GlyphShield: Document Watermarking for the Physical World via Vector Typeface Synthesis

Nan Sun[1], **Yuxing Lu**[2], **Han Fang**[3], * **Hefei Ling**[1], **Sijing Xie**[1], **LuYu Yuan**[1], **Chengxin Zhao**[1]

[1]School of Computer Science and Technology, Huazhong University of Science and Technology
[2]Peking University
[3]National University of Singapore

Abstract

Document protection has become a critical issue for preventing unauthorized copying, distribution, and tampering. Document encryption is a proven solution, but it is not resistant to attacks from the physical world such as screenshots, printing and photographing. A common document protection technique is font-based watermarking, which embeds imperceptible information by using sets of visually similar glyphs to encode traceable data. However, due to the non-differentiable rendering process of vector fonts, these methods often rely on time-consuming and laborious manual design. To address this challenge, we present **GlyphShield**, an innovative end-to-end vector font watermarking framework. We resolve the non-differentiability challenge by simulating differentiable rasterization through the computation of Signed Distance Field (SDF) for Bézier curves in vector fonts. Besides, to handle complex vector font structures, a novel dual-branch vector encoder is employed to ensure high-quality font synthesis. Extensive experiments demonstrate that our approach ensures more natural and smoother message embedding while ensuring robustness against noise attacks in diverse scenarios. Additionally, our framework demonstrates strong generalization across various font styles and languages.

Introduction

Digital watermarking, as an important security tool, aims to convey and store information without arousing suspicion by embedding watermark information in various media such as image (Zhu et al. 2018; Tancik, Mildenhall, and Ng 2019; Lan et al. 2023), audio (Pavlovic et al. 2021; Liu et al. 2022; Chen et al. 2023), video (Wang and Pearmain 2006; Chang et al. 2022; Zhang et al. 2023) or document (Qi et al. 2019; Yang et al. 2023; Sun et al. 2024). It plays a key role in information hiding, privacy protection, traceability tracking, copyright protection, and data integrity verification.

Text documents are the most widely used digital media, making the embedding of invisible watermarks in them highly valuable. However, existing solutions exhibit inadequacies for documents that demand both **physical-world robustness** and **semantic precision**. On the one hand, encryption algorithms (Rivest, Shamir, and Adleman 1978; Koblitz 1987) protect against unauthorized digital access but are defenseless against physical-world attacks such as screenshots or printing. On the other hand, semantic-based watermarking (Fang, Jaggi, and Argyraki 2017; Kirchenbauer et al. 2023; Yoo et al. 2023), while robust across different media, fundamentally alters textual content. This makes it unsuitable for applications where semantic integrity is paramount, such as legal contracts and identification documents.

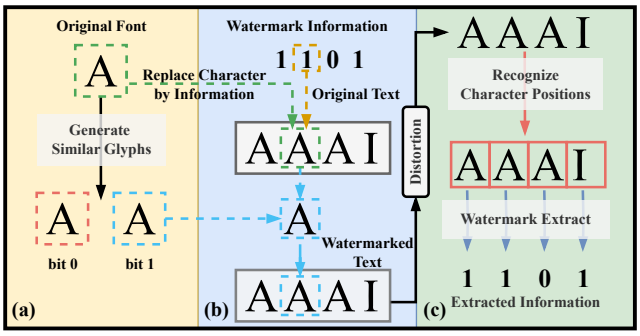

Figure 1: General process of text watermarking based on font watermarking. (a) synthesizing watermarked fonts, (b) embedding watermark information into text, and (c) extracting watermark information from distorted text images.

This gap highlights the need for a different paradigm: watermarking the visual representation of text, rather than its semantic content. One such family of methods is font-based watermarking, which embeds information by constructing a set of visually similar fonts. As illustrated in Figure 1, the typical pipeline for these methods (Xiao, Zhang, and Zheng 2018; Campbell and Kautz 2014; Qi et al. 2019; Yang et al. 2023) consists of three main stages: (a) Font Generation: For each character, a set of visually similar glyphs is generated, where each unique glyph encodes a data bit (e.g., '0' or '1'). (b) Watermark Embedding: During this stage, characters in the original document are systematically replaced with their corresponding glyph variants from the generated set to encode the complete watermark. (c) Watermark Extraction: The document image is first processed to locate character positions. Then, the specific glyph used for each character is recognized, and the associated information is decoded sequentially to reassemble the original message.

However, above methods treat watermarked font synthe-

*Corresponding Author (lhefei@hust.edu.cn)

sis (Encoder) and information extraction (Decoder) as separate stages. This separation arises because the watermarked font is inherently a vector-based representation, defined by a series of coordinate points, whereas information extraction operates in the image domain. Since rendering vector fonts into character images is non-differentiable, the Decoder's gradient cannot be back-propagated to the Encoder, preventing end-to-end optimization. Therefore, achieving joint optimization of the Encoder and Decoder is essential for automated watermark font synthesis.

To address the problem, we introduce **GlyphShield**, a novel end-to-end vector font watermarking framework. At the core of our approach is the joint optimization of the encoder and decoder, achieved by computing the Signed Distance Field (SDF) of vector characters to approximate a differentiable rendering process. The main contributions of this work are as follows:

- We present the first end-to-end vector font watermarking framework, enabling robust and imperceptible generic text watermarking through font-based embedding. By computing the SDF of vector font, we model differentiable rasterization to enable joint optimization of the font synthesis encoder and the message decoder.
- Due to the complex nature of vector font structure, we propose a dual-branch network to predict small glyph perturbations by capturing local and global features of vector fonts to generate high-quality watermarked fonts.
- Extensive experiments demonstrate that our framework outperforms previous SOTA models, ensuring high-quality watermarked font synthesis with robustness to various distortions. Additionally, our model can embed watermark information in texts of different languages and styles, showcasing wide applicability.

Related Work

Image Watermarking. In recent years, there has been a proliferation of research on image watermarking. HiDDeN (Zhu et al. 2018) employs an autoencoder-like DNN architecture to jointly train encoder and decoder for watermark embedding and extraction. StegaStamp (Tancik, Mildenhall, and Ng 2019) presents a robust and reliable image watermarking framework using the U-Net architecture, aiming to enhance watermark robustness, particularly in scenarios involving real-world printing and photography. Xu et al. (song Xu et al. 2022) achieved high-capacity image watermarking by leveraging the properties of Invertible Neural Networks. However, the sparse texture of document images makes it challenging to adapt conventional image watermarking methods.

Font Watermarking. FontCode (Xiao, Zhang, and Zheng 2018) initializes a set of randomly perturbed fonts from the font manifold (Campbell and Kautz 2014), followed by manual screening to construct similarly shaped fonts. A CNN network is then trained for each character to extract embedded information. Qi et al. (Qi et al. 2019) propose creating perturbed glyphs by adjusting strokes and use a template-matching algorithm to extract information by calculating the similarity between the current glyph and the perturbed ones. AutoStegaFont (Yang et al. 2023) adopts a two-stage training strategy: it first generates encoded glyph images, then individually vectorizes and optimizes each watermarked character into its font representation. However, all these methods design the font synthesis and information extraction processes separately, leading to poor collaboration between the encoder and decoder. This separation degrades the visual quality of synthesized fonts and reduces the robustness of information decoding. Additionally, these methods require individual designs for each character, making them time-consuming and inefficient for texts with many glyphs, such as Chinese and Japanese.

Font Synthesis. Deep learning has recently advanced font generation (Park et al. 2021; Tang et al. 2022; Pan et al. 2023; Wang et al. 2023), yet most methods focus on font images and cannot produce genuine vector fonts. For this, several studies have delved into the synthesis of high-fidelity vector fonts. DeepVecFont (Wang and Lian 2021) predicts SVG instructions with LSTMs and uses differentiable rasterization via diffvg (Li et al. 2020) to improve visual fidelity, but diffvg remains slow and struggles to balance gradient quality with rasterization accuracy. Consequently, recent works (Reddy et al. 2021; Xia, Xiong, and Lian 2023; Liu et al. 2023) favor Signed Distance Fields (SDF) on simple geometric primitives for efficient differentiable rasterization. Our approach adapts this by computing the SDF directly on a font's constituent Bézier curves. This enables gradient propagation from the information decoder back to the font generator, allowing for true end-to-end optimization of the watermarking process.

Methodology

Preliminary

Before introducing our method, we first describe the data structure of vector fonts. Unlike pixel-based character images, vector fonts represent shapes with Bézier curves. A vector character G is composed of n closed contours $G = \{C_i\}^n$, where each contour C_i contains m_i control points $C_i = \{P_j\}^{m_i}$. Each control point $P = (x, y)$ includes its coordinates and an identifier s indicating whether it lies on the curve. Given three control points P_0, P_1, P_2, then any point $P(t)$ on the Bézier curve B can be expressed as:

$$B : P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2,\ t \in [0,1] \quad (1)$$

When P_0, P_1 and P_2 are collinear, the curve degenerates into a straight line. In this paper, we represent all straight lines as quadratic Bézier curves by inserting an additional control point $P_1 = (P_0 + P_2)/2$ between two points. This unifies the closed contour C, containing both straight and curved lines, into a contour composed solely of quadratic Bézier curves. Thus, for any point P_j in the closed contour, P_j is an off-curve point if j is even, otherwise it is an on-curve point. Notably, the first point in C serves as the P_0 control point of the first curve and also the P_2 control point of the last curve.

In the actual training process, we concatenate all closed contours $C_i \in \mathbb{R}^{m_i \times 2}$ of a character G. The final shape of our input data is $G = \{C_1 \circ C_2 ... C_n\} \in \mathbb{R}^{\hat{m} \times 2}$, where $\hat{m} = \sum_{i=1}^{n} m_i$.

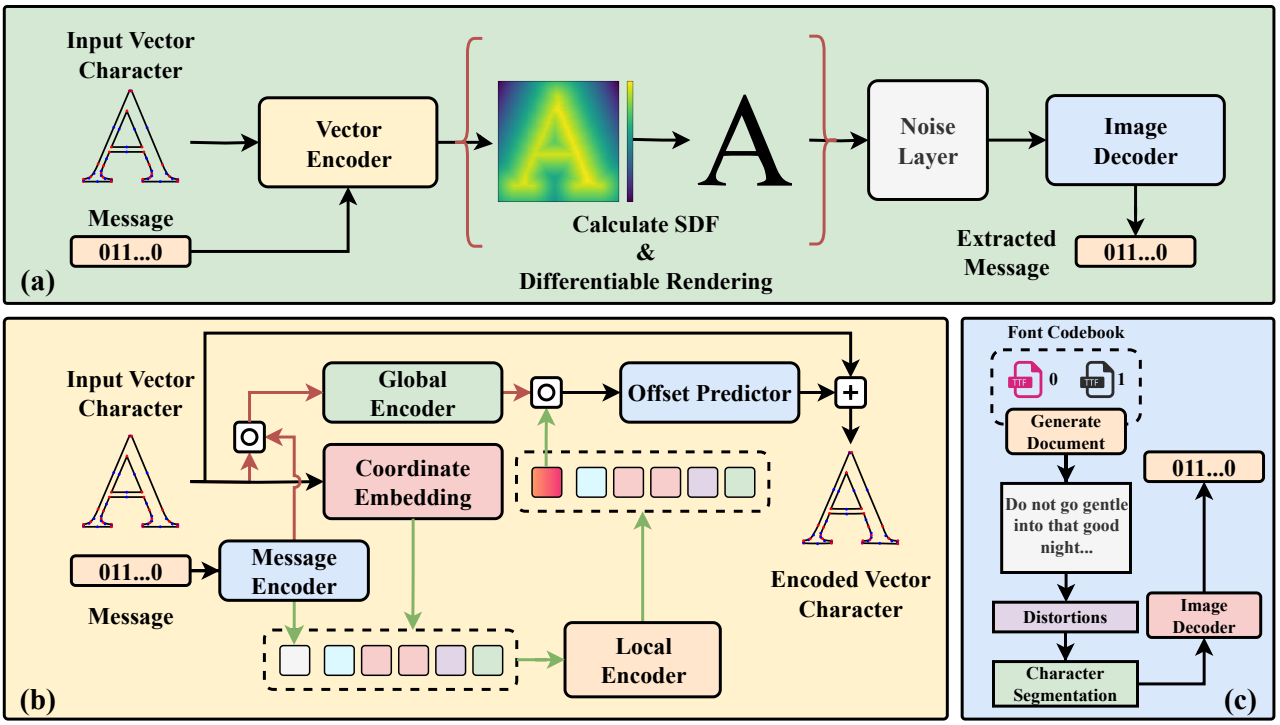

Figure 2: Architecture of the proposed model. The pipeline of our approach is illustrated in (a), highlighting the additional process of differentiable rasterization (denoted by red curly brackets) compared to the general image watermarking framework. Figure (b) details the structure of the vector encoder. Figure (c) demonstrates how the fonts synthesized by our method are used to watermark a document and subsequently extract the embedded watermark. In addition, "$\circ$" denotes feature concatenation, while "+" represents element-wise addition.

Overview

As shown in Figure 2 (a), our pipeline first employs an Vector Encoder to generate watermarked vector characters. These characters are then differentiably rasterized (highlighted in red brackets), passed through a Noise Layer, and finally decoded to recover the embedded message.

The Vector Encoder is responsible for embedding the n-bit binary watermark information $M \in \{0,1\}^n$ into the vector character G, resulting in $\hat{G}$ with the embedded information. Subsequently, we compute the Signed Distance Field $S \in \mathbb{R}^{1\times h\times w}$ of $\hat{G}$ and render S into the image character $\hat{I} \in \mathbb{R}^{1\times h\times w}$ using a differentiable rendering function $f(\cdot)$. The h, w represent the height and width of the image. The image $\hat{I}$ is then passed through a Noise Layer to simulate potential distortions, producing the distorted image $\tilde{I}$. Finally, the Image Decoder extracts the watermark information $\hat{M}$ from the distorted image.

Watermarked Font Synthesis

Figure 2(b) illustrates the Vector Encoder for watermark font synthesis. The dual-branch design extracts complementary global and local features, whose combined outputs predict coordinate offsets of vector characters to embed watermarks. This architecture delivers both high-quality synthesis and robust decoding, as validated in the ablation study.

Message Encoder. The message encoder consists of four layers of MLP, each using Batch Normalization and ReLU activation functions. It is responsible for encoding the raw watermark information into intermediate features $z_m \in \mathbb{R}^d$, where d is the feature dimension (default is 64). These encoded features are then provided to the two branches respectively.

Global Branch. Initially, the input $G \in \mathbb{R}^{\hat{m}\times 2}$ is reshaped into $\tilde{G} \in \mathbb{R}^{2\hat{m}}$. This reshaped input is then fed into a Linear Layer to obtain the intermediate feature $z_c \in \mathbb{R}^{\hat{d}}$, where $\hat{d}$ is 128. Finally, z_c is concatenated with z_m and fed into the Global Encoder module (another four-layer MLP) to produce the fused feature $\hat{z}_c \in \mathbb{R}^{\hat{d}}$.

Local Branch. In the Data Structure section, we mentioned that the vector character G is directly stitched together from multiple closed contours C. However, due to the varying lengths of these closed contours and the lack of separating identifiers, the Global branch can only capture global information, lacking the ability to model finer details. To address this limitation, we employ the Local branch to capture the local relationships within a character.

First, the input G is mapped into feature tokens by the Coordinate Embedding. Assume there are at most n closed contours in a character, and each contour has at most m control points. for the j-th point $P_{ij} \in \mathbb{R}^2$ in the i-th contour, we project it onto a d-dimensional vector e_{ij}. The indices i and j are converted into two one-hot vectors, $\delta_i \in \{0,1\}^n$

and $\delta_j \in \{0,1\}^m$. This can be expressed by the following equation:

$$e_{ij} = W_\alpha P_{ij} + W_\beta \delta_i + W_\gamma \delta_j \tag{2}$$

where $W_\alpha \in \mathbb{R}^{d\times 2}, W_\beta \in \mathbb{R}^{d\times n}$, and $W_\gamma \in \mathbb{R}^{d\times m}$ are the three learnable weight matrices. Assuming we have a total of k control points, this can be expressed as:

$$z_e = \{e_{1,1} \circ e_{1,2} \circ ...e_{i,j}...\} \in \mathbb{R}^{k\times d} \tag{3}$$

After that, we splice z_s as a separate token to z_e and encode it using Global Encoder (a standard 2-layer Transformer encoder(Vaswani et al. 2017) architecture). The Transformer's multi-head attention is set to 8, and the feed-forward dimension is 128, utilizing the GeLU activation function. Finally, we take only the first token after the Global Encoder as the encoded feature z_t of this branch.

Offset Predictor. Finally, we concatenate $\hat{z_c}$ and z_t as the input to the Offset Predictor $\mathcal{P}$, which consists of four-layer MLP, to predict the offsets of each coordinate. The final vector font $\hat{G}$ in which we embed the watermark can be represented as:

$$\hat{G} = G + \psi(\mathcal{P}(z_t \circ \hat{z_c})) \cdot \alpha \tag{4}$$

where $\psi(\cdot)$ is the Tanh function and α is a hyperparameter that regulates the embedding strength, with the value set to 0.004 during training.

Differential Rasterization

In this section, we describe the entire specific process of differentiable rasterization. The core idea behind our implementation lies in transforming the vector rendering process into the computation of a set of distance functions (SDF), which facilitate the normal propagation of gradients.

According to the definition, the signed distance field is the distance from any point $P(x, y)$ on the image to the nearest curve $P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$. So we first find the optimal parameter $\hat{t}$ and then calculate its distance and direction.

First, let us compute the optimal parameter $\hat{t}$. For the vector character G, assume there exist n closed contours. Each closed contour consists of m quadratic Bézier curves. For the j-th curve $P_{ij}(t)$ of the i-th closed contour, we can compute the optimal parameter $\hat{t}$ as follows:

$$\hat{t} = \arg\min_t \|P_{ij}(t) - P(x, y)\|_2^2 \tag{5}$$

The distance $D_{ij}(x, y)$ from any point $P(x, y)$ to the curve $P_{ij}(t)$ can be expressed as a quadratic function with respect to the parameter t:

$$\begin{aligned} D_{ij}(x, y) &= \|P_{ij}(t) - P(x, y)\|_2^2 \\ &= (x(t) - x)^2 + (y(t) - y)^2 = f^4(t) \end{aligned} \tag{6}$$

We use as an optimal parameter the value of t at the point where the value of its first-order derivative is 0 by finding:

$$\frac{\mathrm{d}}{\mathrm{d}t} f^4(t) = 0 \tag{7}$$

This can be easily solved by cubic equation root formula, and thus we can get $\hat{t}$ as nearest point parameter.

After obtaining the optimal parameter $\hat{t}$, we also need to determine whether $P(x, y)$ is inside or outside the curve to obtain the sign. We compute the sign using its tangent vector $v(\hat{t})$ and the cross product of $\overrightarrow{P_{ij}(\hat{t})P(x, y)}$. This can be expressed as:

$$H_{ij}(x, y) = \frac{\overrightarrow{P_{ij}(\hat{t})P(x, y)} \times v(\hat{t})}{\left|\overrightarrow{P_{ij}(\hat{t})P(x, y)} \times v(\hat{t})\right|} \tag{8}$$

where $v(\hat{t})$ can be expressed as:

$$v(\hat{t}) = -2(1-t)P_0 + 2(1-2t)P1 + 2tP_2 \tag{9}$$

After calculating the distance and sign, we first calculate the SDF value for each closed contour, which can be expressed by the following equation:

$$\hat{S}_i(x, y) = \max_{1\le j\le m} \{D_{ij}(x, y) \cdot H_{ij}(x, y)\} \tag{10}$$

After that, we can get the SDF of the whole character by taking the concatenated set of values of all closed contours:

$$\hat{S}(x, y) = \min_{1\le i\le n} \left\{\hat{S}_i(x, y)\right\} \tag{11}$$

In order to obtain a workable image, we also need a differentiable rendering function $f(\cdot)$ to restrict $S \in [-\infty, \infty]$ to $[0, 1]$.To obtain a more learnable result in order to avoid discontinuity of its first derivative, we use Reddy's method (Reddy et al. 2021) to obtain the final result:

$$\hat{I}(x, y) = \begin{cases} 1 & \gamma < \hat{S}(x, y) \\ \frac{1}{2} - \frac{1}{4}((\frac{\hat{S}(x,y)}{\gamma})^3 - 3(\frac{\hat{S}(x,y)}{\gamma})) & -\gamma \le \hat{S}(x, y) \le \gamma \\ 0 & \hat{S}(x, y) < -\gamma \end{cases} \tag{12}$$

where γ is a hyperparameter used to control the degree of rendering smoothness, defaulting to 0.125.

Noise Layer and Image Decoder

Noise Layer. During training, we use the Noise Layer to simulate potential distortions of the watermarked document to enhance the robustness of the model. These distortions include resizing, translation, scaling, rotation, perspective transformation, blurring, Gaussian noise, and color manipulation. To ensure a fair comparison with the baseline, our Noise Layer settings are kept consistent with those used in baseline (Yang et al. 2023).

Image Decoder. The Image Decoder is responsible for extracting the embedded watermark information from the distorted image $\tilde{I}$. As the focus of this paper is not on the decoder's architecture, we utilize the Decoder structure from MBRS (Jia, Fang, and Zhang 2021).

Loss Function

Our loss function consists of two components: the visual loss function $\mathcal{L}_q$, which ensures high-quality font synthesis, and the message loss function $\mathcal{L}_m$, which ensures decoding accuracy.

Visual Loss. It consists of three components that constrain the rendered image $\hat{I}$, the computed SDF value $\hat{S}$, and

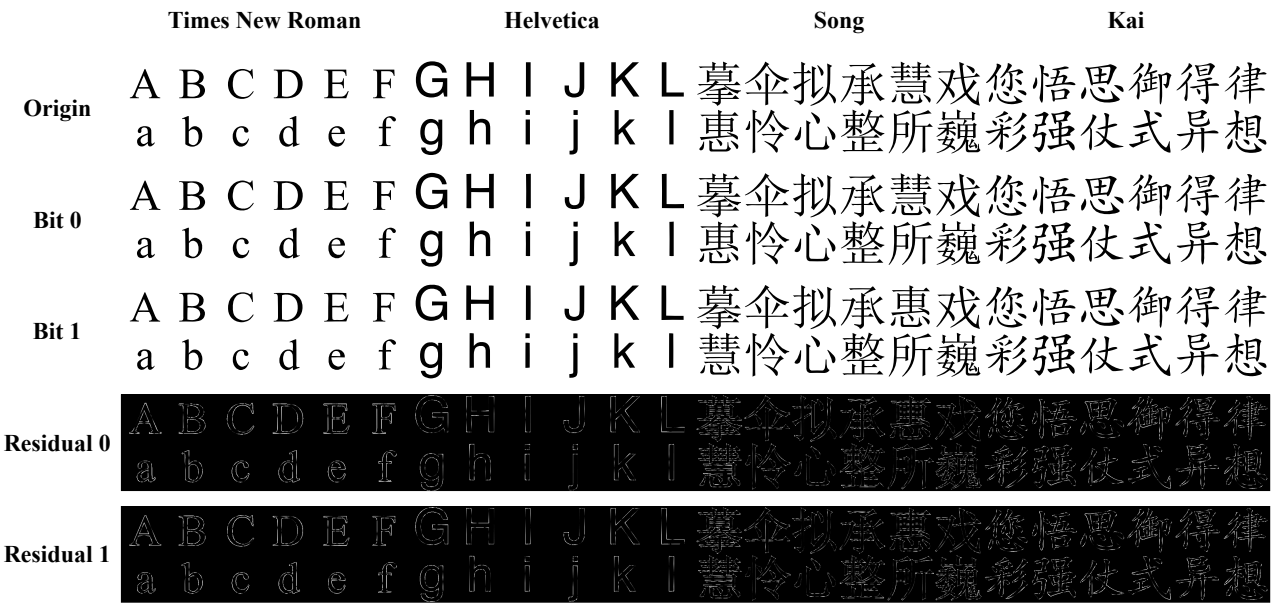

Figure 3: A display of four fonts with embedded information. The first row shows the original font. The second and third rows are the synthesized fonts embedded with bit 0 and bit 1, respectively. The last two rows are the residuals (multiplied by 10) of the synthesized font and the original font.

the vector character $\hat{G}$ of the embedded watermark information. It can be expressed by the following equation:

$$\mathcal{L}_q = \left\| I - \hat{I} \right\|_1 + \left\| S - \hat{S} \right\|_2 + \left\| G - \hat{G} \right\|_2 \quad (13)$$

where I is the image of the original character, S is the SDF of the original character, and G is the vector representation of the original character.

Message Loss. It uses the mean square error to minimize the distance between the extracted watermark $\hat{M}$ and the embedded watermark M:

$$\mathcal{L}_m = \left\| M - \hat{M} \right\|_2 \quad (14)$$

Total Loss. The overall loss can be expressed as:

$$\mathcal{L} = \lambda_1 \mathcal{L}_q + \lambda_2 \mathcal{L}_m \quad (15)$$

where λ_1 and λ_2 are hyperparameters used to balance the two losses, defaulting to 1 respectively.

Embedding Watermark in Document

As illustrated in Figure 2 (c), once trained, the model generates a Font Codebook comprising a collection of TTF fonts, each embedded with specific information and converted from its vector representation using FontForge[1]. Subsequently, this codebook is used to embed watermarks into a document following the process shown in Figure 1.

During the extraction phase, we first obtain the distorted document image (screenshot, photograph, etc.) and then use a pre-trained character segmentation model (Baek et al. 2019) to extract all the character images from the document image. These images are then fed into the Image Decoder to sequentially decode the watermark information.

[1]https://fontforge.org/

Experiments

Implementation Details

To verify the model's performance and generalization, we trained it on four commonly used English and Chinese fonts: Times New Roman, Helvetica, Song, and Kai. The dataset includes 3,498 common Chinese characters and all 52 English letters. The message length is 1. We used a fixed view size of 32×32, the batch size of 64, and Adam as the optimizer with a learning rate of 8×10^{-4}. All experiments are conducted on a single NVIDIA RTX 3090 GPU, and the model is trained for 4000 epochs.

Metrics. The evaluation of our model is based on two primary metrics: the robustness of the model's decoding and the visual quality of the synthesized fonts. To measure robustness, we use the average bit decoding accuracy (ACC). For assessing visual quality, we rely on PSNR and SSIM for quantitative analysis.

Baselines. The baselines for comparison include AutoStegaFont (Yang et al. 2023), the method proposed by Qi et al. (Qi et al. 2019), and FontCode (Xiao, Zhang, and Zheng 2018). Both the method by Qi et al. and FontCode require human assistance and are not yet open source. Out of respect, we rely on the results reported in the respective literature (Yang et al. 2023). All results are averaged over five runs.

Comparison with Previous Methods

Visual Quality. To demonstrate the quality of our synthesized fonts, we provide a visual example in Figure 3. As observed, the synthesized fonts generated by our method are nearly indistinguishable to the naked eye, maintaining

Font	Method	Identity	GN ($std = 0.1$)	JPEG ($Q = 50$)	Scale ($f = 0.5$)	GF ($\sigma = 1$)	Rotation ($deg = 10°$)	Translation ($dis = 0.1$)	Prespective ($f = 0.2$)
Times	Proposed	**100**	**94.72**	**97.75**	**96.09**	**100**	**94.58**	**100**	96.97
	AutoStegaFont	100	89.42	97.03	92.30	99.03	92.78	98.55	**98.07**
Helvetica	Proposed	**100**	**92.24**	**92.18**	**98.43**	**100**	**78.17**	**94.29**	**100**
	AutoStegaFont	89.42	74.04	86.53	89.42	87.50	74.88	87.01	85.57
Song	Proposed	**100**	**96.25**	**97.69**	**85.03**	**98.64**	**96.36**	**98.14**	**99.34**
	AutoStegaFont	92.57	60.57	83.65	78.84	88.46	82.69	84.61	83.65
Kai	Proposed	**99.63**	**74.27**	**83.60**	**71.34**	**97.14**	**84.86**	**97.72**	**94.19**
	AutoStegaFont	75.38	59.61	75.38	66.73	58.65	63.46	57.69	57.69

Table 1: Robustness evaluation under different distortions. "GN" represents Gaussian Noise and "GF" denotes Gaussian Filter. For Rotation, we evaluate the performance under both clockwise and counterclockwise rotations and average the results. Similarly, for Translation, we test the performance across four directions—up, down, left, and right—and average the outcomes. All character images are fixed at a size of 32×32, and the distortions are implemented using Kornia (E. Riba and Bradski 2020).

high visual fidelity. Additionally, the residuals reveal that the modifications introduced by our method are uniformly distributed across the fonts, significantly complicating steganographic analysis. We also provide the quantitative results in Table 2. Our method is able to synthesize higher quality fonts than AutoStegaFont.

Method	Metric	Times	Helvetica	Song	Kai
Proposed	PSNR	33.53	34.41	30.11	32.42
	SSIM	0.9961	0.9973	0.9946	0.9965
AutoStegaFont	PSNR	25.83	28.61	23.56	23.59
	SSIM	0.9565	0.9769	0.9488	0.9493

Table 2: Synthetic quality assessment of four different fonts.

Robustness against Different Distortions. Table 1 presents the ACC of our model under various distortion conditions, including Identity, Gaussian Noise ($std = 0.1$), JPEG Compression (quality factor $Q = 50$), Image Scaling ($f = 0.5$), Gaussian Filter (kernel size 3×3, $\sigma = 0.1$), Rotation ($deg = 10°$), Translation ($dis = 0.1$), and Perspective Transformation (intensity $f = 0.2$). Our method consistently achieves higher accuracy across most distortions. While AutoStegaFont performs reasonably well on English fonts with simpler glyphs and fewer characters, it struggles significantly with Chinese fonts. Apparently the error caused by its two-stage training strategy is further amplified when facing a large number of complex characters. In contrast, our method maintains decoding accuracy above 90% across various fonts, underscoring the importance of end-to-end training and demonstrating the robustness of our approach in handling complex characters.

Comparison under Physical World Distortions. We select three common physical world document distortions: Screenshots, Print-Camera Shooting, and Screen-Camera Shooting to verify the effectiveness of our method. For Qi et al.'s method and FontCode, we align with the experimental setups described in the literature (Yang et al. 2023) and directly use the reported results.

Font	Method	Font Size (px) 16	20	24	32	40
Times	Proposed	**79.30**	**88.46**	**94.23**	**100**	**100**
	AutoStegaFont	57.69	64.80	80.19	100	100
	Qi et al.	51.92	61.53	65.38	67.31	71.15
Helvetica	Proposed	**82.69**	**96.15**	**100**	**100**	**100**
	AutoStegaFont	56.73	65.76	73.07	89.42	94.64
	Qi et al.	59.62	63.08	67.50	70.96	75
Song	Proposed	**63.76**	**81.10**	**87.84**	**100**	**100**
	AutoStegaFont	58.65	65.57	77.30	92.57	95.36
	Qi et al.	50.19	53.85	57.69	65.38	68.65
Kai	Proposed	**63.87**	**72.53**	**88.79**	**99.63**	**99.56**
	AutoStegaFont	51.92	53.82	62.38	75.38	83.33

Table 3: Robustness against Screenshots with four different fonts under different font sizes.

Table 3 demonstrates the model's robustness to screenshots at different resolutions (font sizes). Our method consistently outperforms the baseline across all cases, maintaining an accuracy of approximately 90% when the font size exceeds 24px. However, at lower resolution (16px), we observe a significant drop in the decoding accuracy of Chinese characters compared to English characters. This discrepancy likely arises due to the intricate structure of Chinese fonts, which contain more complex strokes and finer details that become increasingly difficult to preserve and recognize as resolution decreases. In contrast, English characters, with their relatively simpler structures, exhibit greater resilience under the same conditions. Addressing this challenge and improving the robustness of our method for languages with complex writing systems, such as Chinese, remains an open problem that warrants further investigation.

We also evaluate different models under Print-Camera Shooting and Screen-Camera Shooting conditions. Specifically, we fix the cell phone at a shooting angle of $0°$ and a distance of 30cm. For Screen-Camera Shooting, we use a 14-inch "Lenovo XIAOXIN Pro 14" device. For Print-

Methods	PSNR	SSIM	Identity	GN ($std = 0.1$)	JPEG ($Q = 50$)	Scale ($f = 0.5$)	GF ($\sigma = 1$)	Rotation ($deg = 10°$)	Translation ($dis = 0.1$)	Perspective ($f = 0.2$)
HiDDeN	24.70	0.9891	66.07	60.97	51.76	51.58	58.96	53.75	51.76	52.24
SSL	33.04	0.9745	87.12	60.77	55.00	48.08	86.35	51.73	52.12	48.08
CIN	**42.52**	0.9943	99.82	76.71	67.87	62.75	61.14	57.86	57.97	56.92
Ours	33.53	**0.9961**	**100**	**94.72**	**97.75**	**96.09**	**100**	**94.58**	**100**	**96.97**

Table 4: Performance comparison of image-based watermarking methods on visual quality and robustness.

Font	Method	Print-Camera	Screen-Camera
Timse	Proposed	**94.85**	**88.16**
	AutoStegaFont	86.17	82.64
	Qi et al.	69.23	69.42
	FontCode	40	-
Helvetica	Proposed	**96.53**	**92.72**
	AutoStegaFont	88.07	83.26
	Qi et al.	72.88	67.12
Song	Proposed	**92.40**	**85.25**
	AutoStegaFont	80.79	77.51
	Qi et al.	67.69	68.08
Kai	Proposed	**88.46**	**79.47**
	AutoStegaFont	76.59	68.26

Table 5: Robustness against Print-Camera Shooting and Screen-Camera Shooting distortions.

Camera Shooting, images are printed on A4 paper with an "HP LaserJet Pro MFP M126nw" and then captured. Since FontCode provides experimental data only at a font size of 40px, our tests is conducted at this size. The results of our evaluation are shown in Table 5. Our method generally outperforms the baseline in most cases. Accuracy under Screen-Camera Shooting is lower than Print-Camera Shooting due to additional distortions like downsampling ight reflections, and moiré patterns. Improving resistance to these distortions will be a key focus for future research.

Comparison with Image-based Methods. In addition to font-based approaches, we also evaluate image-based baselines, including HiDDeN (Zhu et al. 2018), SSL(Fernandez et al. 2022), and CIN (Ma et al. 2022). As shown in Table 4, our approach achieves superior visual quality and higher decoding accuracy under various distortions. In particular, while CIN performs relatively well on PSNR and SSIM, its robustness drops sharply under geometric transformations, highlighting the limitation of image-based watermarking in handling texture-sparse character images. In contrast, our vector-based design maintains near-perfect decoding accuracy across all perturbations.

Ablation Study

In this section, we present a detailed evaluation of the proposed dual-branch Vector Encoder. As shown in Table 6, using only the Global branch yields the lowest visual quality due to its limited capacity for local detail modeling. Employing only the Local branch improves PSNR by about 2dB but reduces average bit decoding accuracy by nearly 14%, indicating a trade-off between visual fidelity and watermark robustness. The Global branch achieves higher decoding accuracy by capturing the overall glyph structure, while the Local branch focuses on fine contour details to improve visual fidelity. Combining both branches achieves a balanced design, delivering high-quality font synthesis and reliable decoding.

Structure	ACC	PSNR	SSIM
Global	96.07	30.42	0.9862
Local	82.73	32.24	0.9904
Global & Local	**99.14**	**33.53**	**0.9961**

Table 6: Impact of different structures of the Vector Encoder on performance. We conducted experiments using the Times New Roman font, evaluating the average decoding accuracy under random noise attacks and the visual quality of the synthesized fonts.

Conclusion

In this paper, we propose a novel end-to-end vector font watermarking framework for robust and generalizable text protection in the physical world. Existing watermarking methods typically treat font synthesis and watermark extraction as separate processes, which limits both the visual fidelity of the generated fonts and the robustness of the embedded watermark. To address this limitation, we introduce a differentiable rasterization strategy that leverages the Signed Distance Field (SDF), enabling gradients to propagate seamlessly from the watermark decoder back to the font generator for true end-to-end optimization. Moreover, we design a dual-branch vector encoder tailored to the intrinsic structure of vector fonts: the Global branch captures overall glyph geometry, while the Local branch focuses on fine-grained contours. This combination ensures high-quality font synthesis while preserving reliable watermark decoding under various distortions. Extensive experiments demonstrate that our framework consistently outperforms state-of-the-art image- and font-based methods in both visual quality and decoding accuracy, offering an efficient and general solution for real-world document protection scenarios.

Acknowledgments

This work was supported in part by the Natural Science Foundation of China under Grant 62372203 and 62302186, in part by the Major Scientific and Technological Project of Shenzhen (202316021), in part by the National key research and development program of China(2022YFB2601802), in part by the Major Scientific and Technological Project of Hubei Province (2022BAA046, 2022BAA042).

References

Baek, Y.; Lee, B.; Han, D.; Yun, S.; and Lee, H. 2019. Character region awareness for text detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9365–9374.

Campbell, N. D.; and Kautz, J. 2014. Learning a manifold of fonts. *ACM Transactions on Graphics (ToG)*, 33(4): 1–11.

Chang, Q.; Huang, L.; Liu, S.; Liu, H.; Yang, T.; and Wang, Y. 2022. Blind Robust Video Watermarking Based on Adaptive Region Selection and Channel Reference. *Proceedings of the 30th ACM International Conference on Multimedia*.

Chen, G.; Wu, Y.; Liu, S.; Liu, T.; Du, X.; and Wei, F. 2023. WavMark: Watermarking for Audio Generation. *ArXiv*, abs/2308.12770.

E. Riba, D. P. E. R., D. Mishkin; and Bradski, G. 2020. Kornia: an Open Source Differentiable Computer Vision Library for PyTorch. In *Winter Conference on Applications of Computer Vision*.

Fang, T.; Jaggi, M.; and Argyraki, K. 2017. Generating steganographic text with LSTMs. *arXiv preprint arXiv:1705.10742*.

Fernandez, P.; Sablayrolles, A.; Furon, T.; Jégou, H.; and Douze, M. 2022. Watermarking images in self-supervised latent spaces. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3054–3058. IEEE.

Jia, Z.; Fang, H.; and Zhang, W. 2021. Mbrs: Enhancing robustness of dnn-based watermarking by mini-batch of real and simulated jpeg compression. In *Proceedings of the 29th ACM international conference on multimedia*, 41–49.

Kirchenbauer, J.; Geiping, J.; Wen, Y.; Katz, J.; Miers, I.; and Goldstein, T. 2023. A watermark for large language models. In *International Conference on Machine Learning*, 17061–17084. PMLR.

Koblitz, N. 1987. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177): 203–209.

Lan, Y.; Shang, F.; Yang, J.; Kang, X.; and Li, E. 2023. Robust Image Steganography: Hiding Messages in Frequency Coefficients. In *AAAI Conference on Artificial Intelligence*.

Li, T.-M.; Lukác, M.; Gharbi, M.; and Ragan-Kelley, J. 2020. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39: 1 – 15.

Liu, C.; Zhang, J.; Fang, H.; Ma, Z.; Zhang, W.; and Yu, N. 2022. DeAR: A Deep-learning-based Audio Re-recording Resilient Watermarking. In *AAAI Conference on Artificial Intelligence*.

Liu, Y.-T.; Zhang, Z.; Guo, Y.; Fisher, M.; Wang, Z.; and Zhang, S. 2023. DualVector: Unsupervised Vector Font Synthesis with Dual-Part Representation. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 14193–14202.

Ma, R.; Guo, M.; Hou, Y.; Yang, F.; Li, Y.; Jia, H.; and Xie, X. 2022. Towards blind watermarking: Combining invertible and non-invertible mechanisms. In *Proceedings of the 30th ACM International Conference on Multimedia*, 1532–1542.

Pan, W.; Zhu, A.; Zhou, X.; Iwana, B. K.; and Li, S. 2023. Few shot font generation via transferring similarity guided global style and quantization local style. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 19449–19459.

Park, S.; Chun, S.; Cha, J.; Lee, B.; and Shim, H. 2021. Multiple Heads are Better than One: Few-shot Font Generation with Multiple Localized Experts. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 13880–13889.

Pavlovic, K.; Kovacevic, S.; Djurović, I.; and Wojciechowski, A. 2021. Robust speech watermarking by a jointly trained embedder and detector using a DNN. *Digit. Signal Process.*, 122: 103381.

Qi, W.; Guo, W.; Zhang, T.; Liu, Y.; Guo, Z. M.; and Fang, X. 2019. Robust authentication for paper-based text documents based on text watermarking technology. *Mathematical biosciences and engineering : MBE*, 16 4: 2233–2249.

Reddy, P.; Zhang, Z.; Wang, Z.; Fisher, M.; Jin, H.; and Mitra, N. 2021. A multi-implicit neural representation for fonts. *Advances in Neural Information Processing Systems*, 34: 12637–12647.

Rivest, R. L.; Shamir, A.; and Adleman, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2): 120–126.

song Xu, Y.; Mou, C.; Hu, Y. F.; Xie, J.; and Zhang, J. 2022. Robust Invertible Image Steganography. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7865–7874.

Sun, N.; Zhao, C.; Xie, S.; and Ling, H. 2024. InvertedFontNet: Font Watermarking based on Perturbing Style Manifold. *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Tancik, M.; Mildenhall, B.; and Ng, R. 2019. StegaStamp: Invisible Hyperlinks in Physical Photographs. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2114–2123.

Tang, L.; Cai, Y.; Liu, J.; Hong, Z.; Gong, M.; Fan, M.; Han, J.; Liu, J.; Ding, E.; and Wan, J. 2022. Few-Shot Font Generation by Learning Fine-Grained Local Styles. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7885–7894.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, C.-Y.; Zhou, M.; Ge, T.; Jiang, Y.; Bao, H.; and Xu, W. 2023. CF-Font: Content Fusion for Few-Shot Font Generation. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 1858–1867.

Wang, Y.; and Lian, Z. 2021. DeepVecFont. *ACM Transactions on Graphics (TOG)*, 40: 1 – 15.

Wang, Y.; and Pearmain, A. J. 2006. Blind MPEG-2 video watermarking robust against geometric attacks: a set of approaches in DCT domain. *IEEE Transactions on Image Processing*, 15: 1536–1543.

Xia, Z.; Xiong, B.; and Lian, Z. 2023. VecFontSDF: Learning to Reconstruct and Synthesize High-Quality Vector Fonts via Signed Distance Functions. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 1848–1857.

Xiao, C.; Zhang, C.; and Zheng, C. 2018. Fontcode: Embedding information in text documents using glyph perturbation. *ACM Transactions on Graphics (TOG)*, 37(2): 1–16.

Yang, X.; Zhang, J.; Fang, H.; rui Liu, C.; Ma, Z.; Zhang, W.; and Yu, N. H. 2023. AutoStegaFont: Synthesizing Vector Fonts for Hiding Information in Documents. In *AAAI Conference on Artificial Intelligence*.

Yoo, K.; Ahn, W.; Jang, J.; and Kwak, N. 2023. Robust multi-bit natural language watermarking through invariant features. *arXiv preprint arXiv:2305.01904*.

Zhang, Y.; Ni, J.; Su, W.; and Liao, X. 2023. A Novel Deep Video Watermarking Framework with Enhanced Robustness to H.264/AVC Compression. *Proceedings of the 31st ACM International Conference on Multimedia*.

Zhu, J.; Kaplan, R.; Johnson, J.; and Fei-Fei, L. 2018. Hidden: Hiding data with deep networks. In *Proceedings of the European conference on computer vision (ECCV)*, 657–672.