

FontCopilot: A Designer-in-the-Loop System for Chinese Vector Font Design

Shuhui Shi
Zhejiang University
Hangzhou, China
shuhui.shi@zju.edu.cn

Weiyue Gao
Zhejiang University
Hangzhou, China
gvy@zju.edu.cn

Jinyu Song
Zhejiang University
Hangzhou, China
songjinyu23@gmail.com

Xue Zhao
Zhejiang University
Hangzhou, China
xue-zhao@zju.edu.cn

Weitao You*
Zhejiang University
Hangzhou, China
weitao_you@zju.edu.cn

Wei Wang
Beijing Jiaotong University
Beijing, China
wei.wang@bjtu.edu.cn

Yu Liu
Tsinghua University
Beijing, China
honeyyu5@126.com

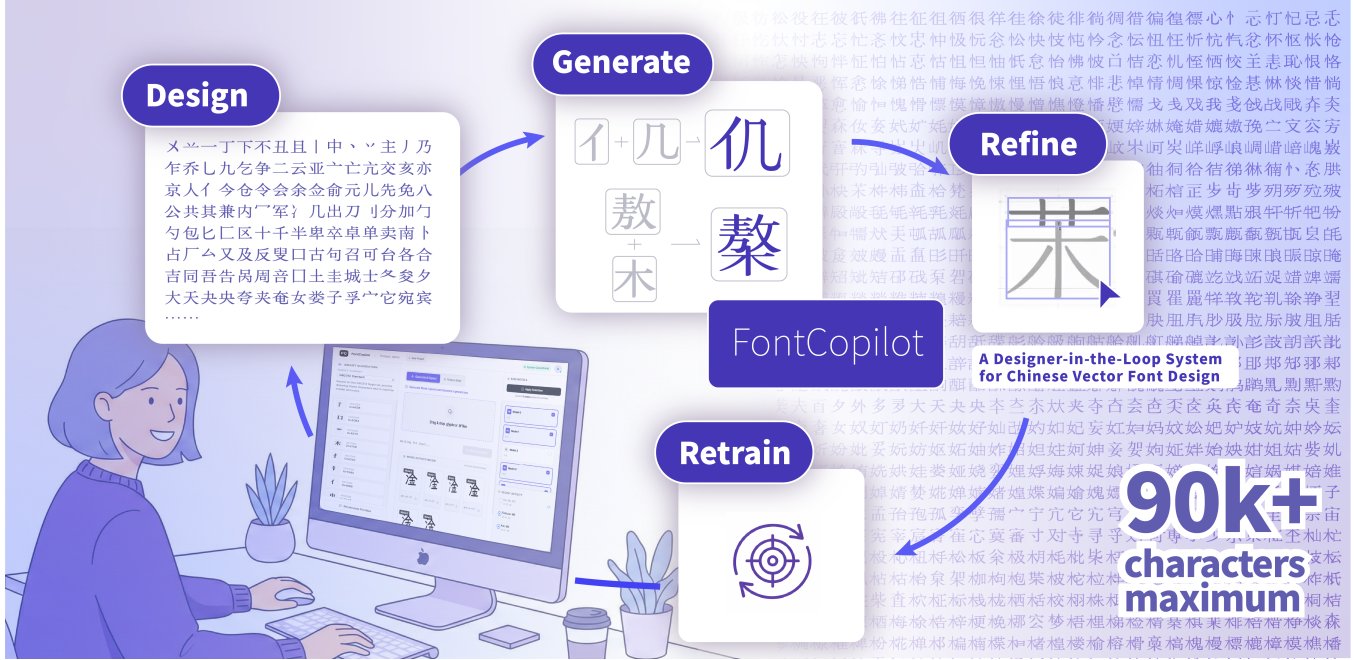


Figure 1: The designer-in-the-loop workflow used to create AI Song with FontCopilot. Starting with manually drawn seed characters, the workflow engages the designer in an iterative interaction loop—Design → Generate → Refine → Retrain—ultimately producing a fully editable, vector-based font file.

*Corresponding author.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
UIST '26, Detroit, MI, USA

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2856-3/2026/11
<https://doi.org/10.1145/3830398.3830559>

Abstract

Designing Chinese fonts is challenging due to the structural complexity of ideographic characters and the scale of character sets (e.g., over 80,000 characters in GB 18030). Existing font generation methods often produce non-editable raster images or offer limited stylistic control, which restricts their use in professional design workflows.

We present FontCopilot, an interactive designer-in-the-loop system for Chinese vector font design. Through an exploratory interview study with 10 professional designers, we identify Component Assembly as a major production bottleneck. FontCopilot targets this labor-intensive phase using a component-based affine transformation model that generates editable vector glyphs from few-shot inputs. The workflow lets designers refine generated results and use their corrections to update the model. These interactions provide direct control over component layout and style.

We evaluate CAR through model ablations and a comparison with a prior Chinese vector font generation method, study FontCopilot with 9 professional designers, and document a four-week production case. FontCopilot received “Excellent” ratings on five of six UEQ dimensions, and one designer used 300 seed characters to produce 10,461 editable glyphs for *AI Song*. Together, these evaluations show how editable component completion can carry local structural decisions from a small seed set to a complete Chinese typeface.

CCS Concepts

• **Human-centered computing** → **Interactive systems and tools**; **Empirical studies in HCI**; • **Computing methodologies** → *Computer vision*.

Keywords

Chinese Font Design, Vector Font Generation, Creativity Support Tools

ACM Reference Format:

Shuhui Shi, Jinyu Song, Wei Wang, Weiyue Gao, Xue Zhao, Yu Liu, and Weitao You. 2026. FontCopilot: A Designer-in-the-Loop System for Chinese Vector Font Design. In *The 39th Annual ACM Symposium on User Interface Software and Technology (UIST '26)*, November 02–05, 2026, Detroit, MI, USA. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3830398.3830559>

1 Introduction

Recent learning-based methods, including GAN- and diffusion-based approaches, have advanced automated font synthesis from limited style exemplars [5, 14, 28]. Yet most image-based approaches produce rasterized outputs rather than editable vector glyphs, and they can have difficulty preserving the topology required for professional font production [7, 12, 18]. Professional workflows require glyphs that can be scaled, inspected, and repeatedly revised as vector outlines. This gap is especially consequential in Chinese typography, where a typeface must maintain structural accuracy and stylistic consistency across thousands of complex characters.

Chinese font design faces three intertwined challenges. First, the scale is large: the foundational GB 2312 standard contains 6,763 characters, while the modern GB 18030-2022 standard expands this inventory to over 80,000 characters. Second, the structure is complex: Chinese glyphs require precise arrangement of nested components, so even small synthesis errors can break legibility or visual consistency. Third, existing tools offer limited agency: many methods operate as one-shot generators that return bitmaps or rigid outputs [11, 14, 28, 29], leaving designers little room to refine style, correct structure, or integrate results into industrial workflows. Recent vector synthesis methods improve automation,

but they do not directly support the fine-grained correction and iterative model updating required in the workflow studied here [11, 13, 29, 34, 35].

To address this gap, we conducted an exploratory interview study with 10 professional designers to examine real workflows and bottlenecks. Our findings reveal that component assembly is the dominant pain point: a repetitive and error-prone stage that consumes substantial effort while contributing little to creative expression. Based on these insights, we developed FontCopilot, a designer-in-the-loop system that automates large-scale component assembly through a component-based affine transformation pipeline, while enabling designers to steer style and make fine-grained corrections. FontCopilot’s Component Affine Transformation Regressor (CAR) predicts resolution-independent transformations for reusable components and applies them to Bézier control points, producing editable vector glyphs. Designers begin with seed characters, review the synthesized results, refine unsatisfactory component layouts, and use their corrections to update the model. This design follows Shneiderman’s “High Automation and High Control” principle [23], combining automated component assembly with designer control over refinement.

Our contributions are:

- We present FontCopilot¹, a designer-in-the-loop system that combines component-based completion with an iterative workflow of seed design, synthesis, refinement, and model updating. The system is supported by a structured inventory of more than 90,000 Chinese characters.
- We introduce CAR, a component-based affine transformation regressor that predicts transformations for reusable components and applies them to vector outlines, leaving the synthesized glyphs editable.
- We evaluate FontCopilot through model ablations and a comparison with a prior Chinese vector font generation method, a study with nine professional font designers, and a four-week production case in which one designer used 300 seed characters to produce the 10,461-character *AI Song* typeface.

2 Related Work

2.1 Chinese Vector Font Generation

Chinese font generation is inherently challenging due to the need to cover tens of thousands of characters with complex internal structures. Early systems relied on rule-based decomposition and component reuse [8, 9, 33], but required extensive manual effort and scaled poorly.

Learning-based approaches improve scalability by modeling glyph structure and style. CVFont [11] introduces component decomposition and layout prediction for large-scale synthesis from limited exemplars. DeepVecFont [29] further bridges raster and vector domains via dual-modality learning, enabling high-quality vector glyph generation. DeepVecFont-v2 [30] replaces the recurrent sequence model with Transformers and introduces auxiliary-point supervision and a self-refinement module for longer and more complex outlines.

¹Project repository: <https://github.com/AnCoSONG/FontCopilot>

Subsequent work focuses on representation and fidelity: DualVector [13] models glyphs as positive/negative path pairs to improve topology, VecFontSDF [34] adopts implicit signed distance functions for geometric precision, and AutoStegaFont [35] leverages differentiable rasterization for vector optimization. These methods improve vector synthesis quality.

However, these methods primarily target automated whole-glyph synthesis. Their outputs remain editable as vector paths, but the systems do not directly support localized component refinement or model updating from designer revisions. FontCopilot instead focuses on component-level completion within an iterative professional workflow.

2.2 Interactive Creativity Support

HCI research on creativity support examines how computational systems can assist creative work while preserving user control. Shneiderman’s “High Automation and High Control” principle [23] argues that effective systems should combine strong automation with user control. This balance matters in professional design, where designers need to inspect, revise, or reject system outputs.

Prior work examines interactive systems that support exploration and iteration [1, 4, 20, 24]. Systems such as DesignScape [17] demonstrate how suggestions can expand the design space while keeping users in control. More recent systems use tightly coupled interaction loops that allow continuous user steering [25, 32].

These findings suggest that for complex design tasks such as font design, synthesis quality alone is insufficient; designers also need ways to inspect and revise system outputs.

2.3 Iterative Refinement and Model Updating

Interactive systems can incorporate user feedback through repeated generation and refinement. Early approaches enable iterative optimization through feedback and selection [16, 26], while recent systems support refinement, steering, and model updates in creative workflows [3, 15, 17, 25].

In graphics and typography, such systems show that combining algorithmic generation with human judgment can improve controllability. Interactive vector design tools allow users to guide structure and style while delegating low-level geometry generation [27, 31].

FontCopilot uses this interaction pattern to address the limitations in Section 2.1. Rather than treating synthesis as a one-time whole-glyph operation, it uses an iterative loop in which the system assembles components, designers correct structure and style, and the corrected glyphs are used to update the model. This workflow combines automated component assembly with designer control over refinement.

3 Exploratory Interview Study

3.1 Participants, Procedure, and Analysis

We recruited 10 experienced practitioners (P1–P10) in Chinese font design through the authors’ professional networks and participant referrals within the Chinese type-design community. Their experience ranged from less than one year to more than ten years (Appendix A). The first author conducted approximately 30-minute semi-structured interviews in Chinese via Tencent Meeting. The

study received institutional ethics approval; participants provided informed consent and received compensation. Sessions were audio-recorded, automatically transcribed, and manually checked against the recordings. The authors translated the reported quotations into English.

We followed Braun and Clarke’s thematic analysis procedure [2]. The first author reviewed the transcripts, iteratively coded passages about workflows, tools, production bottlenecks, and expectations for technical support, and grouped related codes into candidate themes. The first author checked the themes against the transcripts, and the co-authors refined them through discussion until reaching consensus. The interview protocol is provided in Appendix A.

3.2 Current Workflow of Chinese Font Design

Chinese font design follows a structured yet labor-intensive pipeline. Designers begin with **Design Conception**, establishing the visual principles of a typeface with representative characters. Four participants (P1, P3, P5, P6) described using the “12-character rule” (永意风国今警我然转随鹰酬), whose varied strokes and structures support early style testing. For example, 永 includes most of the basic stroke types used in Chinese characters, 风 tests a semi-enclosed structure, and 酬 shows how the style handles a character with many vertical strokes. P10 extended the initial set to 20–30 characters for additional edge cases. During **Seed Expansion**, designers create approximately 400–500 base glyphs to cover common components and structural patterns (P1, P2).

The process then transitions to **Component Assembly**, where thousands of characters are constructed by reusing and recombining components (P2, P5, P7). This stage dominates production time and often involves team-based or outsourced workflows. Participants reported using Glyphs, Adobe Illustrator, and proprietary foundry software to manage components and vector structures, but much of the large-scale retrieval, assembly, and correction work remains manual.

Finally, designers perform **Optimization and Release**, including contour correction, stroke consistency adjustment, and structural normalization (P3, P4, P10). This stage often involves converting parametric representations into static outlines to ensure stability and compatibility.

3.3 Core Challenges in Practice

Challenge 1: Massive Scale and Repetitive Labor. The component assembly stage is universally identified as the primary challenge. Participants described it as a high-intensity repetitive process; for example, P2 noted that completing a standard Song-style font typically requires a 2–3 person team collaborating for about six months. Designers (P5, P7) characterized this stage as “boring mechanical work,” while P8 highlighted its contribution to severe occupational burnout. The scale of production amplifies both labor cost and error propagation across thousands of glyphs.

Challenge 2: Vector-Native Topology Constraints. Existing image-based synthesis tools did not meet the participants’ production requirements. P7, P8, and P10 reported that tracing bitmap outputs into vectors often produced messy nodes and imprecise control points. P10’s “60-point standard” required correct structure with few control points left to adjust. Participants treated

Table 1: Mapping from exploratory interview insights to system design goals. We mapped representative expert quotes and extracted codes into core challenges, establishing the corresponding design goals for FontCopilot.

Representative Evidence & Code	Core Challenge	Design Goal (DG)
“Completing a standard Song-style font typically requires a 2-3 person team collaborating for about six months.” (P2) Code: High-intensity repetitive labor.	Challenge 1: Massive scale of the Chinese character inventory (tens of thousands).	DG1: Automate Scalable Component Assembly. Relieve designers from the mechanical assembly of complex glyphs.
“These tools generate bitmaps. The bitmap-to-vector quality is poor, the control-point placement is imprecise, and I still have to trace the result myself.” (P7) Code: Output format mismatch.	Challenge 2: Preserving strict vector-native topology and editability.	DG2: Preserve Vector-Native Editability. Generate editable vector control points rather than rasterized pixels.
P10 used documents and review meetings to direct outsourced assembly; P8 wanted technical support to handle production while designers retained creative decisions. Code: Separation of design and production.	Challenge 3: The stylistic steering gap across design and production.	DG3: Support Iterative Designer Steering. Provide an interactive loop for designers to iteratively refine and retrain.

editable vector structure as a prerequisite for further production work, rather than judging outputs only by raster appearance.

Challenge 3: Lack of Stylistic Agency. The separation of design and production can weaken designers’ direct control over how their intent is carried into assembled glyphs. P10 used documents and review meetings to direct outsourced assembly, which increased communication and review work. P8 wanted technical support to take over production while leaving creative decisions with the designer. These accounts motivated an iterative workflow in which designers can inspect and correct assembled glyphs before updating the model.

3.4 Design Goals

Based on these findings, we define three design goals (DG1–DG3): scalable component assembly, vector-native editability, and iterative designer steering (Table 1).

4 Method

4.1 Dataset

Existing image- and vector-based approaches for Chinese font generation [5, 7, 12, 14, 18, 28] often fail to meet professional requirements because (a) public or commercial glyph databases are not always open or accessible, (b) open datasets lack consistent, fine-grained component/topology annotations for complex nested structures, and (c) many resources provide only Unicode mappings without explicit visual–component correspondence essential for structural supervision. To address **DG1: Automate Scalable Component Assembly** and **DG2: Preserve Vector-Native Editability**, we constructed a purpose-built decomposition dataset that directly maps Unicode-encoded characters to component lists and layout types; this mapping enables large-scale automated component assembly while preserving vector fidelity.

Our collection comprises 92,560 Unicode-encoded characters (see Appendix B for provenance and formatting). Each record contains: the Unicode codepoint, an ordered list of constituent components,

and a layout label drawn from our six-class scheme (NL00–NL05). Figure 2 illustrates these six layouts: **NL00** Isolated, **NL01** Left–Right, **NL02** Top–Bottom, **NL03** Enclosed, **NL04** Left–Middle–Right, and **NL05** Top–Middle–Bottom. Notably, **NL03** subsumes eight traditional enclosure variants (e.g., full enclosure, upper enclosure, left–lower enclosure). We merge these variants into one class because subtype frequencies are sparse and would impede neural network convergence; our model can *adaptively learn* the enclosure-specific geometry during training.

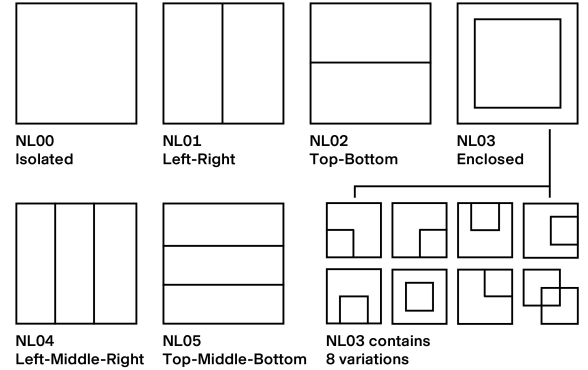


Figure 2: Six main layouts in our dataset. NL03 (Enclosed) covers eight traditional enclosure variants but is labeled as a single class to ensure sufficient sample density and enable adaptive learning.

4.1.1 Crowdsourced Annotation and Quality Control. Due to the massive scale of the dataset, we collaborated with a professional crowdsourcing partner for component decomposition and layout annotation. Workers were strictly selected based on their deep familiarity with Chinese glyph structures. Before annotation, they received standardized training via online workshops that demonstrated the full workflow and covered edge cases in detail.

Quality control included: (1) periodic random audits during annotation, (2) automated sanity checks for component and layout consistency, and (3) two rounds of expert review by 15 graduate volunteers, each assigned different dataset partitions. Discrepancies triggered targeted re-annotation or specific retraining sessions. This rigorous process ensures reliable supervision suitable for scalable assembly (DG1) and precise vector generation (DG2).

4.2 Model

Our model synthesizes Chinese characters by combining component outlines using affine transformations. The core module, CAR (Component Affine transformation Regressor), predicts a resolution-independent affine matrix for each component. These matrices are applied directly to the components' Bézier control points to produce high-fidelity, editable vector outputs, consistent with DG2.

4.2.1 Model Architecture. For a character C with components $R1, R2, \dots$, CAR generates a synthetic character S by merging the transformed components. The architecture, shown in Figure 3, comprises three modules: a shared **feature extractor**, a **feature fusion module**, and the **regressor**.

Feature extractor computes embeddings for each component. Any pretrained image encoder can be used; in our experiments, we use VGG19-bn pretrained on ImageNet. Weights are shared across all components to maintain consistency.

Feature fusion module aggregates embeddings of all components to provide context-aware inputs to the regressor. For example, in a left-right layout, the right component's stroke density can inform the scaling of the left component. Fusion strategies include stacking, AdaIN, and cross-attention; performance varies slightly across layouts. Details and layout-specific selection rules are in Appendix C.5.

Regressor is an n -layer MLP that outputs affine matrices. The final linear layer is initialized with zero weights and identity bias, ensuring initial transformations are identity and gradually refined during training. Predicted matrices are applied to component outlines, which are then assembled to form the final glyph.

4.2.2 Loss and Optimization. CAR is trained using a composite loss that penalizes structural anomalies (e.g., component overlap or imbalance) while encouraging visual fidelity. Training occurs in raster space for efficiency; after convergence, the predicted affine parameters are directly applied to the vector control points of the original components without retraining. This design bridges raster-based training efficiency and high-precision vector inference. Detailed loss formulations, hyperparameters, and iterative layout strategies are provided in Appendix C.1.

5 System Design

5.1 System Workflow

FontCopilot implements **DG3: Support Iterative Designer Steering** through a three-stage designer-in-the-loop workflow: Design Conception, Interactive Generation, and Optimization and Release (Figure 4).

Design Conception. The designer begins by selecting a target standard (e.g., GB 2312) and supplying a small set of seed characters.

Using the dataset mappings from each character to its layout and components, FontCopilot counts how many target characters can be assembled from each candidate's components and ranks the candidates by this compositional yield. As the designer copies or completes suggested characters, the panel advances to the next candidates. Changing the target prompts the system to recalculate the ranking.

Interactive Generation. With a seed set in place, the designer enters the following loop:

Design → Generate → Refine → Retrain.

After the designed characters are uploaded, CAR (Section 4.2) assembles components to synthesize new glyphs. The designer reviews the results, downloads selected glyphs, adjusts problematic component bounding boxes in Glyphs, and uploads the corrected .glyphs file for incremental fine-tuning. Each round uses these corrections to update component placement while preserving the component outlines as editable vectors.

Optimization and Release. As the character inventory grows, the designer may correct outputs after each round or consolidate the corrections once coverage is sufficient. The system can produce glyphs beyond the immediate target set, from which the designer selects the characters required for the final font.

5.2 Interface Design

FontCopilot is implemented as a web dashboard using Vue and FastAPI. It supports the upload and download of both .glyphs and .ttf files. Downloaded .glyphs files retain component and layer information for further editing in Glyphs.

As shown in Figure 5, the interface uses a three-column dashboard.

Left Sidebar (Smart Suggestion): The designer selects a target standard, such as GB 2312 or GB 18030, and views the ranked character suggestions. Each item provides "Copy" (Unicode) and "Check" actions. Marking a character as complete removes it from the current list and displays the next suggestion.

Central Workspace (Core Interaction Loop): The designer switches between "Generate & Expand" and "Finetune Model" modes and uploads a .glyphs or .ttf file. Generated glyphs appear in a preview grid; the designer can inspect a larger preview and download a .glyphs file for correction in Glyphs or a .ttf file for font output.

Right Sidebar (Model Control & History): The designer can select one or more base models (Model 0–Model 6) and compare their outputs in the central workspace. The "Recent Activity" timeline records generation and fine-tuning operations.

6 Evaluation

We evaluate FontCopilot from four complementary perspectives: (1) model-level performance validation through ablation analysis (Section 6.1), (2) a quantitative comparison against prior Chinese vector font generation methods (Section 6.2), (3) user experience in interactive workflows (Section 6.3), and (4) real-world applicability via a production case study (Section 6.4).

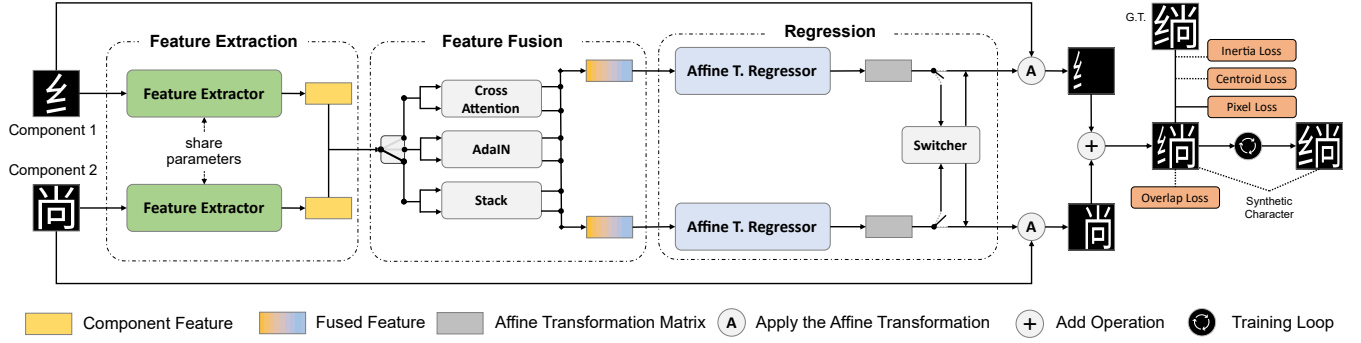


Figure 3: Overall model architecture: feature extractor → fusion module → CAR regressor.

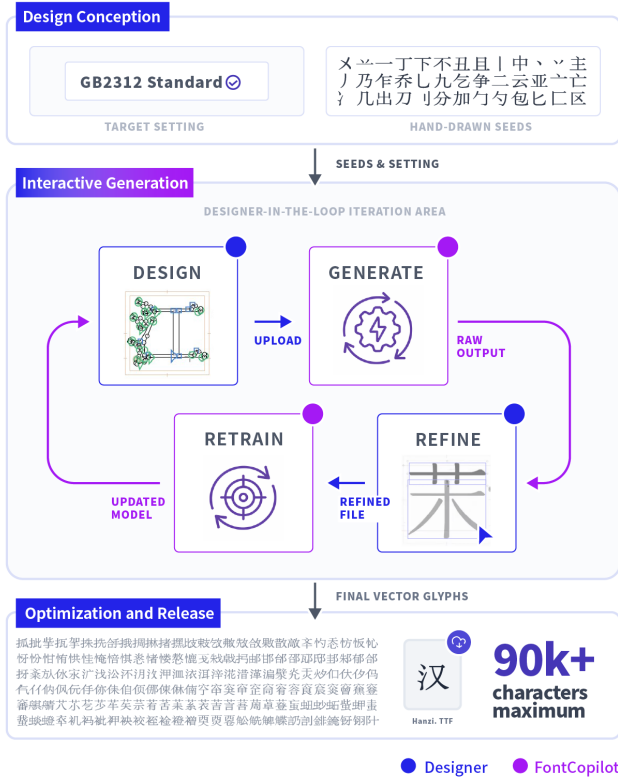


Figure 4: FontCopilot's designer-in-the-loop workflow, consisting of Design Conception, Interactive Generation, and Optimization and Release.

6.1 Ablation Study

6.1.1 Experimental Settings. To validate the effectiveness of our algorithmic design choices, we conduct experiments on seven widely-used Chinese fonts (e.g., SourceHanSans, BabelStone), covering diverse structural styles. Each dataset is split into 80% training and 20% validation with fixed random seeds to ensure reproducibility.

We train all variants under a unified setting, rendering character and component images at a 256×256 resolution, where glyphs are decomposed and reassembled through the proposed component assembly paradigm. Standard reconstruction and perceptual metrics

are adopted, including MAE, RMSE, LPIPS [36], and FID [6], to evaluate both pixel-level fidelity and holistic perceptual quality.

Detailed hyperparameter configurations (e.g., optimizers, learning rates, and specific loss weights) alongside the full quantitative results are provided in Appendix C.

6.1.2 Results. Effect of Structural Constraints. Figure 6 highlights the impact of each loss on structural fidelity. Removing the proposed structural constraints leads to noticeable artifacts in component assembly, including misalignment, distortion, and broken stroke continuity. In contrast, the introduction of the *centroid loss* aligns the centroids and corrects severe misalignments. The *inertia loss* helps stabilize affine transformations, preventing excessive geometric deviation and preserving consistent stroke placement across characters. The *overlap loss* further reduces invalid component intersections, ensuring cleaner spatial composition. Quantitatively, Table 2 confirms this progression on Source Han Sans NL03. Compared to the pixel-loss baseline, centroid loss yields a large reduction in reconstruction and perceptual error (FID: 81.88 → 49.51; LPIPS: 0.380 → 0.315). Inertia loss further lowers FID and LPIPS, although MAE and RMSE rise very slightly. With overlap loss enabled, all four metrics reach their lowest values.

Effect of Architectural Design. We further evaluate three architectural choices beyond the loss design (Tables 3, 4, and 5). Table 3 shows that feature fusion consistently improves performance across multiple layouts. Without fusion, the regressor tends to predict near-constant affine transformations for individual components. On NL02, enabling the switcher module to adapt a pre-trained NL01 model outperforms training from scratch (Table 4, MAE: 0.1683 → 0.1562). For data-limited three-component layouts, Table 5 shows that iterative invocation of the two-component CAR also reduces error on NL04 and NL05 compared to a dedicated three-component model (e.g., NL05 MAE: 0.2217 → 0.1836).

Implications for the Design Workflow. Beyond pixel-level improvements, these loss and architectural designs play a critical role in supporting the interactive font-design workflow. When structural losses and architectural modules work together, generated glyphs exhibit more consistent spatial relationships between components, allowing designers to refine proportion and spacing through bounding-box adjustments rather than repairing gross misassembly. This directly supports downstream refinement in the

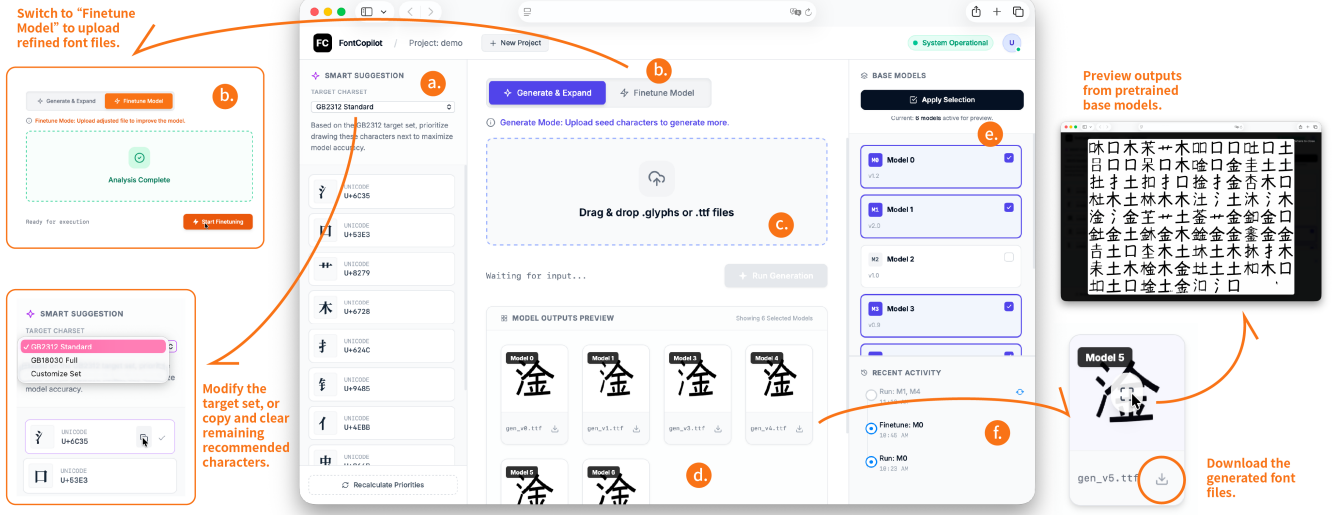


Figure 5: The FontCopilot interface: (a) the “Smart Suggestion” character-recommendation panel; (b) controls for “Generate & Expand” and “Finetune Model”; (c) the file-upload area; (d) the glyph preview and download area; (e) base-model selection; and (f) the Recent Activity timeline.

interactive workflow, where designers rely on stable and predictable component structures for efficient component-layout refinement.

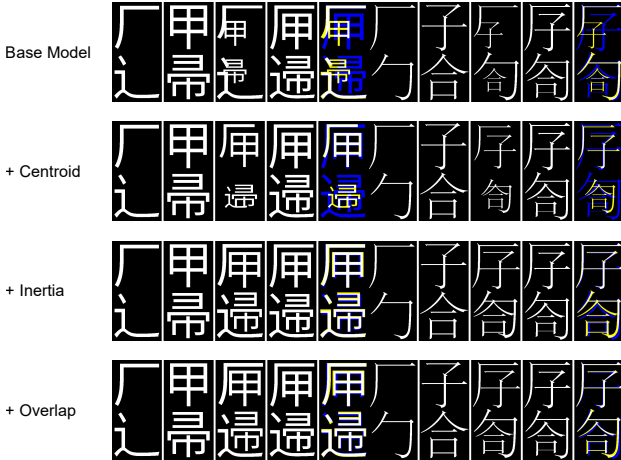


Figure 6: The ablation study of our losses. From left to right, the images represent component 1, component 2, synthetic character, ground truth, and the difference between synthetic character and ground truth, respectively. The color overlay indicates regions present only in the ground truth (blue), only in the synthesis (yellow), or in both (white).

A comprehensive analysis of all ablation variants is provided in Appendix D. Additional successful and failure cases across the primary two-component layouts are provided in Appendix E.

Model Setting	MAE↓	RMSE↓	FID↓	LPIPS↓
Base Model	0.2064	0.4499	81.8785	0.3799
+ Centroid	0.1912	0.4309	49.5050	0.3150
+ Inertia	0.1936	0.4316	35.8110	0.2757
+ Overlap	0.1891	0.4264	33.4290	0.2709

Table 2: The quantitative ablation study of losses, evaluated on SourceHanSans (NL03).

Layout	Fusion	MAE↓	RMSE↓	FID↓	LPIPS↓
NL01	✓	0.1312	0.3540	12.7156	0.2019
NL01	×	0.1676	0.4022	13.6127	0.2332
NL02	✓	0.1562	0.3880	13.0982	0.2210
NL02	×	0.2630	0.5079	17.8102	0.3234
NL03-0,1	✓	0.1891	0.4264	33.4290	0.2709
NL03-0,1	×	0.2687	0.5113	59.3721	0.4039

Table 3: The evaluation results of the feature fusion, evaluated on SourceHanSans. NL03-0,1 denote the models are trained on the data of the first and second variations of NL03.

Switcher	MAE↓	RMSE↓	FID↓	LPIPS↓
×	0.1683	0.4035	13.2593	0.2341
✓	0.1562	0.3880	13.0982	0.2210

Table 4: The ablation results for the Switcher on NL02.

Model	Layout	MAE↓	RMSE↓	FID↓	LPIPS↓
NL01 I.I.	NL04	0.2270	0.4683	67.0076	0.3407
NL04	NL04	0.2474	0.4913	97.0602	0.3782
NL02 I.I.	NL05	0.1836	0.4212	33.0506	0.3209
NL05	NL05	0.2217	0.4648	54.8968	0.3815

Table 5: The ablation study of iterative invocation strategy. I.I. represents iterative invocation.

纒涌嫖颯嚶鋤鯁緹擷絨級棕語
 拙裨剔駛榔纓頻矜揍擢螭隸梁
 津外保謹伐沙球幌驴纓蹠鼓低
 讹鞫肱俑肱蛙钐褸浩蚰晒削狙

Figure 7: The 52 FZSYJ characters used for evaluation.

6.2 Comparison with Prior Chinese Vector Font Generation Methods

CAR is designed to fit practical font production pipelines, generating Chinese vector glyphs by composing designer-authored components. Prior methods, by contrast, are predominantly style-reference-based. Despite this formulation mismatch, we still compare against DeepVecFont-v2 (DVF-v2) [30]—the most suitable available baseline—to more comprehensively assess synthesis quality. Other methods are excluded because of the lack of an accessible implementation (e.g., CVFont [11]), a Latin-only design (e.g., DualVector [13], VecFontSDF [34]), or failure to converge in our reproduction (e.g., DeepVecFont [29]).

6.2.1 Experimental Protocol. To rigorously evaluate generative performance in real-world scenarios, we compare our method CAR against DVF-v2 on complex Chinese vector character generation. We follow DVF-v2’s official training procedure and the fonts released in its repository; Table 6 summarizes both configurations. Because DVF-v2 requires the target character set to be fixed before training, we randomly sample 52 evaluation characters from FangZhengShuiYunJian (FZSYJ), as shown in Figure 7. DVF-v2 is trained on these characters across 210 fonts for 750 epochs, whereas CAR is trained for 42 epochs on the FZSYJ NL01 split (2,833 characters) and never observes the 52 held-out glyphs. We set `max_seq_len`=231 as complex Chinese characters require substantially longer sequence lengths than the default configuration (71) provided in the official repository. Both models are evaluated on the same 52 glyphs.

6.2.2 Results. In this experiment, CAR outperforms DVF-v2 both quantitatively and qualitatively. As shown in Table 7, CAR obtains lower values on all metrics, with the largest gains in FID (283.34 → 48.21) and LPIPS (0.4671 → 0.2446); as shown in Figure 8, CAR also produces visibly more faithful glyphs. This is consistent with the mechanism of the two approaches: DVF-v2 directly generates every control point of glyphs and degrades in quality as stroke count and path length grow, whereas CAR inherits stroke geometry

	DeepVecFont-v2	CAR (Ours)
Train on	210 fonts × 52 chars	FZSYJ (2,833 chars)
Test on	FZSYJ (52 chars)	FZSYJ (52 chars)
Epochs	750	42
ref_nshot	8	N/A
max_seq_len	231	No limitation
Learning rate	1×10^{-4}	2×10^{-3}

Table 6: Training configuration of both models in the comparison with DeepVecFont-v2.

Method	MAE↓	RMSE↓	FID↓	LPIPS↓
DeepVecFont-v2 [30]	0.3777	0.6042	283.34	0.4671
CAR (Ours)	0.1713	0.4068	48.21	0.2446

Table 7: Comparison with DeepVecFont-v2 on the 52 FZSYJ test characters, under DVF-v2’s official protocol. All metrics are lower-is-better.



Figure 8: Qualitative comparison with DeepVecFont-v2 on the 52 FZSYJ test characters. Within each of the two groups, from left to right: component 1, component 2, the glyph synthesized by CAR, the glyph synthesized by DVF-v2, and the ground truth.

directly from designer-authored components and casts complex Chinese vector font generation as regressing component affine transformation matrices, thereby synthesizing complex Chinese characters at much higher quality.

This FZSYJ comparison and the ablations in Section 6.1 follow different protocols (training fonts, character splits, and test subsets), so their metrics should not be compared across tables.

6.3 User Study

6.3.1 Participants. To ensure continuity with the exploratory interview study (Section 3), we invited the same group of professional designers to evaluate the system. Nine participants (P1–P9) took part in this study, while P10 was unavailable due to scheduling constraints. All participants are experienced practitioners in Chinese font design, and their demographic details are provided in Appendix A. Each participant received compensation for their time.

6.3.2 Experimental Settings. The study was conducted remotely via Tencent Meeting, with each session lasting approximately two hours. All participants provided informed consent prior to the experiment.

We designed a task-driven evaluation to simulate realistic font production. Figure 9 illustrates the interactive workflow through the specific case of P3. First, designers created 7 predetermined seed characters using Glyphs. Unlike the traditional “12-character rule,”

these 7 seeds were selected by our auto-recommendation algorithm (Section 5.1) to maximize the combinatorial yield of components.

Participants then uploaded the initial 7 seeds to FontCopilot to generate a first batch of 36 characters. After previewing the generated outputs across different base models, they downloaded a selected candidate set. They then refined these 36 results in Glyphs by adjusting the bounding boxes of the components, and fed the corrected .glyphs file back into the system for retraining. In the subsequent iteration, the fine-tuned model expanded the character set to 101 glyphs.

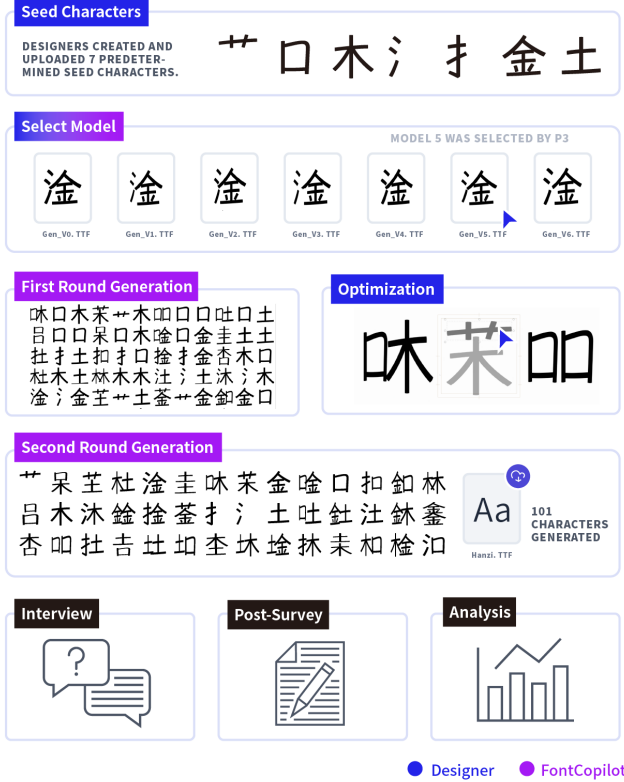


Figure 9: The iterative designer-in-the-loop process illustrated through the case of P3. Starting from 7 algorithmically recommended seeds, the designer selects a base model, refines the structural layout in Glyphs, and retrain the system to expand the character set to 101 glyphs.

6.3.3 Data Collection and Analysis. Following the task, participants completed the User Experience Questionnaire (UEQ) [10]. In our analysis, we focus particularly on *Efficiency* and *Dependability*, as they relate directly to whether designers could work efficiently and maintain control while iteratively refining generated glyphs. The official UEQ benchmark dataset [22] provides a reference distribution for contextualizing the scores rather than a comparison with a competing product.

Additionally, we conducted semi-structured interviews to capture expert reflections on workflow integration and controllability. Interview data from P1–P9 were transcribed and analyzed using

Table 8: UEQ Analysis Results. (Scale: -3 to +3)

Scale	Mean	SD	Benchmark [†]	Result
Attractiveness	2.37	0.56	≥ 1.75	Excellent
Perspicuity	1.89	0.72	≥ 1.90	Good
Efficiency	2.06	0.43	≥ 1.78	Excellent
Dependability	2.03	0.76	≥ 1.65	Excellent
Stimulation	2.28	0.69	≥ 1.55	Excellent
Novelty	2.06	0.62	≥ 1.40	Excellent

[†] Threshold for “Excellent” category in the official dataset.

thematic analysis [2], focusing on how the system addresses core professional pain points.

6.3.4 Results. We report UEQ scores against the established benchmark and interpret them alongside participants’ reflections on FontCopilot relative to their usual workflows.

Quantitative Assessment (UEQ). As shown in Table 8, participants rated FontCopilot “Excellent” in five out of six dimensions, including *Efficiency* (2.06) and *Dependability* (2.03). These scores place it within the top 10% tier of the UEQ benchmark.

System Strengths. Participants consistently reported efficiency benefits, particularly from automating scalable component assembly. As P7 noted, the system “omits the tedious mechanical assembly process,” while P5 described the workflow as “much lighter.”

Equally important is the preservation of vector-native editability. Designers emphasized that generated results remain editable within standard tools. P2 highlighted that outputs are “directly adjustable on the generated results,” and P7 noted that components can be “freely decomposed and refined” without suffering from rasterization. This allowed designers to continue refinement in their usual vector-font editor.

Participants also described how their corrections influenced later outputs. P4 described the feedback process as “transparent and predictable,” noting that local bounding-box adjustments were reflected in subsequent generations.

System Limitations and Future Directions. Despite these strengths, participants identified limitations in fine-grained stroke modeling. At the stroke level, pure affine transformations can introduce rigidity; P2 noted “stroke weight inconsistency” under extreme geometric compression. At the structural level, P8 pointed out the lack of a global preview panel, which sometimes caused a “blind” feeling when assessing overall stylistic consistency across the font family.

Participants suggested parametric stroke-weight compensation, more component-aware modeling, and global adjustment mechanisms. These suggestions define concrete extensions to the current affine assembly approach.

6.4 Case Study: AI Song

To examine FontCopilot in a realistic production setting, we conducted a four-week, author-supported collaboration with an expert type designer to develop a complete Chinese font titled *AI Song*. This case study documents sustained use of the designer-in-the-loop workflow across end-to-end font production.

Design Conception

榕城花下那江南

Seed Characters

メ一丁下不且且 | 中、ノ主ノ乃乍乔し九乞争二云亚亡亢
交亥亦京人イ今令余余余元儿先免八公共其兼内〃军 凡
出刀 刃分加勾勺包匕匚区十千半卑卓单卖南卜占厂厶又及反
叟口古句召可台各合吉同吾告局周音口土圭城士冬夕天央
央夸夹奄女婁子孚〃它宛宴寸寿小少尚衰尧尸屯山工已巴干广
升开弋弓弗弟ヨシイ心↑ 戈戔我户扁手扌执支攴攴文斤方旦
昔更 〃曼曾月有木朱東林果棠榕欠止正此歹爻每比毛氏气水
江火 炎焦 牙牛犬 王瓜瓦甘甫田由甲申番户登白皇皮皿日
矢石示 票禾秋穴立章米糸 羊羽者耒耳肖自至舌舟良 艹
花莫虫衣 囊西见角言 贝走足辛辛辟辰 那西里金 长门
佳雨需青非革韦页风 马骨高影鬼鱼鸟鹿黑齐齿龙 衣 卜
 月

Figure 10: The design conception of *AI Song*. The designer sketched a poetic phrase to establish the intended Song-style appearance before expanding the component inventory.

Over the course of the project, the designer manually created 300 seed characters, from which FontCopilot synthesized 10,461 glyphs covering the GB 2312 standard and thousands of extended characters. The case records how component assembly and vector-native generation supported expansion from a curated seed set to the target font inventory.

Design Conception. The designer began by sketching the phrase “榕城花下那江南” to define a feminine and elegant Song-style aesthetic (Figure 10). After this stylistic anchor was formed, the designer selected GB 2312 as the target character set and used FontCopilot’s recommendation pipeline to progressively expand the component inventory.

Interactive Generation. Following the system’s generation sequence, the designer first completed a set of fundamental components (e.g., “彳”, “口”, “艹”) and uploaded them to the system. FontCopilot then generated candidate outputs across multiple base models. The designer visually inspected these in the web interface, selected the candidate closest to the intended style, and downloaded it to perform structural refinements within Glyphs by adjusting component bounding boxes.

The designer completed two rounds of interaction. After each round, the adjusted structural layouts were uploaded to the pipeline for fine-tuning, updating component placements in later outputs. This process let the designer steer glyph proportions through localized bounding-box edits rather than redrawing complex vector outlines.

Through this iterative loop, 300 manually designed characters supported the synthesis of the full 10,461-character inventory. The designer completed the production work over four weeks on a part-time basis. For context, P2 estimated in Section 3.3 that a 2–3 person team typically spends about six months producing a standard Song-style font.

Optimization and Release. To document the output produced directly by the iterative workflow, we exported all 10,461 glyphs

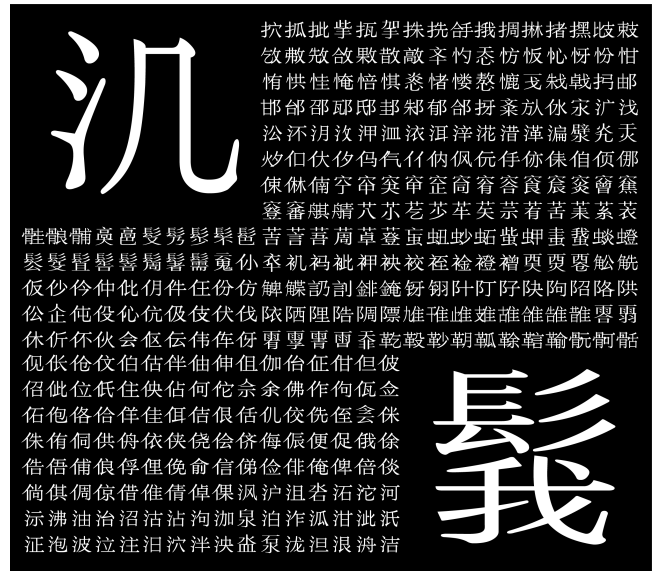


Figure 11: Raw specimens of the *AI Song* font generated by FontCopilot after two rounds of interaction. The final outputs were exported without per-glyph post-correction.

without comprehensive per-glyph post-editing. The resulting .ttf retained editable vector structures across the target inventory, giving the designer a complete font-scale output for final review and refinement (Figure 11).

Together, the four-week process provides sustained-use evidence that FontCopilot can support end-to-end production at this scale. This evidence comes from one author-supported case with a single designer, which defines its scope. The final *AI Song* font is included in the supplementary materials as a .ttf file.

7 Discussion

7.1 Predictable Automation through Editable Components

Chinese font production turns a local design decision into a problem of scale: a component must remain structurally and stylistically coherent when reused across thousands of characters. FontCopilot makes designer-created components the unit of both completion and correction. CAR preserves vector structure during affine composition, leaving generated glyphs inspectable and revisable in Glyphs. Its predictability makes errors traceable to identifiable components and correctable through local edits, while its constraint to existing geometry defines the method’s scope.

The iterative loop makes these corrections cumulative. The system handles component reuse and placement, while designers create seed glyphs, select candidates, adjust bounding boxes, and use corrected glyphs to update the model. Each edit corrects the current glyph and informs component placement in later outputs. This combines high automation with high control [23] and follows an interactive machine learning pattern [1]. The user study’s *Efficiency* and *Dependability* ratings, together with descriptions of the process as transparent and predictable, indicate that participants valued

both reduced assembly work and continued control over refinement.

7.2 From Local Corrections to Font-Scale Production

The evaluation follows this progression across three scales. The ablations and comparison with a prior Chinese vector font generation method examine CAR’s synthesis quality under the reported experimental settings. In the approximately two-hour user study, nine designers began with 7 seed characters, corrected the first 36 outputs in Glyphs, and retrained the system to expand the set to 101 glyphs. The *AI Song* case followed the same workflow over four weeks, expanding 300 seed characters into 10,461 glyphs. These stages connect local transformation, interactive correction, and production-scale expansion within one workflow.

Across the studies, participants described mechanical assembly as tedious and the FontCopilot workflow as lighter; P4 described the correction process as transparent and predictable. These accounts suggest that repeated construction and verification can shift to the system while design judgment remains active, consistent with cognitive offloading [21]. In *AI Song*, the designer established the intended style through a poetic phrase and seed components, then used two rounds of bounding-box correction and fine-tuning. At font scale, the loop carries designer corrections forward alongside glyph generation.

7.3 Limitations and Future Work

CAR is constrained by the geometry of the available components. It cannot create new stroke shapes, and extreme compression can cause stroke-weight inconsistency or structural rigidity. The approach may therefore be less suitable for highly calligraphic or decorative styles that require non-affine deformation. Future work could investigate parametric component representations, stroke-weight compensation, and collision-aware constraints while retaining editable vector outputs. Representative failure cases and further discussion are provided in Appendix E.

The current workflow also requires designers to download generated glyphs files, correct them in Glyphs, and upload the revisions for fine-tuning. This preserves compatibility with an established font editor but introduces context switching. In addition, P8 reported that the lack of a global preview could make it difficult to judge consistency across the font family. Integrating multi-glyph previews and global adjustment tools could support these assessments earlier in the workflow.

The user study involved nine experienced practitioners completing an approximately two-hour task, and its UEQ scores situate their reported experience within the UEQ reference distribution. The *AI Song* case complements this short-session evaluation with four weeks of sustained use by one designer in an author-supported production process. Longer-term deployments with additional designers could examine learning, correction effort, style coverage, and the quality of fonts after final designer review.

8 Conclusion

Chinese vector font design is a problem of scale: a component decision made in one glyph must remain structurally and stylistically

coherent across thousands of others. We presented FontCopilot, a designer-in-the-loop system that addresses this problem through editable component completion. CAR predicts affine transformations for designer-authored vector components, while designers inspect the synthesized glyphs, adjust component layouts, and use the revised glyphs to update later outputs. We derived the workflow requirements from an exploratory interview study with 10 professional designers and evaluated the system through model ablations, a comparison with a prior Chinese vector font generation method, a study with nine professional designers, and the four-week *AI Song* production case. Participants reported efficiency benefits, valued the editable outputs, and described the workflow as transparent and predictable. In *AI Song*, 300 seed characters supported the production of 10,461 glyphs. FontCopilot makes local structural decisions scalable: editable component-level choices accumulate into a complete Chinese typeface.

Acknowledgments

This work was supported by the Provincial Key Research and Development Plan of Zhejiang Province under Grant No. 2026LDC01023(XC) and by the National Natural Science Foundation of China under Grant No. 62577048.

References

- [1] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *AI magazine* 35, 4 (2014), 105–120.
- [2] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.
- [3] Yining Cao, Yiyi Huang, Anh Truong, Hijung Valentina Shin, and Haijun Xia. 2025. Compositional Structures as Substrates for Human-AI Co-creation Environment: A Design Approach and A Case Study. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. ACM, 1–25. doi:10.1145/3706598.3713401
- [4] Jonas Frich, Michael Mose Biskjaer, and Peter Dalsgaard. 2018. Twenty years of creativity research in human-computer interaction: Current state and future directions. In *Proceedings of the 2018 Designing Interactive Systems Conference*. 1235–1257.
- [5] Haibin He, Xinyuan Chen, Chaoyue Wang, Juhua Liu, Bo Du, Dacheng Tao, and Yu Qiao. 2022. Diff-Font: Diffusion Model for Robust One-Shot Font Generation. *arXiv preprint arXiv:2212.05895* (2022).
- [6] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems* 30 (2017).
- [7] Yue Jiang, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. 2019. Sefont: Structure-guided chinese font generation via deep stacked networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 4015–4022.
- [8] Koichi Kamichi. 2003. KAGE-An Automatic Glyph Generating Engine For Large Character Code Set. In *Glyph and Typesetting Workshop*. Kyoto University.
- [9] Koichi Kamichi. 2011. Producing a Large Kanji Characters Set Font Using Web Collaboration Service. *IPSJ SIG Technical Report* 2011, 4 (2011), 1–8.
- [10] Bettina Laugwitz, Theo Held, and Martin Schrepp. 2008. Construction and Evaluation of a User Experience Questionnaire. *USAB 2008* 5298, 63–76. doi:10.1007/978-3-540-89350-9_6
- [11] Zhouhui Lian and Yichen Gao. 2022. CVFont: Synthesizing Chinese Vector Fonts via Deep Layout Inferring. *Computer Graphics Forum* 41, 6 (June 2022), 212–225. doi:10.1111/cgf.14580
- [12] Yitian Liu and Zhouhui Lian. 2023. FontTransformer: Few-shot High-resolution Chinese Glyph Image Synthesis via Stacked Transformers. *Pattern Recognition* (2023), 109593.
- [13] Ying-Tian Liu, Zhifei Zhang, Yuan-Chen Guo, Matthew Fisher, Zhaowen Wang, and Song-Hai Zhang. 2023. DualVector: Unsupervised Vector Font Synthesis with Dual-Part Representation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 14193–14202. doi:10.1109/cvpr52729.2023.01364
- [14] Pengyuan Lyu, Xiang Bai, Cong Yao, Zhen Zhu, Tengting Huang, and Wenyu Liu. 2017. Auto-encoder guided GAN for Chinese calligraphy synthesis. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*,

- Vol. 1. IEEE, 1095–1100.
- [15] Jiaju Ma, Chau Vu, Asya Lyubavina, Catherine Liu, and Jingyi Li. 2025. Computational Scaffolding of Composition, Value, and Color for Disciplined Drawing. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology (UIST '25)*. ACM, 1–15. doi:10.1145/3746059.3747605
 - [16] Justin Matejka, Michael Glueck, Erin Bradner, Ali Hashemi, Tovi Grossman, and George Fitzmaurice. 2018. Dream lens: Exploration and visualization of large-scale generative design datasets. In *Proceedings of the 2018 CHI conference on human factors in computing systems*. 1–12.
 - [17] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2015. Designscape: Design with interactive layout suggestions. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 1221–1224.
 - [18] Song Park, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. 2021. Multiple heads are better than one: Few-shot font generation with multiple localized experts. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 13900–13909.
 - [19] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. 2019. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2337–2346.
 - [20] Jeba Rezwana and Mary Lou Maher. 2023. Designing creative AI partners with COFI: A framework for modeling interaction in human-AI co-creative systems. *ACM Transactions on Computer-Human Interaction* 30, 5 (2023), 1–28.
 - [21] Evan F Risko and Sam J Gilbert. 2016. Cognitive offloading. *Trends in cognitive sciences* 20, 9 (2016), 676–688.
 - [22] Martin Schrepp, Jörg Thomaschewski, and Andreas Hinderks. 2017. Construction of a benchmark for the user experience questionnaire (UEQ). (2017).
 - [23] Ben Shneiderman. 2020. Human-centered artificial intelligence: Reliable, safe & trustworthy. *International Journal of Human-Computer Interaction* 36, 6 (2020), 495–504.
 - [24] Ben Shneiderman. 2022. *Human-centered AI*. Oxford University Press.
 - [25] Sangho Suh, Meng Chen, Bryan Min, Toby Jia-Jun Li, and Haijun Xia. 2024. Luminate: Structured Generation and Exploration of Design Space with Large Language Models for Human-AI Co-Creation. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*. ACM, 1–26. doi:10.1145/3613904.3642400
 - [26] Hideyuki Takagi. 2002. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proc. IEEE* 89, 9 (2002), 1275–1296.
 - [27] Yuki Tatsukawa, I-Chao Shen, Mustafa Doga Dogan, Anran Qi, Yuki Koyama, Ariel Shamir, and Takeo Igarashi. 2025. FontCraft: Multimodal Font Design Using Interactive Bayesian Optimization. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. ACM, 1–14. doi:10.1145/3706598.3713863
 - [28] Yuchen Tian. 2017. zi2zi: Master chinese calligraphy with conditional adversarial networks. *Internet* <https://github.com/kaonashi-tyc/zi2zi> 3 (2017), 2.
 - [29] Yizhi Wang and Zhouhui Lian. 2021. DeepVecFont: Synthesizing high-quality vector fonts via dual-modality learning. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–15.
 - [30] Yuqing Wang, Yizhi Wang, Longhui Yu, Yuesheng Zhu, and Zhouhui Lian. 2023. DeepVecFont-v2: Exploiting Transformers to Synthesize Vector Fonts with Higher Quality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 18320–18328.
 - [31] Jeremy Warner, Kyu Won Kim, and Bjoern Hartmann. 2023. Interactive Flexible Style Transfer for Vector Graphics. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*. ACM, 1–14. doi:10.1145/3586183.3606751
 - [32] Justin D. Weisz, Jessica He, Michael Muller, Gabriela Hofer, Rachel Miles, and Werner Geyer. 2024. Design Principles for Generative AI Applications. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*. ACM, 1–22. doi:10.1145/3613904.3642466
 - [33] Paul Yiu Chung Wong and Siu Chi Hsu. 1995. Designing Chinese typeface using components. In *Proceedings Nineteenth Annual International Computer Software and Applications Conference (COMPSAC'95)*. IEEE, 416–421.
 - [34] Zeqing Xia, Bojun Xiong, and Zhouhui Lian. 2023. VecFontSDF: Learning to Reconstruct and Synthesize High-Quality Vector Fonts via Signed Distance Functions. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 1848–1857. doi:10.1109/cvpr52729.2023.00184
 - [35] Xi Yang, Jie Zhang, Han Fang, Chang Liu, Zehua Ma, Weiming Zhang, and Nenghai Yu. 2023. AutoStegaFont: Synthesizing Vector Fonts for Hiding Information in Documents. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 3 (June 2023), 3198–3205. doi:10.1609/aaai.v37i3.25425
 - [36] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 586–595.

A Exploratory Interview Study Details

This appendix reports the participant demographics for the exploratory interview study (Table 9) and the semi-structured interview protocol used to examine current workflows, production bottlenecks, and expectations for technical support.

A.1 Interview Protocol

The interviews were guided by the following questions, with open-ended follow-ups based on each participant’s responses:

- (1) **Professional Background:** How long have you worked in font design? Do you have cross-disciplinary experience, such as moving from graphic design to type design or from designing other writing systems to Chinese?
- (2) **Current Workflow:** What is your typical workflow for Chinese font creation?
 - Which professional software or tools do you use?
 - Which characters do you usually design first?
 - Have you assembled a complete character set? If so, how long did it take?
 - Which stages would you most like technology to support?
- (3) **Prior Tool Experience and Expectations:** Have you used AI-assisted creation tools, particularly for font design? What capabilities would you expect from an assisted font-design tool?

B Dataset

B.1 Radical and Component

Most Chinese characters can be decomposed into components. To clarify this term, we first distinguish components from Chinese character radicals. According to the *Table of Indexing Chinese Character Components*², there are 301 standard radicals.

Figure 12 shows characters containing the radical "女". Radicals are primarily used for character lookup and identify only one part of a character. For example, "夭", the right-hand part of "妖", is not a radical. We therefore use *component* to refer to each separable part of a character: both "女" and "夭" are components of "妖".

The dataset contains approximately 11,000 components, excluding nested definitions.

As shown in Figure 13, each record contains an ID, the character itself, its Unicode codepoint, radical information, a layout label, and the decomposed components.

B.2 Accessibility

We release a public subset of the dataset through the project repository.³ Access to the complete dataset is available upon request and subject to authorization.

C Technical and Implementation Details

This section details the mathematical definitions of the loss functions, network hyperparameters, and the specific layout strategies utilized in the Component Affine transformation Regressor (CAR).

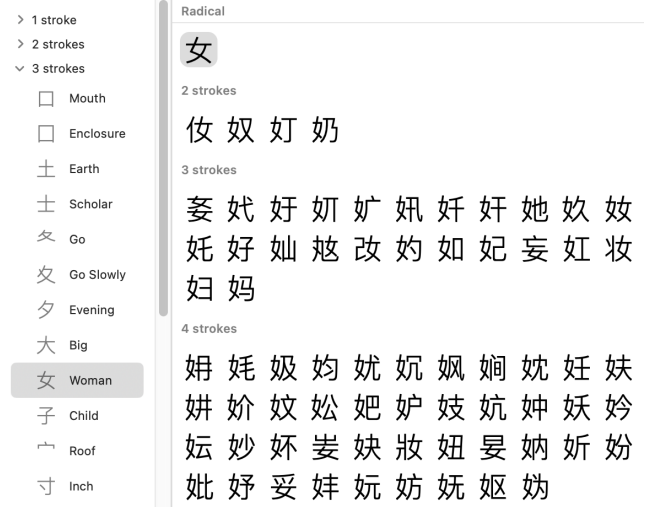


Figure 12: Chinese characters containing the radical "女", grouped by the stroke count of the remaining component.

Search Hanzi Unicode Radical Layout Box0 Box1 Box2 Apply							
ID	Hanzi	Unicode	Radical	Layout	Box0	Box1	Box2
0	𡚦	3400	𡚦	𡚦	𡚦		
1	𡚧	3401	𡚧	𡚧	𡚧	人	
2	𡚨	3402	𡚨	𡚨	七	𡚨	
3	𡚩	3403	𡚩	𡚩	𡚩		
4	𡚪	3404	𡚪	𡚪	𡚪		
5	𡚫	3405	𡚫	𡚫	𡚫		
6	𡚬	3406	𡚬	𡚬	𡚬		
7	𡚭	3407	𡚭	𡚭	九	乙	
8	𡚮	3408	𡚮	𡚮	入	乙	
9	𡚯	3409	𡚯	𡚯	丁	乙	
10	𡚰	340a	𡚰	𡚰	士	乙	
11	𡚱	340b	𡚱	𡚱	士	乙	

Figure 13: Dataset records. Each sample contains an ID, Hanzi, Unicode codepoint, radical, layout, and decomposed components.

C.1 Loss Functions

First, the regular pixel-wise loss is computed to measure the difference between the synthesized character S and ground truth C :

$$L_{pixel} = D(S, C) \quad (1)$$

where D represents any distance function. In our case, we simply use the L1 distance.

The affine transformation matrix is initialized with an identity matrix such that the components may have a large overlap at the beginning of the training loop. Therefore, the pixel loss L_{pixel} contributes significantly at the beginning. However, as the network converges, the overlap region becomes smaller, and so does the pixel loss. As a consequence, the network stops pushing the two components away from each other. To solve this issue and make each component move in the correct direction, we added the overlap loss:

$$L_{overlap} = \frac{\sum_{x,y} \min(\max(S(x, y) - 1, 0), 1)}{\sum_{x,y} S(x, y)} \quad (2)$$

²https://en.wikipedia.org/wiki/Table_of_Indexing_Chinese_Character_Components

³<https://github.com/AnCoSONG/FontCopilot>

Table 9: Demographic information and prior tool experience of the exploratory interview participants.

ID	Gender	Age Group	Occupation	Experience	Software	Prior AI-Tool Exp.
P1	Male	18~25	Full-time Type Designer	4–6 years	Glyphs	Yes
P2	Female	31~40	Full-time Type Designer	> 10 years	Glyphs, In-house Software	Yes
P3	Female	31~40	Full-time Type Designer	7–10 years	Glyphs, In-house Software	No
P4	Female	31~40	Part-time Type Designer	< 1 year	Glyphs	No
P5	Male	18~25	Student	1–3 years	Glyphs, Adobe Illustrator	No
P6	Male	18~25	Full-time Type Designer	1–3 years	Glyphs	No
P7	Female	26~30	Full-time Type Designer	4–6 years	Glyphs, Adobe Illustrator	No
P8	Undisclosed	26~30	Full-time Type Designer	4–6 years	Glyphs	Yes
P9	Male	31~40	Other Designer	< 1 year	Adobe Illustrator	No
P10	Male	31~40	Full-time Type Designer	> 10 years	Glyphs	Yes

where $S(x, y)$ denotes the pixel value of S . In the two-component synthesis case, a pixel value greater than 1 means two components overlap there.

Besides, the centroid is another important attribute of Chinese font characters. A reasonable synthesized character S should share the same centroid position as the ground truth C to guarantee the balance of the synthesized character. Therefore, we also introduced a centroid loss:

$$L_{cent.} = \frac{1}{2} D\left(\frac{\Phi(S, 1, 0)}{\Phi(S, 0, 0)}, \frac{\Phi(C, 1, 0)}{\Phi(C, 0, 0)}\right) + D\left(\frac{\Phi(S, 0, 1)}{\Phi(S, 0, 0)}, \frac{\Phi(C, 0, 1)}{\Phi(C, 0, 0)}\right) \quad (3)$$

where Φ represents raw moments:

$$\Phi(I, i, j) = \sum_{x,y} I(x, y) x^i y^j \quad (4)$$

To make sure that every pixel of the synthesized character S has the same distance to the centroid as the ground truth C , we introduced the second-order central moment, *i.e.*, inertia loss:

$$L_{inertia} = D(\Psi(S), \Psi(C)) \quad (5)$$

where Ψ represents the inertia that is calculated as follows:

$$\Psi(I) = \Phi(I, 2, 0) - \frac{\Phi(I, 1, 0)^2}{\Phi(I, 0, 0)} + \Phi(I, 0, 2) - \frac{\Phi(I, 0, 1)^2}{\Phi(I, 0, 0)} \quad (6)$$

In practice, the inertia loss regulates the shape and bounding box of the synthesized character.

The final loss function is then defined as:

$$\mathcal{L} = \alpha L_{pixel} + \beta L_{overlap} + \gamma L_{cent.} + \theta L_{inertia} \quad (7)$$

The lowercase Greek characters represent the weight of each loss term. The weights used in all experiments are listed in Table 10. Note that we tested the selection of each loss weight for at least 5 times to obtain the final configurations.

Layout	αL_{pixel}	$\beta L_{overlap}$	$\gamma L_{cent.}$	$\theta L_{inertia}$
NL01	1	0 or 1	0	0
NL02	1	0 or 1	0	0
NL03	1	1	$5e-2$	$1e-8$

Table 10: Loss weights used in the experiments.

C.2 Training and Implementation Hyperparameters

For data preprocessing, character and component images were rendered using PIL. ImageFont and centered within a 256×256 resolution. For each sample, the target character and its component images are horizontally stacked into one raster image; only the component images are fed to CAR, while the target character serves solely as the supervision signal for the loss functions above.

During the training phase, all models were optimized using Stochastic Gradient Descent (SGD) with an initial learning rate of 2×10^{-3} and a momentum of 0.9. A StepLR scheduler was adopted to decay the learning rate by half every six epochs, and all models were trained for a total of 42 epochs.

To address the geometric variance across different structural layouts, the weights of the loss terms were dynamically adjusted. For stable, non-nested layouts (e.g., NL01 and NL02), only the pixel reconstruction loss and the overlap loss were enabled. For highly complex and enclosed layouts (e.g., NL03), all loss terms—including centroid and inertia constraints—were activated to strictly guide the stable convergence of the affine transformations.

C.3 Implementation Details for Different Layouts

As illustrated in the main text, we divide Chinese fonts into six layouts. For all layouts, we centralize the component images, randomly flip them, and feed them into the CAR for training.

- (1) For NL01 (Left-Right), the CAR is trained from scratch, but the feature extractor is initialized with the pretrained parameters.
- (2) For NL02 (Top-Bottom), we first attempted to train the CAR from scratch. To improve performance, an intuitive idea is to use the parameters of NL01 pre-trained models. Since NL01 characters are quite different from NL02's, the affine transformation regressor trained on NL01 is not directly suitable for NL02. To overcome this, we introduced a switcher module that is a 6×6 matrix with all positions set to 0 except for (0,4), (4,0), (2,5), and (5,2), which have values of 1. This module swaps the parameters affecting the horizontal and vertical axes in the affine transformation, allowing the NL01 affine transformation matrices to adapt to NL02 characters. With the pretrained parameters, the CAR of NL02 converges faster and gains better performance.
- (3) For NL03 (Enclosed), the base CAR failed easily. To solve this, we increased the weight of the centroid and inertia losses to make the CAR converge.
- (4) For NL04 (Left-Middle-Right) and NL05 (Top-Middle-Bottom), we adjusted our model architecture to fit this three-component font synthesis task. However, the performance of these models is far from satisfactory. To address this, we proposed an iterative invocation strategy where we used the pretrained NL01/NL02 CAR to perform the NL04/NL05 synthesis task. For example, for an NL04 character, we first invoked the NL01 pretrained CAR to synthesize the intermediate character based on the first two components. We then invoked the NL01 pretrained CAR again to synthesize the final character based on the intermediate character and the last component. Evaluation results demonstrate the effectiveness of this design.

C.4 Training Visualization

We show the training process of our model in Figure 14. As the training progresses, our method exhibits an increasingly refined synthesis of characters, approaching a higher level of fidelity to the ground truth.

C.5 Fusion Strategies

We implemented three types of fusion methods, namely Stack, AdaIN, and Cross Attention. As for Stack, we simply concatenate them together in the channel dimension. As for AdaIN, we compute the feature using the equation below:

$$\hat{f}_{R1} = \sigma(f_{R2}) \frac{f_{R1} - \mu(f_{R1})}{\sigma(f_{R1})} + \mu(f_{R2}) \quad (8)$$

where $\hat{f}_x$ refers to the fused feature, f_x represents the feature of component x , and μ and σ represent the mean and std of the feature, respectively. For cross attention fusion, the fused feature is calculated using the following equation:

$$\hat{f}_{R1} = \text{Softmax}\left(\frac{Q(f_{R1})K(f_{R2})^T}{\sqrt{d}}\right)V(f_{R2}) \quad (9)$$

where Q , K , and V represent the linear layers which output query, key, and value vectors for attention, respectively.

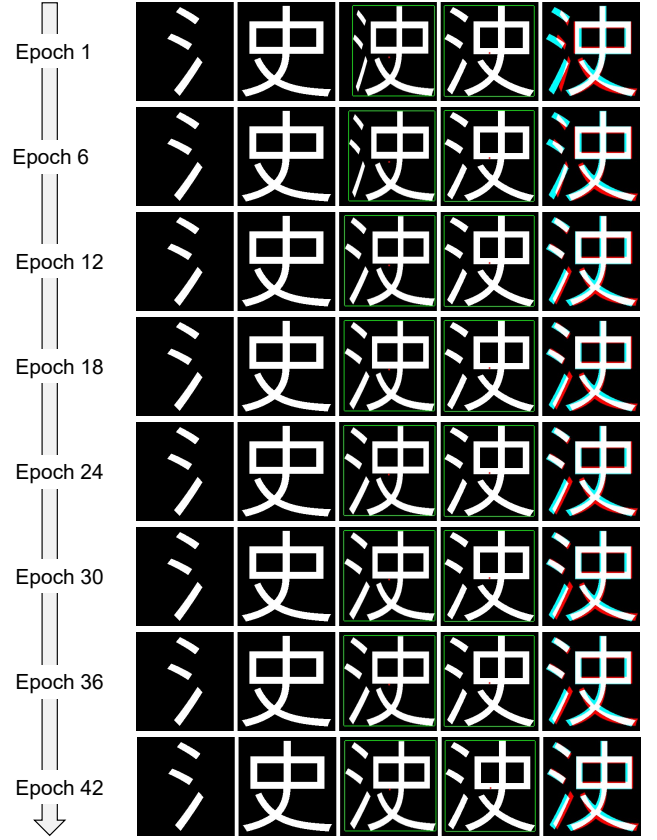


Figure 14: The visualization of the training process. For each row, from left to right, there are component 1, component 2, generated character, ground truth, and the difference between generated character and ground truth, respectively.

We additionally tested the performance of our proposed CAR using SPADE [19] as a feature fusion method. The affine parameters of SPADE are learned from the feature of another component.

As shown in Table 11, Stack performs best in all situations, especially with limited data (NL03). Cross attention performs relatively worse.

C.6 Iterative Invocation V.S. Three-Components CAR

We present some samples generated by the two-component CAR with the iterative invocation strategy and the three-component CAR to further illustrate the effectiveness of the approach, as shown in Figure 15.

Compared with the three-component CAR, the iterative invocation strategy achieved better results. However, the performance of CAR on NL04 and NL05 still requires further structural refinement in future work.

D Extended Ablation Study Analysis

This appendix provides additional qualitative observations for the ablation study and lists the licenses of the seven training fonts.

Layout	Fusion	MAE↓	RMSE↓	FID↓	LPIPS↓
NL01	Stack	0.1312	0.3540	12.7156	0.2019
NL01	AdaIN	0.1385	0.3641	12.6574	0.2085
NL01	Attention	0.1426	0.3699	13.0480	0.2126
NL01	SPADE	0.1361	0.3607	12.7894	0.2063
NL01	None	0.1676	0.4022	13.6127	0.2332
NL02	Stack	0.1562	0.3880	13.0982	0.2210
NL02	AdaIN	0.1654	0.3998	13.0116	0.2311
NL02	Attention	0.1658	0.4004	13.0685	0.2335
NL02	SPADE	0.1611	0.3943	13.2121	0.2263
NL02	None	0.2630	0.5079	17.8102	0.3234
NL03*	Stack	0.1891	0.4264	33.4290	0.2709
NL03*	AdaIN	0.2040	0.4437	33.7215	0.2880
NL03*	Attention	0.2113	0.4518	33.9402	0.2965
NL03*	SPADE	0.2182	0.4598	36.1949	0.3300
NL03*	None	0.2687	0.5113	59.3721	0.4039
NL03**	Stack	0.1446	0.3714	35.3016	0.2140
NL03**	AdaIN	0.1475	0.3758	55.1334	0.2409
NL03**	Attention	0.1668	0.4000	38.1286	0.2382
NL03**	SPADE	0.1687	0.4023	42.2689	0.2403
NL03**	None	0.1619	0.3947	56.1288	0.2563

Table 11: The quantitative results of different fusion methods. NL03* denotes NL03-0,1 and NL03** denotes NL03-2,3,4,5.

D.1 Extended Analysis of Losses

As summarized in Table 2 and Figure 6 of the main text, adding the centroid, inertia, and overlap losses reduces the reported reconstruction and perceptual errors. In the qualitative examples, the model trained only with pixel loss often shows uneven component placement. The overlap loss reduces component crowding and intersections, while the inertia and centroid losses improve spatial balance and bounding-box alignment.

D.2 Extended Analysis of Feature Fusion

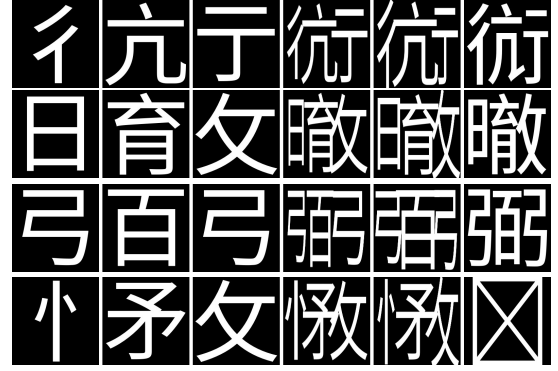
Table 3 shows that models without feature fusion sometimes predict nearly constant affine transformations for specific components and do not account for contextual scaling. Feature fusion allows the regressor to use the spatial relationship and relative scale between adjacent components when predicting the transformation matrices.

D.3 Adaptation for NL02 Layouts

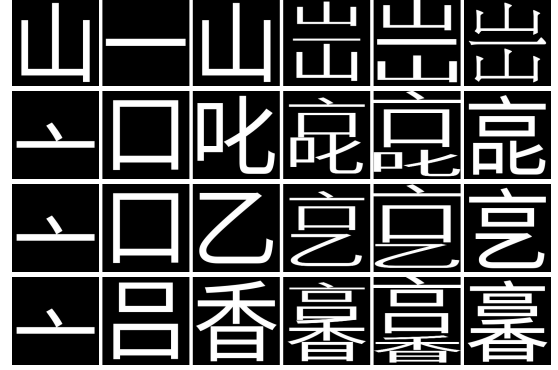
For Top-Bottom (NL02) layouts (Table 4), training the model from scratch converges more slowly and yields higher reconstruction errors. Introducing the switcher module transfers parameters from the pretrained Left-Right (NL01) model; the resulting model converges faster and obtains lower reconstruction errors, indicating that parameters learned for NL01 can support NL02 training.

D.4 Stability of Iterative Invocation

For Left-Middle-Right (NL04) and Top-Middle-Bottom (NL05) layouts (Table 5), directly regressing three components produces higher errors in the reported setting. The iterative invocation strategy



(a) NL04



(b) NL05

Figure 15: The NL04(15a) and NL05(15b) samples are generated by the two-component CAR with iterative invocation and the three-component CAR. For each row, there are component 1, component 2, component 3, samples generated by two-component CAR with iterative invocation strategy, three-component CAR, and ground truth, respectively. Ground truth displaying a ".notdef" character means the character is not supported by the font.

reuses the two-component CAR step by step and obtains lower errors than the three-component CAR without requiring an additional three-component architecture.

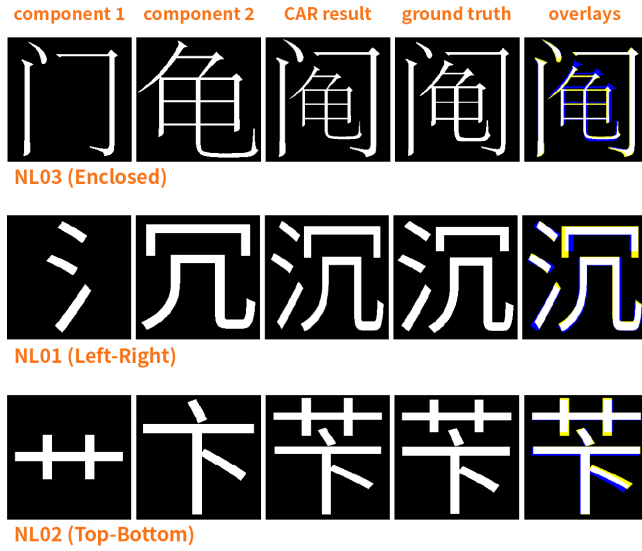
D.5 Training-Font Selection and Licenses

The seven fonts used in our experiments were distributed under licenses permitting the reported research use. *SourceHanSans*, *SmileySans*, *YouAiYuanTi*, and *WenDingKaiTi* are licensed under the SIL Open Font License (OFL) 1.1; *BabelStone* is released under the Arphic Public License; and *AlibabaPuHuiTi* and *OPPOSansR* are provided under their respective corporate free-to-use licenses.

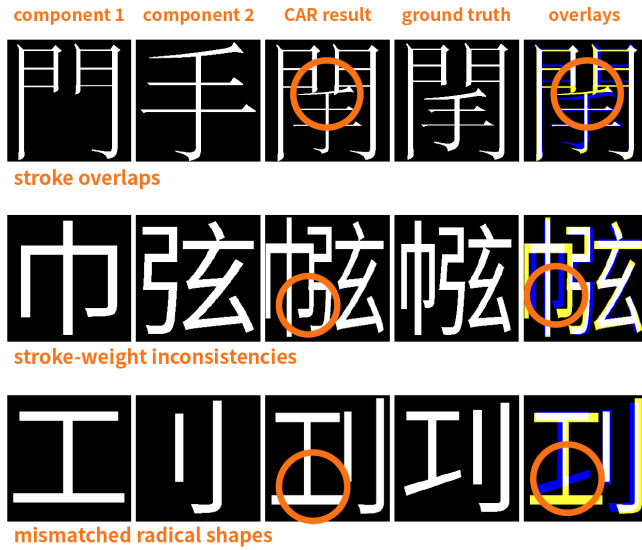
E Representative Success and Failure Cases

Successful Cases. Figure 16a shows representative successful results for the three two-component layouts. Across enclosed, left-right, and top-bottom structures, CAR places and scales the input components to closely match the target glyphs. In the overlays,

most stroke regions coincide, while the remaining differences are confined to narrow contour boundaries.



(a) Successful cases.



(b) Failure cases.

Figure 16: Representative CAR results. (a) Successful cases across NL03, NL01, and NL02, with white overlay regions indicating agreement. (b) Failure cases showing residual stroke overlap, local stroke-weight inconsistency, and mismatched radical shape; orange circles mark the affected regions.

Failure Cases. Figure 16b presents failure cases in which component-wise affine transformation does not fully reproduce the intended glyph geometry. Non-uniform scaling may introduce local differences in stroke weight, while transforming a radical as a whole

cannot always capture the fine-grained shape adjustments required by a particular character. Our overlap loss substantially reduces implausible intersections between components; however, residual stroke overlaps or imperfect junctions may still occur under large transformations or in densely structured glyphs. Despite these residual geometric discrepancies, CAR preserves component structure and vector contours, enabling designers to correct the generated glyphs through component repositioning, rescaling, or fine-grained contour editing.