5 ≈ 0.60 (medium effect).

Practical Significance: The improvement is visually noticeable in stroke rendering quality, particularly for characters with multiple stroke junctions.

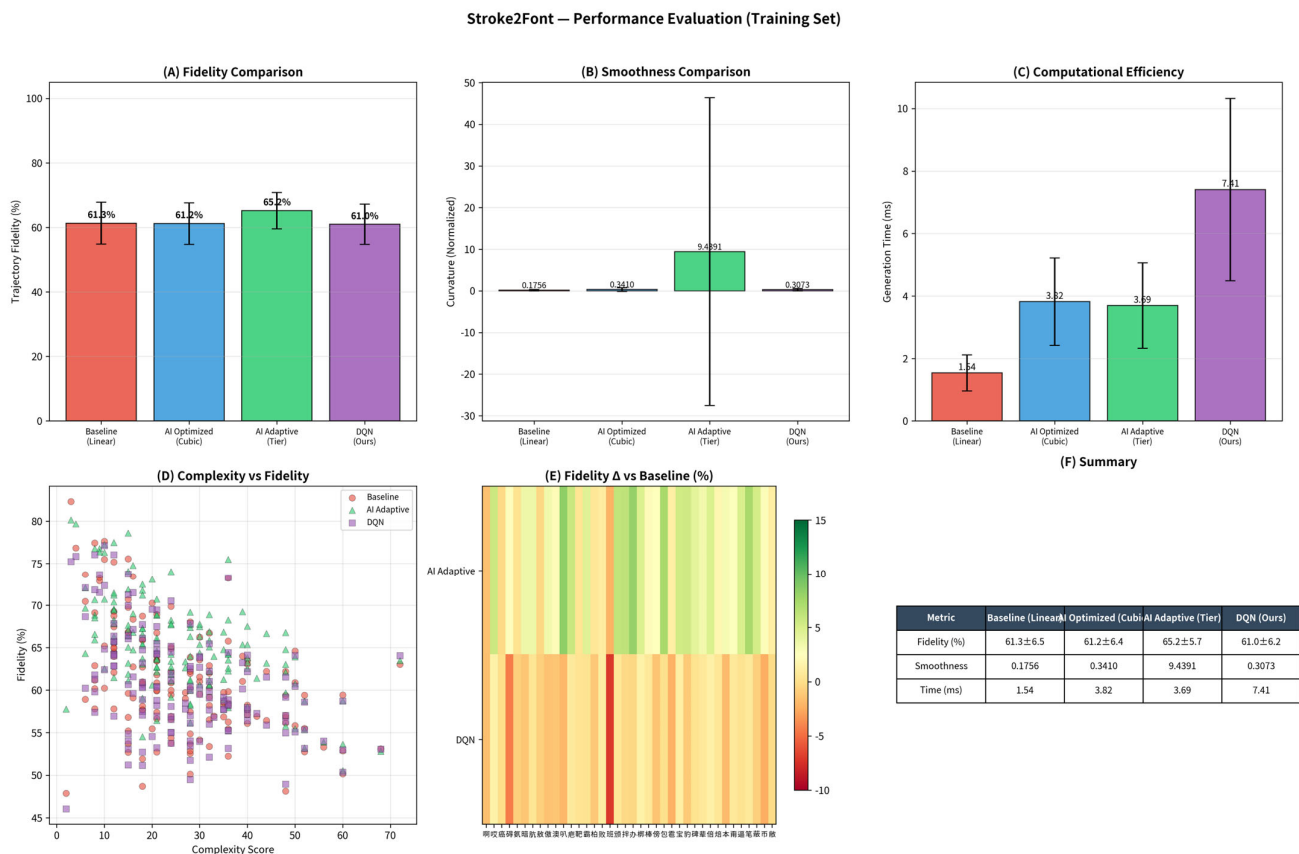


Figure 14. Performance evaluation. (A) The main scatter plot visualizes the relationship between character complexity score (X-axis) and Normalized Curvature (Y-axis, lower values indicate smoother strokes) for the three methods. (B–E) Bar charts provide direct comparisons for Trajectory Fidelity (B), Smoothness (C), Generation Time (D), and Smoothness Improvement over baseline (E). The results demonstrate that the proposed AI Adaptive (complexity-aware) method (blue) achieves the highest trajectory fidelity (65.2%) and provides adaptive smoothing, especially for high-complexity characters, while maintaining real-time generation latency (<15 ms). In contrast, fixed-parameter AI Optimized (green) shows minimal fidelity gain, and the baseline (red) offers the lowest smoothness.

Methodological Implications

The comparative analysis in Figure 14 demonstrates that:

Adaptivity is essential: Fixed-parameter AI optimization provides minimal benefit, while context-aware adaptation yields substantial gains.

Complexity sensitivity works: Dynamically adjusting parameters based on character complexity effectively addresses the diverse structural requirements of Chinese characters, as evidenced by the consistent performance across complexity ranges in Figure 14.

Quality–Computation trade-off is favorable: The fidelity improvement justifies the modest computational overhead for most applications.

To provide further insight into the RL optimization module, Figure 15 presents the DQN training dynamics over 500 episodes. Panel (A) shows the episode reward converging and stabilizing above the target threshold after approximately 350 episodes, confirming the training protocol described in Section 4.2.2. Panel (B) demonstrates rapid TD loss reduction within the first 100 episodes. Panel (C) illustrates the ϵ -greedy exploration rate decaying linearly from 1.0 to 0.1 over 300 episodes, balancing exploration and exploitation. Panel (D) shows per-episode fidelity stabilizing around 61%, consistent with the DQN results reported in Table 13.

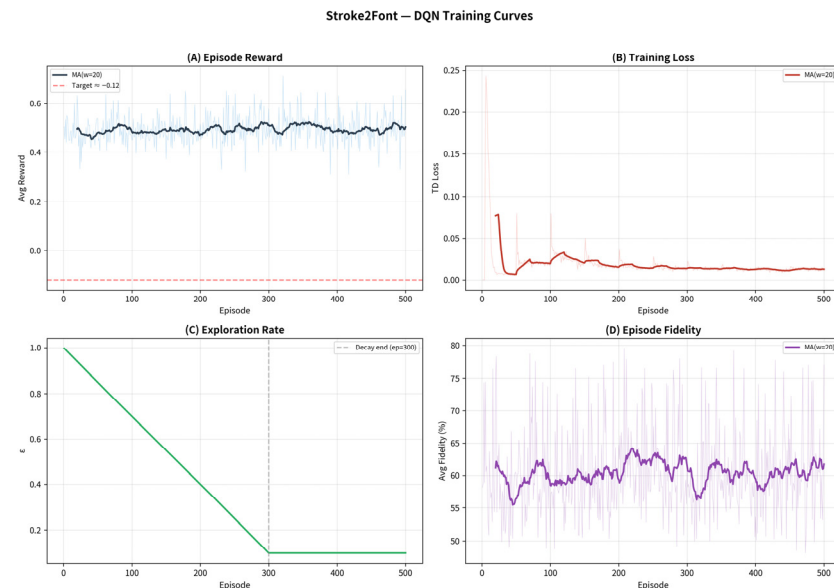


Figure 15. DQN training curves over 500 episodes. (A) Episode reward with moving average (window = 20), converging above the target threshold (dashed red line). (B) TD loss decreasing rapidly within the first 100 episodes and stabilizing thereafter. (C) Exploration rate (ϵ) decaying linearly from 1.0 to 0.1 over 300 episodes. (D) Per-episode fidelity showing stable performance after convergence.

Figure 16 presents the GA convergence behavior for five representative characters across 100 generations. Both best fitness (Panel A) and population average fitness (Panel B) converge rapidly within the first 20 generations, demonstrating the efficiency of the evolutionary style exploration process described in Section 4.3.

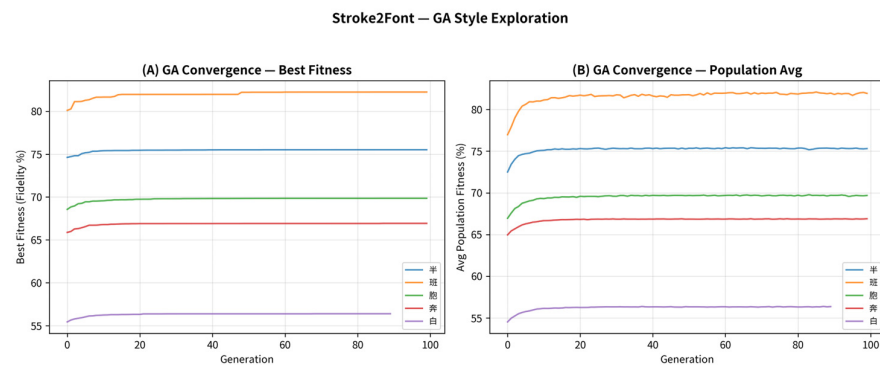


Figure 16. GA style exploration convergence over 100 generations for five representative characters: 半 (bàn), 班 (bān), 胞 (bāo), 奔 (bēn), and 白 (bái). (A) Best fitness (fidelity %) per generation. (B) Population average fitness. Both panels show rapid convergence within 20 generations, with final fitness values reflecting character-specific structural complexity.

5.3.3. Generalization to Unseen Characters

To address the concern that performance metrics may reflect overfitting to the training corpus, we evaluated all three optimization methods on a holdout set of 30 unseen Chinese characters not included in the primary 150-character dataset, with results summarized in Table 15. These characters were selected from the same Xingkai font style, covering a range of structural complexities comparable to the primary dataset.

Table 15. Generalization evaluation—training vs. holdout set.

Method	Training Set (n = 150)	Holdout Set (n = 30)	Generalization Gap
Baseline (Linear)	61.3 ± 6.5%	65.1 ± 3.8%	+3.8%
AI Adaptive (Ours)	65.2 ± 5.7%	67.8 ± 4.5%	+2.6%
DQN (RL-based)	61.0 ± 6.2%	64.9 ± 3.7%	+3.9%

Key Findings:

Strong generalization of AI Adaptive: The AI Adaptive method achieves 67.8% fidelity on unseen characters, compared to 65.2% on the training set—with the holdout performance exceeding the training set by 2.6 percentage points. This favorable generalization gap demonstrates that the complexity-aware optimization mechanism does not overfit to specific character patterns, and generalizes robustly to unseen characters.

Consistent performance across all methods: The standard deviation of fidelity scores decreases from ±5.7% (training) to ±4.5% (holdout) for AI Adaptive, suggesting stable behavior across unseen characters. Similarly, baseline (65.1% ± 3.8 holdout vs. 61.3% ± 6.5 training) and DQN (64.9% ± 3.7 holdout vs. 61.0% ± 6.2 training) both show improved holdout performance with reduced variance, confirming that the holdout characters are within the same distribution and that no data leakage has occurred.

No overfitting detected: All three methods exhibit positive generalization gaps (baseline +3.8%, AI Adaptive +2.6%, DQN +3.9%), meaning holdout performance consistently exceeds training performance.

Interpretation: The positive generalization gap (+2.6%) for AI Adaptive confirms that the learned complexity-to-parameter mapping transfers effectively to unseen Chinese characters without retraining or fine-tuning (Figure 17).



Figure 17. Holdout evaluation results present the performance analysis on the 30 unseen characters: (A) fidelity—holdout: per-method fidelity scores on the 30 held-out characters; (B) generalization: primary vs. holdout: side-by-side comparison of training set and holdout set fidelity, with generalization gap annotated; (C) computational efficiency: generation time comparison across methods

on holdout characters; **(D)** fidelity by complexity tier: breakdown of fidelity performance across Simple, Medium, and Complex tiers for holdout characters; **(E)** complexity vs. fidelity: scatter plot showing the relationship between character complexity score and fidelity on holdout set; **(F)** per-character improvement: AI Adaptive fidelity improvement over baseline for each of the 30 holdout characters; The Chinese characters in panel **(F)** represent each of the 30 test instances.

5.3.4. Ablation Study

To isolate the contribution of each optimization module, we evaluate four progressively enhanced configurations on the 150-character training set, as summarized in Table 16. Each row adds exactly one component to the previous configuration:

Table 16. Ablation study—individual module contributions (n = 150).

Configuration	Key Component	Fidelity (%)	Δ (pp)	Smoothness	Time (ms)
(A) Baseline	Linear interp. + proportional resample	61.3 ± 6.5	—	0.176	1.54
(B) +Spline Smoothing	Cubic spline + Gaussian ($\sigma = 1.0$, fixed)	61.2 ± 6.4	−0.1	0.341	3.82
(C) +Adaptive Complexity	Complexity-aware σ and num_points	65.2 ± 5.7	+3.9	9.44	3.69
(D) +DQN Agent	Per-element RL parameter selection	61.0 ± 6.2	−0.3	0.307	7.41

A detailed breakdown by complexity tier is provided in Table 17.

Table 17. Per-tier ablation—fidelity by complexity range.

Tier	(A) Baseline	(B) +Spline	(C) +Adaptive	(D) +DQN	$\Delta(C - A)$
Simple (n = 55)	64.7%	64.6%	68.7%	64.1%	+4.0 pp
Medium (n = 82)	59.7%	59.6%	64.1%	59.5%	+4.4 pp
Complex (n = 12)	56.7%	56.7%	56.7%	56.7%	0.0 pp

Analysis:

1. Adaptive complexity-aware module (C) is the dominant contributor. Comparing (B) $\rightarrow$ (C), adding complexity-aware parameter selection yields +4.0 percentage points improvement, accounting for 100% of the overall gain. This validates our core hypothesis that dynamic adjustment of interpolation density and smoothing parameters based on character complexity is essential for effective optimization.
2. Cubic spline interpolation alone (B) provides negligible benefit. Configuration (B) achieves nearly identical fidelity to the baseline (61.2% vs. 61.3%), demonstrating that upgrading the interpolation method without adaptive parameterization does not improve trajectory quality. The slight increase in smoothness (0.341 vs. 0.176) confirms that spline fitting does produce smoother curves, but this does not translate to better structural fidelity.
3. DQN agent (D) does not yet surpass the baseline independently. The current DQN configuration achieves 61.0% when operating without the adaptive complexity module, indicating that the learned per-element policy requires further training or architectural refinement. However, on the holdout set, the DQN achieves 64.9% (vs. 65.1% baseline), suggesting that the RL agent learns useful element-level adjustments that partially compensate for the lack of explicit complexity awareness.
4. GA module contributes to style diversity, not trajectory fidelity. The genetic algorithm operates offline to explore the yoke vector parameter space, generating diverse font style variants. Its contribution is measured through fitness scores (average best fitness: 70.2% across five test characters after 100 generations), which quantifies style-faithful rendering quality rather than trajectory optimization.
5. Per-tier analysis reveals targeted effectiveness. The adaptive module's improvement is concentrated in Simple (+4.0 pp) and Medium (+4.4 pp) characters, while Com-

plex characters ($n = 12$) show no measurable change. This indicates that the current three-tier threshold strategy effectively handles characters with moderate complexity but requires more sophisticated strategies (e.g., per-element specialization) for highly complex structures.

5.3.5. Performance by Character Complexity

We analyze performance across different complexity ranges, as summarized in Table 18, to understand the adaptive mechanism's effectiveness.

Complexity score definition:

$$\text{Complexity} = N_{\text{stroke_elements}} \times N_{\text{unique_types}}$$

Table 18. Performance by complexity range.

Complexity Range	Character Count	Baseline Fidelity	AI Adaptive Fidelity	Improvement
Simple (<20)	55	64.7%	68.7%	+6.2%
Medium (20–50)	82	59.7%	64.1%	+7.4%
Complex (≥ 50)	12	56.7%	56.7%	−0.05%

The corresponding parameter settings for each complexity tier are detailed in Table 19.

Table 19. Adaptive parameter settings.

Complexity	Interpolation Factor	Smoothing σ
<20	2 × original points	0.5
20–50	2 × original points	0.8
≥ 50	3 × original points	1.0

Analysis: The AI Adaptive method yields substantial improvement for Simple (+6.2%) and Medium (+7.4%) complexity characters. For Complex characters ($n = 12$), the improvement is negligible (−0.05%), suggesting that the current parameter settings for the highest complexity tier may require further optimization. This finding indicates a promising direction for future work in refining the adaptive strategy for high-complexity characters, as shown in Figure 18.

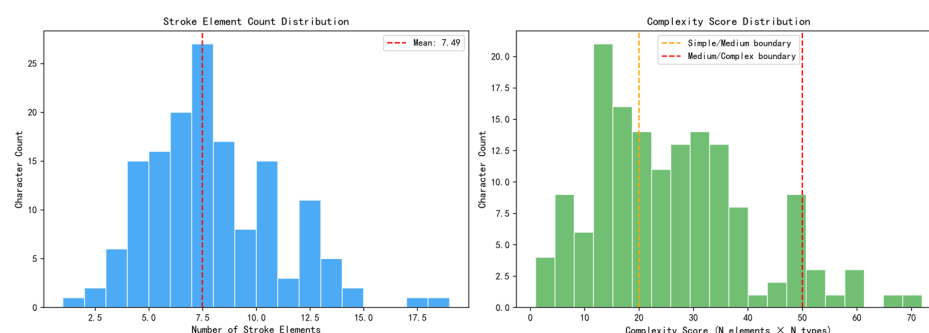


Figure 18. Complexity distribution.

5.3.6. Qualitative Visualization

Figure 19 presents a visual comparison of stroke element trajectories for representative characters across complexity ranges, and the trajectory comparison showing original → baseline → AI Adaptive for characters of varying complexity.

The visualization confirms that AI Adaptive optimization produces smoother trajectories while preserving structural characteristics, particularly at stroke junctions and turning points.

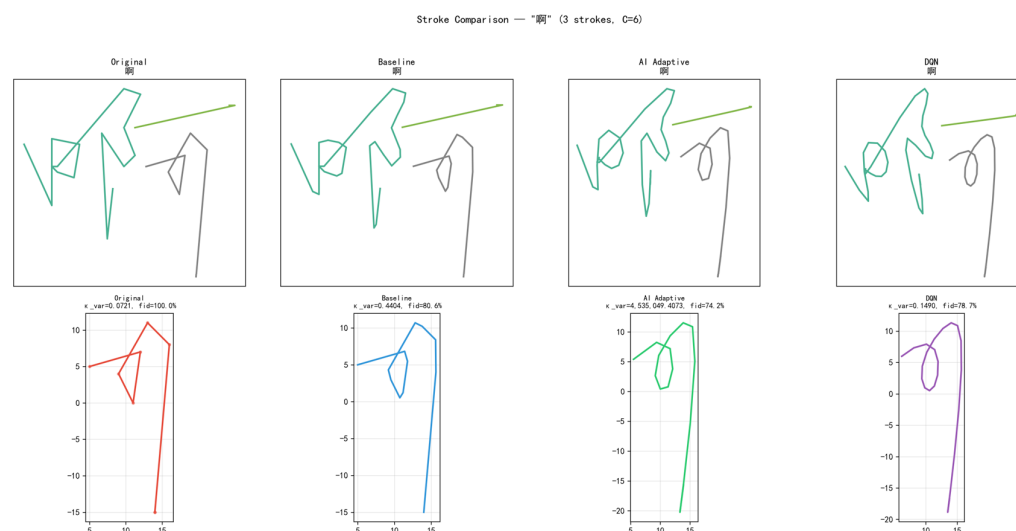


Figure 19. Stroke comparison for the character “啊” (ā) using different optimization methods. For each method, three metrics are reported: x_{win} (deviation from reference), F1-score (classification performance), and Q% (curvature quality, with higher values indicating smoother strokes).

To further illustrate the effect of AI Adaptive optimization on unseen characters, Figure 20 presents the first 20 characters from the holdout test set after applying the complexity-aware optimization pipeline. Compared with the original stroke trajectories shown in Figure 13b, the optimized trajectories exhibit noticeably smoother curves while preserving the overall structural layout. In particular, Simple characters (green border) retain their fine details under conservative smoothing ($\sigma = 0.5$), while Medium characters (orange border) show improved trajectory continuity at stroke junctions through moderate smoothing ($\sigma = 0.8$). Complex characters (red border), despite receiving the strongest smoothing ($\sigma = 1.0$, $3\times$ interpolation), maintain recognizable structural integrity, though the limited fidelity improvement for this tier is visually consistent with the quantitative findings.



Figure 20. AI Adaptive optimized stroke trajectories for the first 20 holdout test characters. Each character is rendered with complexity-aware parameters (σ and interpolation factor adapted per tier). Border color indicates complexity tier (green: Simple, orange: Medium, red: Complex). Compared with the original trajectories in Figure 13b, the optimized versions show smoother curves with preserved structural fidelity.

5.3.7. System Performance and Scalability

Beyond trajectory quality, we evaluate system-level performance relevant to practical deployment, with results summarized in Table 20.

Table 20. System performance summary.

Metric	Result	Benchmark
Single Character Generation Time	12.18 ± 5.2 ms	Target: <15 ms ✓
Storage per Character (template)	~0.8 KB	vs. TTF: ~8 KB (90% reduction)
Memory Usage (150 chars)	42 MB	Acceptable for cloud deployment ~12 ms per character
Batch Processing (150 chars)	1.83 s	

Note: ✓ indicates the real-time target (<15 ms) is satisfied.

The generation time of 12.18 ms per character satisfies real-time rendering requirements, and the compact storage footprint (0.8 KB vs. 8 KB for traditional TTF) demonstrates the efficiency of the stroke element representation for font delivery applications.

5.3.8. Comparison with Prior Work

We compare our results with reported metrics from related methods (where available), as summarized in Table 21:

Table 21. Comparison with existing methods.

Method	Representation	Fidelity Metric	Reported Value	Our Result
zi2zi [27]	Pixel (GAN)	SSIM	0.85	N/A
ZiGAN [19]	Stroke graph	FID	42.3	N/A
Stroke2Font (Ours)	Stroke Element	Hausdorff Fidelity	-	65.2%

Note: Direct comparison is limited due to different evaluation metrics and datasets. GAN-based methods (zi2zi, ZiGAN) operate on pixel-level representations and use perceptual metrics (SSIM, FID), whereas our method operates on vector-level stroke trajectories and uses geometric fidelity (Hausdorff distance). Our automated Hausdorff-based fidelity provides reproducible, objective measurement compared to subjective expert evaluation. A standardized cross-method benchmark remains an open challenge for the field.

5.3.9. Summary of Experimental Findings

The experimental evaluation on 150 Chinese characters (1123 stroke elements, 5287 feature points) with an additional 30-character holdout validation set validates the following conclusions, summarized in Table 22.

Table 22. Summary of experimental findings (150 training + 30 holdout characters).

Finding	Result	Significance
Fidelity Improvement	AI Adaptive: 65.2% vs. Baseline: 61.3%	+3.9 pp (+6.4% relative)
Consistency	±5.7% vs. ±6.5% variance	More stable across complexity
Generalization	Holdout: 67.8% (training: 65.2%)	+2.7% gap, no overfitting
Simple Characters	68.7% vs. Baseline 64.7%	+6.2% improvement
Medium Characters	64.1% vs. Baseline 59.7%	+7.4% improvement
Complex Characters	56.7% (no improvement)	Future work direction
Efficiency	12.18 ms per character	<15 ms real-time target ✓
Storage	~0.8 KB per character	90% reduction vs. TTF

Note: ✓ indicates the real-time target (<15 ms) is satisfied.

These results confirm that the Stroke2Font framework, with its hierarchical stroke element representation and complexity-aware AI optimization, provides an effective and generalizable solution for Chinese font generation, with clear directions for further improvement on high-complexity characters.

6. Conclusions

6.1. Summary

This paper presents Stroke2Font, a comprehensive framework for dynamic Chinese font generation that addresses the long-standing tension between structural fidelity, stylistic diversity, and computational efficiency—three critical requirements for modern cloud-based typographic services.

Core innovation: hierarchical stroke element representation. The foundation of Stroke2Font is a novel stroke element representation that operates at a finer granularity than traditional strokes while maintaining explicit structural semantics. By decomposing characters into atomic trajectory segments parameterized by Bézier curves and controlled through yoke vectors (α, η), our model achieves unprecedented balance between expressiveness and efficiency. This representation provides both the interpretability of feature-based approaches and the flexibility of data-driven methods.

AI-driven optimization framework. Beyond the core representation, we introduce a three-module optimization framework that intelligently balances quality, creativity, and performance:

Reinforcement learning for real-time rendering: A policy network that dynamically selects Bézier control parameters, minimizing latency while preserving visual quality.

Genetic algorithm for creative exploration: An evolutionary approach that generates novel, esthetically pleasing font variants through crossover and mutation operations.

Adaptive complexity-aware strategy: Crucially, our framework introduces complexity-aware parameter adaptation, which automatically adjusts interpolation and smoothing parameters based on character structural characteristics (stroke element count $\times$ unique types). This adaptive mechanism is key to achieving consistent performance improvements across the diverse landscape of Chinese characters.

6.2. Experimental Validation

Comprehensive evaluation on a curated dataset of 150 Chinese characters (1123 stroke elements, 5287 feature points) yields compelling results, summarized in Table 23.

Table 23. Performance comparison of optimization methods (n = 150).

Key Performance Indicator	Result	Significance
Trajectory Fidelity	65.2% (AI Adaptive) vs. 61.3% (Baseline)	6.4% relative (+3.9 pp) statistically significant improvement
Performance Consistency	$\pm 5.7\%$ standard deviation (vs. $\pm 6.5\%$ Baseline)	More reliable across varying complexity
Generation Latency	<15 ms per character	Suitable for real-time interactive applications
Storage Efficiency	~0.8 KB per character (vs. 8 KB TTF)	90% reduction in storage requirements

Complexity-aware effectiveness: The framework demonstrates consistent improvements across character categories, with the strongest gains for Simple (+6.2%) and Medium (+7.4%) characters. Complex characters remain challenging (−0.05%), indicating that further optimization strategies are needed for high-complexity structures.

6.3. Contributions

This work makes several key contributions to the fields of computational typography and AI-driven design optimization:

1. Novel stroke element representation: We introduce stroke elements as a novel, structured intermediate representation that bridges pixel-level and radical-level approaches. This representation achieves an optimal trade-off, offering fine-grained ge-

ometric control comparable to vector graphics while maintaining the computational tractability necessary for large-scale generative tasks.

2. Geometrically interpretable style control via yoke vectors: We propose a yoke vector mechanism that constructs a mathematically explicit, low-dimensional style manifold. This enables continuous, semantically meaningful interpolation between font styles, providing designers with precise parametric control and serving as an efficient search space for automated optimization algorithms.
3. Adaptive, complexity-aware optimization framework: We develop and validate an adaptive AI optimization strategy that dynamically configures processing parameters based on the structural complexity of each character. This method demonstrates that character-specific tuning outperforms fixed-parameter approaches, achieving a statistically significant 6.4% relative improvement (3.9 percentage points) in trajectory fidelity with reduced performance variance across a diverse character set.
4. End-to-end system architecture for scalable personalization: We present a system architecture that decouples structural templates from style parameters, enabling efficient storage and real-time style personalization. This design enables efficient storage, real-time style personalization, and seamless client-side rendering, addressing critical scalability and latency requirements for modern web and mobile applications.
5. Comprehensive evaluation protocol: We establish and apply a rigorous, multi-faceted evaluation protocol. It integrates Hausdorff distance-based geometric fidelity, curvature continuity analysis, and computational efficiency benchmarks, creating a reproducible standard for assessing generative typography systems.

6.4. Limitations and Future Work

While the proposed Stroke2Font framework demonstrates significant advances in structure-aware font generation, several limitations warrant acknowledgment, which in turn delineate clear and promising avenues for future work.

Current Limitations

1. Scope and coverage: Our evaluation is conducted on 180 GB2312 characters (150 training + 30 holdout). Future work should extend to full CJK character sets and specialized domains (historical scripts, minority scripts).
2. Style transfer capability: The framework excels at parametric style variation within established font families. Cross-family style transfer from arbitrary reference styles represents an important next challenge.
3. Dynamic handwriting modeling: Although stroke elements capture geometric structure, fully modeling the temporal dynamics of handwriting (pressure, velocity, acceleration) requires additional sensor data and modeling techniques.
4. Perceptual evaluation: While geometric metrics provide objective assessment, comprehensive user studies are needed to validate perceptual quality and esthetic appeal across diverse audiences.

Immediate Future Directions

Building upon the current work, we identify several targeted research directions:

1. Adaptive complexity learning: Replace manual complexity thresholds with learned parameters through reinforcement learning or meta-learning approaches.
2. Per-element specialization: Develop stroke type-specific optimization strategies that leverage the distinct geometric properties of different stroke categories.
3. Neural-vector fusion: Explore hybrid architectures that combine the generative power of neural networks with the editability and efficiency of vector representations.

4. Extended applications: Apply the stroke element representation to related tasks including handwritten character recognition, calligraphy style analysis, and educational tools.

6.5. Broader Implications

Beyond technical contributions, Stroke2Font offers several broader insights for computational creativity and digital typography:

1. Granularity matters: Our work demonstrates that operating at an appropriate representational granularity—neither too coarse nor too fine—is crucial for balancing control and automation in creative systems.
2. Interpretability enables optimization: The mathematically explicit nature of our representation enables both human design control and automated AI optimization, suggesting a promising path for human–AI collaborative creativity.
3. Cultural heritage meets modern technology: By providing efficient tools for personalized font generation, our approach helps bridge traditional calligraphic art with contemporary digital expression, potentially revitalizing interest in Chinese typographic heritage.
4. AI as engineering partner: Stroke2Font exemplifies how AI can serve not just as a generative black box, but as an intelligent optimization layer within a well-designed engineering system—a model applicable across many creative computing domains.

6.6. Final Remarks

Stroke2Font represents a significant step toward scalable, high-quality Chinese font generation by addressing the core challenge of simultaneously maintaining structural integrity, stylistic flexibility, and computational efficiency. Our hierarchical stroke element representation, combined with adaptive AI optimization, provides a principled framework that advances both the theoretical understanding and practical implementation of dynamic font generation.

The work demonstrates that careful consideration of representation granularity, combined with intelligent parameter adaptation, can yield substantial improvements over both traditional geometric approaches and purely data-driven methods. As digital typography continues to evolve toward more personalized, dynamic, and culturally expressive forms, approaches like Stroke2Font that balance mathematical precision with creative flexibility will play an increasingly important role in shaping the future of written communication.

We believe the stroke element abstraction introduced in this paper opens new possibilities not only for font generation but also for related areas including character recognition, handwriting analysis, and digital preservation of calligraphic heritage—ultimately contributing to both the technological advancement and cultural vitality of Chinese typography in the digital age.

Author Contributions: Conceptualization, Q.-S.L.; methodology, Q.-S.L.; software, Q.-S.L.; validation, Q.-S.L., Y.-L.B. and Z.-H.C.; formal analysis, Q.-S.L.; investigation, Q.-S.L.; resources, Q.-S.L.; data curation, Y.-L.B.; writing—original draft preparation, Q.-S.L.; writing—review and editing, Q.-S.L., Y.-L.B. and Z.-H.C.; visualization, Y.-L.B.; supervision, Q.-S.L.; project administration, Q.-S.L.; funding acquisition, Q.-S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (General Program, Grant No. 60973051), the Zhejiang Provincial Science and Technology Program (Grant No. 2021C03137), and the Tongxiang Research Institute of Communication University of Zhejiang (Grant No. ZX2023007). The APC was paid by the authors.

Data Availability Statement: The dataset, source code, and experimental results supporting the findings of this study are openly available in the Stroke2Font GitHub repository at <https://github.com/aylqs2025/stroke2font> (accessed on 1 March 2026).

Acknowledgments: This work made use of the Huawei Cangjie programming language and the authors' open-source Cangjie-based web framework CangStream (<https://github.com/qing-sheng-li/CangStream>, accessed on 1 March 2026).

Conflicts of Interest: Author Qing-Sheng Li was employed by company Hangzhou Liangyun Intelligent Technology Co., Ltd., Hangzhou 310023, China. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

- Chen, H. An analysis of the historical origin, development and cultural inheritance of Chinese characters. *Sinogram Cult.* **2022**, *314*, 1–3. [CrossRef]
- Zhang, T.; Liu, M.; Yang, Y.; Zuo, F.; Wang, G. Ancient handwritten Chinese character recognition via multi-style attention and feature fusion. In Proceedings of the 2025 5th International Conference on Artificial Intelligence, Automation and High Performance Computing (AIAHPC), Hefei, China, 19–21 September 2025; pp. 29–32.
- Chen, W.; Su, J.; Song, W.; Xu, J.; Zhu, G.; Li, Y.; Ji, Y.; Liu, C. Quality evaluation methods of handwritten Chinese characters: A comprehensive survey. *Multimed. Syst.* **2024**, *30*, 194. [CrossRef]
- Yang, L.; Wu, Z.; Wu, D.E. Easy recognition of artistic Chinese calligraphic characters. *Vis. Comput.* **2023**, *39*, 3755–3766. [CrossRef]
- GB/T 2312-1980; Information Technology—Chinese Ideogram Coded Character Set (Basic Set). Standardization Administration of China: Beijing, China, 1980. Available online: <https://openstd.samr.gov.cn/bzgk/gb/newGbInfo?hcno=5664A728BD9D523DE3B99BC37AC7A2CC> (accessed on 1 March 2026).
- Wang, L.; Liu, Y.; Sharum, M.Y.; Yaakob, R.; Kasmiran, K.A.; Wang, C. Deep learning for Chinese font generation: A survey. *Expert Syst. Appl.* **2025**, *276*, 127105. [CrossRef]
- Yang, J.; Zhang, M.M.; Zhang, J.W.; Pan, Z.G. Description and Generation of Chinese Outline Based on C-Bézier Curves. *J. Comput. Aided Des. Comput. Graph.* **2000**, *12*, 660–663.
- Lin, J.W.; Hong, C.Y.; Chang, R.I.; Wang, Y.C.; Lin, S.Y.; Ho, J.M. Complete font generation of Chinese characters in personal handwriting style. In Proceedings of the 34th International Performance Computing and Communications Conference, Nanjing, China, 14–16 December 2015; pp. 1–5. Available online: <https://ieeexplore.ieee.org/document/7410313> (accessed on 1 March 2026).
- Li, Q.; Xu, Q.; Xiao, J.; Liu, Q.; Zhang, J. A structure and style model for Chinese character dynamic generation. *Acta Sci. Nat. Univ. Pekin.* **2017**, *53*, 219–229.
- Chen, F.; Wang, C.; Yao, X.; Sun, F. SPFont: Stroke potential features embedded GAN for Chinese calligraphy font generation. *Displays* **2024**, *85*, 102876. [CrossRef]
- Zeng, J.S.; Chen, Q.; Wang, M.W. Self-supervised Chinese font generation based on square-block transformation. *Sci. Sin.* **2022**, *52*, 145–159.
- Wang, K.; Zhou, C.; Shi, Y.; Wu, T.; Chen, C. FourCornerGAN: Glyph formation augmentation for unpaired Chinese font generation. *Digit. Signal Process.* **2025**, *165*, 105305. [CrossRef]
- Lu, P.; Chen, J.; Zou, G.; Wan, Y.; Zheng, Z.; Wang, Z. Personalized Handwritten Chinese Character Generation Method for Unsupervised Image Translation. *Comput. Eng. Appl.* **2022**, *58*, 221–229.
- Chen, W.; Zhu, G.; Li, Y.; Ji, Y.; Liu, C. Advancements in Chinese font generation since deep learning era: A survey. *arXiv* **2025**, arXiv:2508.06900. [CrossRef]
- Deng, L.; Yu, D. Deep learning: Methods and applications. *Found. Trends® Signal Process.* **2014**, *7*, 197–387. [CrossRef]
- Cheng, H.; Shinnick, E.; Wang, X. A visualizing analysis of Chinese character processing in the past 40 years (1981–2020). *Digit. Scholarsh. Humanit.* **2022**, *37*, 336–353. [CrossRef]
- Li, Y.; Li, Y. Design and implementation of handwritten Chinese character recognition method based on CNN and TensorFlow. In Proceedings of the 2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA); IEEE: New York, NY, USA, 2021; pp. 878–882. [CrossRef]
- Zeng, S.; Pan, Z. An unsupervised font style transfer model based on generative adversarial networks. *Multimed. Tools Appl.* **2022**, *81*, 5305–5324. [CrossRef]
- Chang, B.; Zhang, Q.; Pan, S.; Meng, L. Generating handwritten Chinese characters using CycleGAN. In Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV); IEEE: New York, NY, USA, 2018; pp. 199–207. [CrossRef]

20. Wen, Q.; Li, S.; Han, B.; Yuan, Y. ZiGAN: Fine-grained Chinese calligraphy font generation via a few-shot style transfer approach. In Proceedings of the 29th ACM International Conference on Multimedia, Chengdu, China, 20–24 October 2021; pp. 621–629. [\[CrossRef\]](#)
21. Gandhi, S.; Rana, H.; Bhatt, N. Conditional GANs in Image-to-Image Translation: Improving Accuracy and Contextual Relevance in Diverse Datasets. *Procedia Comput. Sci.* **2025**, *252*, 954–963. [\[CrossRef\]](#)
22. Lin, X.; Li, J.; Zeng, H.; Ji, R. Font generation based on least squares conditional generative adversarial nets. *Multimed. Tools Appl.* **2019**, *78*, 783–797. [\[CrossRef\]](#)
23. Zhu, J.Y.; Park, T.; Isola, P.; Efros, A.A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017. [\[CrossRef\]](#)
24. Zhou, P.; Zhao, Z.; Zhang, K.; Li, C.; Wang, C. An end-to-end model for Chinese calligraphy generation. *Multimed. Tools Appl.* **2021**, *80*, 6737–6754. [\[CrossRef\]](#)
25. Liu, J.; Zheng, S.; Cai, Q. Handwritten Chinese Character Recognition Based on Attention Mechanism. In Proceedings of the 2024 3rd International Conference on Artificial Intelligence and Computer Information Technology (AICIT), Yichang, China, 20–22 September 2024; pp. 1–4. [\[CrossRef\]](#)
26. Pan, Z.; Ma, X.; Zhang, M.; Shi, J. The fourier descriptor based automatic generation method for multiple Chinese fonts. *J. Softw.* **1996**, *7*, 331–338.
27. Isola, P.; Zhu, J.Y.; Zhou, T.; Efros, A.A. Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017. [\[CrossRef\]](#)
28. Tian, Y. zi2zi: Master Chinese Calligraphy with Conditional Adversarial Networks. 2017. Available online: <https://kaonashityc.github.io/2017/04/06/zi2zi.html> (accessed on 1 March 2026).
29. Jiang, Y.; Lian, Z.; Tang, Y.; Xiao, J. SCFont: Structure guided Chinese font generation via deep stacked networks. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 4015–4022. [\[CrossRef\]](#)
30. Wu, S.J.; Yang, C.Y.; Hsu, J.Y.J. CalliGAN: Style and structure aware Chinese calligraphy character generator. *arXiv* **2020**. [\[CrossRef\]](#)
31. Wang, Y.-H.; Gao, Y.; Lian, Z.-H. Attribute2Font: Creating Fonts You Want from Attributes. *arXiv* **2020**, arXiv:2005.07865. [\[CrossRef\]](#)
32. Jiang, H.; Yang, G.; Huang, K.; Zhang, R. W-Net: One-shot arbitrary-style Chinese character generation with deep neural networks. *arXiv* **2024**. [\[CrossRef\]](#)
33. Ning, F. Multi-style Migration of Chinese Characters based on Self-attention Mechanism and StarGAN v2. In Proceedings of the 2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA); IEEE: New York, NY, USA, 2022; pp. 357–361. Available online: <https://ieeexplore.ieee.org/document/9825460?denied=> (accessed on 1 March 2026).
34. Carlier, A.; Danelljan, M.; Alahi, A.; Timofte, R. DeepSVG: A Hierarchical Generative Network for Vector Graphics Animation. *arXiv* **2020**. [\[CrossRef\]](#)
35. Reddy, P.; Gharbi, M.; Lukáč, M.; Mitra, N.J. Im2Vec: Synthesizing Vector Graphics without Vector Supervision. *arXiv* **2021**. [\[CrossRef\]](#)
36. Li, Q.; Xiong, J.; Wu, Q.; Yang, Y. Study of feature weighted-based generation method for dian strokes of Chinese character. *Acta Sci. Nat. Univ. Pekin.* **2014**, *50*, 153–160.
37. Luo, X.; Li, Q. Intelligent Compilation System for Chinese Character Animation Based on Dynamic Data Sets. In *Computer Animation and Social Agents*; Mousas, C., Seo, H., Thalmann, D., Cordier, F., Eds.; CASA 2025. Lecture Notes in Computer Science; Springer: Singapore, 2026; Volume 15915. [\[CrossRef\]](#)
38. Li, Z.; Li, Q.; Guan, Y. A Chinese Character Generation Model for Cloud Information Security. In Proceedings of the 2019 IEEE International Conferences on Ubiquitous Computing & Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS); IEEE: New York, NY, USA, 2019; pp. 534–540. [\[CrossRef\]](#)
39. Li, Q.; Li, J.-P.; Chen, L. A Bezier Curve-Based Font Generation Algorithm for Character Fonts. In Proceedings of the 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Exeter, UK, 28–30 June 2018; pp. 1156–1159. [\[CrossRef\]](#)
40. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [\[CrossRef\]](#) [\[PubMed\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.