AutoStegaFont: Synthesizing Vector Fonts for Hiding Information in Documents

Xi Yang[1], **Jie Zhang**[1,2], **Han Fang**[*3], **Chang Liu**[1], **Zehua Ma**[1],
Weiming Zhang[*1], **Nenghai Yu**[1]
[1]University of Science and Technology of China
[2]University of Waterloo
[3]National University of Singapore
yx9726@mail.ustc.edu.cn, jiezhangsp@gmail.com, fanghan@nus.edu.sg
{hichangliu@mail., mzh045@mail., zhangwm@, ynh@}ustc.edu.cn

Abstract

Hiding information in text documents has been a hot topic recently, with the most typical schemes of utilizing fonts. By constructing several fonts with similar appearances, information can be effectively represented and embedded in documents. However, due to the unstructured characteristic, font vectors are more difficult to synthesize than font images. Existing methods mainly use handcrafted features to design the fonts manually, which is time-consuming and labor-intensive. Moreover, due to the diversity of fonts, handcrafted features are not generalizable to different fonts. Besides, in practice, since documents might be distorted through transmission, ensuring extractability under distortions is also an important requirement. Therefore, three requirements are imposed on vector font generation in this domain: automaticity, generalizability, and robustness. However, none of the existing methods can satisfy these requirements well and simultaneously.

To satisfy the above requirements, we propose **AutoStegaFont**, an automatic vector font synthesis scheme for hiding information in documents. Specifically, we design a two-stage and dual-modality learning framework. In the first stage, we jointly train an encoder and a decoder to invisibly encode the font images with different information. To ensure robustness, we target designing a noise layer to work with the encoder and decoder during training. In the second stage, we employ a differentiable rasterizer to establish a connection between the image and the vector modality. Then, we design an optimization algorithm to convey the information from the encoded image to the corresponding vector. Thus the encoded font vectors can be automatically generated. Extensive experiments demonstrate the superior performance of our scheme in automatically synthesizing vector fonts for hiding information in documents, with robustness to distortions caused by low-resolution screenshots, printing, and photography. Besides, the proposed framework has better generalizability to fonts with diverse styles and languages.

Introduction

Text documents are one of the most widely-used media in our daily lives, and we can access useful information effectively by reading them. Meanwhile, it is still interesting to hide more information in text documents, which can achieve massive practical commercial applications. For example, embedding unique hyperlinks in documents can push abundant information to readers through cellphone scanning; embedding invisible watermarks can help protect the copyright of documents; embedding identification information such as employee IDs can help government agencies, the military, or companies trace the source of confidential documents after being leaked. All these applications have attracted increasing interest in both industry and academic fields (Xiao, Zhang, and Zheng 2018; Qi et al. 2019).

Due to the redundancy characteristics, information hiding in natural images has been well-studied (Baluja 2017; Zhu et al. 2018; Wengrowski and Dana 2019; Tancik, Mildenhall, and Ng 2020) and is mainly based on pixel-wise modifications. However, compared with natural images, document images only consist of simple text structures and smooth backgrounds, lacking complex textures and colors, which makes it more challenging to invisibly embed information.

Considering this, a few recently-reported studies exploit the characteristics of document files rather than document images to hide information, by modifying glyphs (i.e., the particular shape designs) of the font used. Specifically, Xiao *et al.* (Xiao, Zhang, and Zheng 2018) propose FontCode. They gather human volunteers to initialize the candidates of slightly perturbed glyphs from a font manifold (Campbell and Kautz 2014), which are further filtered to construct a font codebook. The information can be embedded by replacing the original glyphs with similar glyphs from the codebook. To extract the information, they train one classifier for each glyph, which means 52 classifiers are needed for the English alphabet. However, the process of constructing a codebook is labor-intensive and time-consuming. Meanwhile, the extraction strategy is only suitable for alphabetic languages such as English and cannot be extended to ideographic languages with a wide range of glyphs like Chinese. Besides, this method can only correctly extract information from high-resolution documents without much distortion.

Similarly, Qi *et al.* (Qi et al. 2019) propose to manually modify the strokes of Chinese glyphs to obtain deformations of glyphs to construct the font codebook. On the extraction side, a template-matching method (Yoo and Han 2009) is used to match the deformations. They share the same limitations with FontCode, namely, the expensive cost to manually construct the font codebook, weak generalizability to fonts

[*]Corresponding authors.

in different languages, and limited robustness to distortions.

To satisfy the requirements for automaticity, generalizability, and robustness simultaneously, we propose **AutoStegaFont**, a two-stage and dual-modality learning framework to automatically synthesize vector fonts for hiding information in documents. Unlike the methods based on handcrafted features, we explore neural networks to modify glyph vectors automatically. As Figure 1 shows, in the first stage, we jointly train an encoder and a decoder to synthesize glyph images encoded with a specific message. Fed with the message and glyph image, the encoder produces an encoded glyph image, from which the decoder can extract the message. To guarantee robustness against distortions from low-resolution screenshots, printing, and photography, an elaborately designed noise layer is introduced in the training.

Since our goal is to synthesize the encoded font vectors, in the second stage, we employ a differentiable rasterizer to establish a connection between the image modality and the vector modality for conveying the information carried on a glyph image to its corresponding glyph vector. Specifically, we set the coordinates in the drawing commands of the original glyph vector as optimizable parameters and iteratively update them to minimize the difference between the rendered image and the image generated by the well-trained encoder. Meanwhile, during the optimization, we require the decoder can still correctly extract information from the rendered image. After the automatic optimization process, the encoded vectors are synthesized. Experimental results demonstrate that our framework can be used to automatically synthesize font vectors in different styles or languages. Besides, the information embedded in documents with the synthesized fonts can still be extracted after being distorted by low-resolution screenshots, printing, and photography.

In summary, our contributions are three-fold:

- For the first time, we propose a two-stage and dual-modality learning framework to automatically synthesize font vectors that can be used to hide information in documents. By first generating encoded glyph images and then conveying the information carried on the images to the corresponding vectors, a high-quality vector font encoded with specific information can be synthesized.
- We targetively design a noise layer to work with the encoder and decoder during training. It not only ensures robustness against distortions in common document usage scenarios but also facilitates the modifications on the image modality to be mapped to the vector modality.
- Extensive experiments demonstrate that our framework can be used to automatically synthesize vector fonts in different languages and that the information embedded in documents using them can be extracted robustly under low-resolution screenshots, printing, and photography. Furthermore, we discuss how the encoder trained with the noise layer learns to embed information by modifying glyph outlines.

Related Work

DNN-based Information Hiding in Images. In recent years, many works that aim to hide information in images have been developed, such as HiDDeN (Zhu et al. 2018) and StegaStamp (Tancik, Mildenhall, and Ng 2020). Specifically, HiDDeN proposed an autoencoder-like architecture to jointly train an encoder and a decoder for information embedding and extraction. Based on this, StegaStamp further enhanced robustness to distortions resulting from real-world printing and photography. Nevertheless their effectiveness in natural images, it is non-trivial to directly extend these methods to text documents, which mainly consist of editable discrete characters and simple backgrounds. Even if we regard text documents as document images, these methods will also induce perceptible distortions that affect visual quality. To address it intrinsically, we suggest hiding information in fonts rather than document images.

Information Hiding in Text Documents. Hiding information in documents is more challenging than images. Earlier approaches focused on slightly adjusting the format of electronic documents to embed information, such as word and line spacing (Brassil, Low, and Maxemchuk 1999; Rizzo, Bertini, and Montesi 2016), which are fragile to real-world distortions. To remedy it, some studies (Ueoka, Murawaki, and Kurohashi 2021; Abdelnabi and Fritz 2021; Yang et al. 2022) tried to make semantic modifications, but there is no guarantee that the semantics of the modified text remain exactly the same as the original. Recently, FontCode (Xiao, Zhang, and Zheng 2018) was proposed to embed information by perturbing glyphs in English documents. In detail, they first gathered human volunteers to initialize the perturbed glyphs and form a font codebook. Then, for each letter, a corresponding classification network was trained for information extraction. Similarly, Qi *et al.* (Qi et al. 2019) proposed to create perturbed glyphs by moving strokes and adopt a template-matching algorithm to extract the information by calculating the similarity of the current glyph to the perturbed glyphs. These methods are not flexible in practice due to three limitations: (1) They rely heavily on human involvement to design the fonts, which is time-consuming and labor-intensive. (2) These methods are not generally applicable for different fonts. (3) They are only feasible in some limited scenarios such as documents with large font sizes. The subsequent experiments demonstrate that our **AutoStegaFont** can effectively overcome all above limitations.

Font Generation. In the last decade, deep learning has achieved unprecedented success in font image generation (Gao et al. 2019; Wang, Gao, and Lian 2020). However, it is still under-explored to generate font vectors. Despite the attempts of a few approaches (Carlier et al. 2020; Lopes et al. 2019), their performance is still far from satisfactory. Recently, Wang *et al.* (Wang and Lian 2021) first propose a novel method that can synthesize high-quality font vectors, in which they leverage a differentiable rasterizer to refine the visual quality of the generated vectors. Motivated by this, we introduce the rasterizer to transfer the information carried on encoded glyph images to the corresponding glyph vectors.

Method

In this section, we first elaborate the proposed two-stage and dual-modality learning framework to automatically synthe-

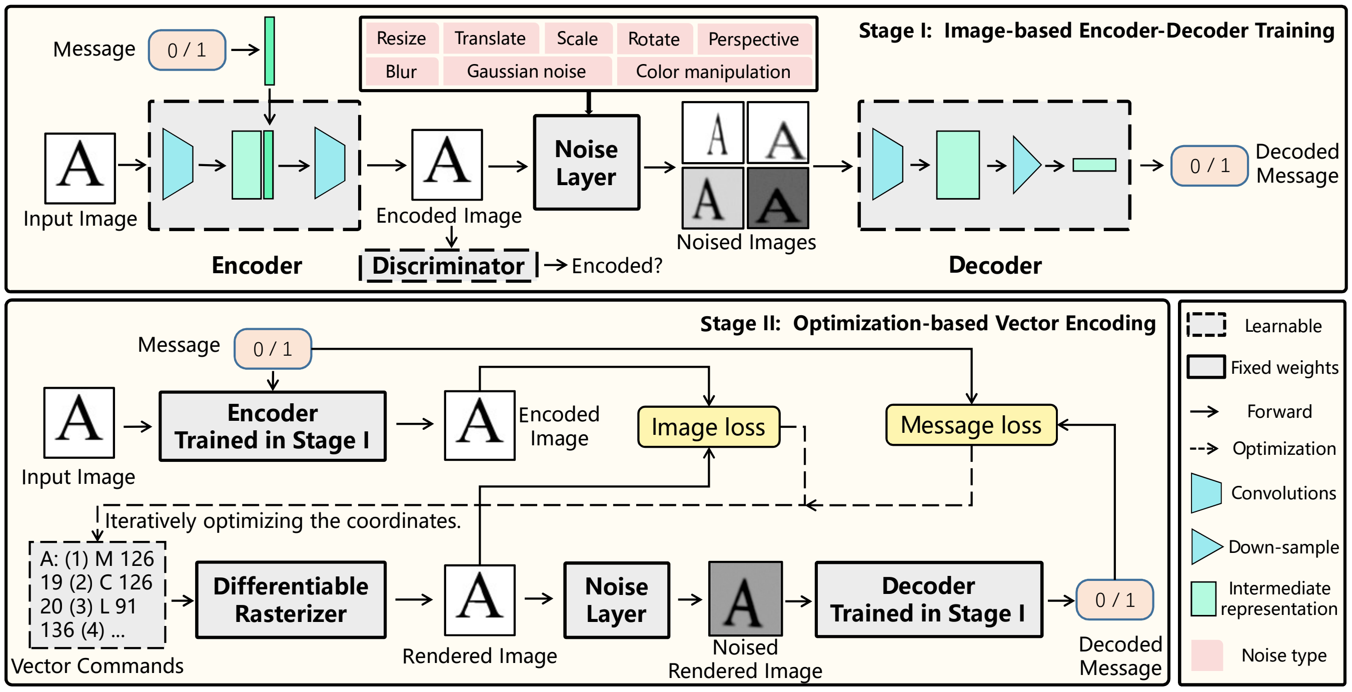

Figure 1: An overview of our two-stage and dual-modality framework. Stage I: we jointly train the encoder and decoder along with a noise layer. Stage II: we build the connection between the image modality and the vector modality. Briefly, we iteratively optimize the drawing commands of an input vector until its rendered image is indistinguishable from the corresponding encoded image and is encoded with the same message. We use absolute coordinates for each command, and ‘M’, ‘L’, and ‘C’ in front of the coordinates denote the move, line, and curve commands, respectively.

size vector fonts for hiding information in documents. After that, we describe how to use the synthesized fonts to embed and extract information in documents.

Framework Overview

Here, we briefly describe the pipeline of our AutoStegaFont framework. As shown in Figure 1, in the first stage, we train the encoder-decoder network for encoding and recovering information in glyph images along with a noise layer simulating the distortions in document usage scenarios. After training, we use the well-trained encoder to modify each glyph of a given font so that each glyph is encoded with a one-bit message. The message carried by the encoded images can still be recovered by the decoder after passing through the noise layer. In the second stage, we set the coordinates of the drawing commands of the original glyph vector as to-be-optimized parameters and employ a differentiable rasterizer to render the glyph vector to the glyph image. During the optimization process, we shall guarantee pixel-level consistency between the rendered image and the encoded image. In addition, the decoder shall extract the correct message from the rendered image. Finally, we can obtain the encoded vectors, based on which we can construct computer-usable font files (e.g., TTF files) to hide information in document files.

Stage I: Image-based Encoder-Decoder Training

Our glyph image synthesis system mainly comprises three components: an encoder E, a noise layer N, and a decoder D. The inputs of the encoder E include a grayscale original glyph image I_{ori} of shape 64×64 and a one-bit message $M \in \{0, 1\}$. The output of E is an encoded image I_{en} of the same shape as I_{ori}. Then, I_{en} is distorted by the noise layer N to produce a noised image I_{no}. The decoder D tries to extract the corresponding one-bit message M_{out} from I_{no}.

Encoder. The encoder is trained to encode a one-bit message on each glyph image while minimizing perceptual differences between the input and the encoded glyph images. It first applies convolutions with residual blocks to form an intermediate representation. Then the one-bit message is replicated to the same shape as I_{ori} and concatenated with the intermediate representation. After several subsequent convolutions, the encoder produces I_{en}. To guarantee visual quality, we introduce the basic loss $\mathcal{L}_{vq}$, which calculates the mean squared error (MSE) between I_{ori} and I_{en}, *i.e.*,

$$\mathcal{L}_{vq} = MSE(I_{ori}, I_{en}). \quad (1)$$

For preserving feature-wise details of the glyph images, the perceptual loss (Johnson, Alahi, and Fei-Fei 2016) is adopted by calculating the feature distance in different layers of the VGG network (Simonyan and Zisserman 2014), which is represented as $\mathcal{L}_{percep}(I_{ori}, I_{en})$. Besides, we introduce the adversarial loss $\mathcal{L}_A$ to further enhance the visual quality of I_{en}, which tries to make the additional adversarial discriminator A cannot distinguish I_{en} from I_{ori}, *i.e.*,

$$\mathcal{L}_A = \log(1 - A(I_{en})). \tag{2}$$

Meanwhile, for the discriminator, the loss function is

$$\mathcal{L}_{disc} = \log(A(I_{en})) + \log(1 - A(I_{ori})). \tag{3}$$

Noise Layer. During training, we apply a series of differentiable perturbations to approximate the distortions caused by physically displaying and imaging the documents between the encoder and the decoder. Based on the previous work (Tancik, Mildenhall, and Ng 2020), we design the noise layer suitable for document usage scenarios, including resizing, translation, scaling, rotation, perspective transformation, blurring, Gaussian noise, and color manipulation. The encoded glyph image I_{en} will go through this noise layer before being fed into the decoder. It is worth mentioning that, our noise layer can drive the encoder to modify the glyph outline, which not only ensures robustness to the above distortions but also facilitates the mapping of the modifications on the image modality to the vector modality.

Decoder. The decoder is trained to recover a one-bit message from a noised glyph image I_{no}. Since each glyph image is encoded with one bit message, the decoder is equivalent to a binary classifier, whose output M_{out} means the probability of bit '0' or '1'. Therefore, we use the cross-entropy loss to minimize the message extraction loss $\mathcal{L}_m$, *i.e.*,

$$\mathcal{L}_m = -M \cdot \log(M_{out}) - (1 - M) \cdot \log(1 - M_{out}). \tag{4}$$

Training. In Stage I, we jointly train the encoder and the decoder, and the whole training loss can be formulated as follows:

$$\mathcal{L}_1 = \lambda_{vq}\mathcal{L}_{vq} + \lambda_p\mathcal{L}_{percep} + \lambda_A\mathcal{L}_A + \lambda_m\mathcal{L}_m, \tag{5}$$

which is the weighted sum of the loss terms.

Stage II: Optimization-based Vector Encoding

The goal in the second stage is to convey the information from an encoded glyph image to its corresponding vector. To achieve this, we employ a precise differentiable rasterizer R that can render the drawing commands (consisting of coordinate types and coordinate values) of an input glyph vector to a glyph image of the specified size. Besides, we also leverage the noise layer N, the well-trained encoder E, and the decoder D in Stage I, for the forward and optimization process in Stage II.

Forward. The input of the rasterizer is the drawing commands of the original glyph vector. We use C to represent the coordinates in the commands. The rasterizer outputs the rendered image I_{rend} of the drawing commands. Then, I_{rend} is distorted by the noise layer N to produce a noised rendered image I'_{no}, from which the well-trained decoder D tries to extract the one-bit message M'.

Algorithm 1: Optimization-based encoding. The encoder E and decoder D are pre-trained in Stage I. We set the number of iteration $k = 200$, and the optimization step length $l = 1$. The loss functions are demonstrated in Eq.(6) to Eq.(8).

```
Input: The original font image $I_{ori}$, the coordinate values in the
    drawing commands of the original font vector $C$, the one-bit
    message $M$.
Output: The coordinate values in the drawing commands of the
    encoded font vector $C_{en}$.
 1: $I_{en} = E(I_{ori}, M)$;
 2: // Generate encoded font image $I_{en}$ using the encoder $E$
    trained in the first stage.
 3: $C_{en} = C$;
 4: for $k$ iterations do
 5:     $I_{rend} = Rastrizer(C_{en})$;
 6:     $M' = D(I_{rend})$;
 7:     $\mathcal{L}_2 = \lambda_{img}\mathcal{L}_{img}(I_{en}, I_{rend}) + \lambda_{msg}\mathcal{L}_{msg}(M, M')$;
 8:     $C_{en} = C_{en} - l \cdot \nabla_{C_{en}}\mathcal{L}_2$;
 9: end for
10: return $C_{en}$
```

Optimization. We set the coordinates C as to-be-optimized parameters and iteratively update C through the standard back-propagation until its rendered image I_{rend} carries the same message as the encoded image I_{en}. Then, the encoded commands C_{en} can be used to directly build the vector format encoded glyph. To achieve this goal, we need to make I_{rend} as consistent as possible with I_{en}, which can transfer the invisible and robust modifications on I_{en} to the drawing commands of the glyph vector. For this, we introduce loss $\mathcal{L}_{img}$ to constrain the pixel-level consistency, *i.e.*,

$$\mathcal{L}_{img} = MSE(I_{en}, I_{rend}). \tag{6}$$

Besides, to further ensure that the rendered image carries the one-bit message, we append the message extraction loss :

$$\begin{aligned}\mathcal{L}_{msg} &= -M \cdot \log(M') - (1 - M) \cdot \log(1 - M') \\ &= -M \cdot \log(D(I_{rend})) - (1 - M) \cdot \log(1 - D(I_{rend})).\end{aligned} \tag{7}$$

The optimization objective function in Stage II is represented as follows:

$$\mathcal{L}_2 = \lambda_{img}\mathcal{L}_{img} + \lambda_{msg}\mathcal{L}_{msg}, \tag{8}$$

where λ_{img} and λ_{msg} are used to balance the two terms. After optimization, we can obtain the final optimal vector commands C_{en}, *i.e.*,

$$C_{en} = \arg\min_{C \in \mathbb{C}} \mathcal{L}_2(C; I_{en}, M), \tag{9}$$

where $\mathbb{C}$ denotes the selection of all possible coordinate values. The detailed process is provided in Algorithm 1.

Message Embedding and Extraction in Documents

After performing the above two-stage procedure for all glyphs in a given font F, we obtain two encoded fonts, namely, F_0 carrying bit 0 and F_1 carrying bit 1, both of which are visually indistinguishable from F. Here, we illustrate in detail how to use these fonts to embed and extract a multi-bit message in a document, as shown in Figure 2.

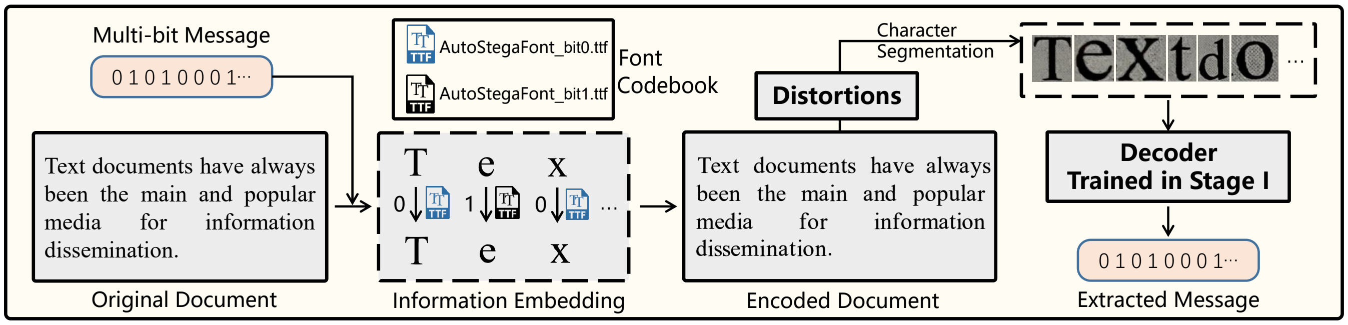

Figure 2: The font-based message embedding and image-based message extraction process. We synthesize two fonts using the AutoStagaFont framework, where all glyphs in one font are encoded with bit 0 and all glyphs in the other are encoded with bit 1. Then, we embed a multi-bit message in the given document (a PDF or DOC file) using these fonts (TTF files). To extract the message, we segment the document image into individual glyph images and extract the bitwise information in order.

Embedding. Given an original document (*e.g.*, a PDF or DOC file) as input, we first read the text content from it. Taking English as an example, the text content mainly consists of letters. To embed a one-bit message in a letter, we replace the original letter with the corresponding letter in the encoded font file selected according to the one-bit message. For example, if we want to embed bit 0 in the current letter 'A', we replace it with the corresponding 'A' in F_0, and conversely, if we want to embed bit 1, we replace it with the 'A' in F_1. As each letter can carry one bit of message, we can embed a multi-bit message (*e.g.*, watermarks, hyperlinks, employee IDs) in a document with a series of letters.

Extraction. To extract the multi-bit message from a document image, we first use the character region-aware text detector, CRAFT (Baek et al. 2019), to segment each glyph in the document image individually. Then, each individual glyph image will be sent into the decoder trained in the first stage to produce a one-bit message, so that the multi-bit message can be finally extracted. More importantly, since the decoder is trained for all glyphs in a given font, we do not need to recognize the semantic information of each glyph before extracting the message, which greatly improves efficiency.

Experimental Results

Experiment Settings

Dataset. We select Times New Roman and Helvetica as representatives of serif and sans-serif English fonts, respectively. Besides, to verify the generalizability of our framework in different languages, we also conduct experiments on Song, which is the most commonly used Chinese font. We use the open source font editor FontForge[1] to obtain rendered glyph images and corresponding SVG commands.

Implementation Details. We utilize Adam (Kingma and Ba 2014) as the optimizer of our models. To obtain better optimization results in the second stage, we adopt the glyph image super-resolution model released by (Wang and Lian 2021) to increase the resolution of the initial encoded glyph images from 64×64 to 256×256. We adopt the differentiable rasterizer provided by (Li et al. 2020). For the weight factors in the first stage, we set $\lambda_{vq} = 5, \lambda_{percep} = 0.01, \lambda_m = \lambda_A = 1$. In the second stage, we set $\lambda_{img} = 1$ and $\lambda_{msg} = 10^{-5}$. The number of iterations k is set to 200.

Baseline. There are currently two methods suitable for comparison, *i.e.*, the template-based method (Qi et al. 2019) and the manifold-based method FontCode (Xiao, Zhang, and Zheng 2018). We reproduce Qi's method as the baseline. For a fair comparison, the amount of modifications on glyphs is guaranteed to be the same as our method. For FontCode, the experimental process is difficult to reproduce because it requires a large number of volunteers to participate in designing the font codebook, and there is no publicly available code. Therefore, we align the experimental settings provided in their paper and use the released results for comparison.

Metrics. The peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) are used to evaluate the visual quality of the encoded glyph images. To evaluate robustness under different distortions, we adopt the average extraction accuracy ($\overline{ACC}$), *i.e.*, the percentage of bits correctly extracted.

Experimental Results

Visual Quality. In Figure 3, we showcase our synthesized fonts corresponding to Times New Roman and Song, respectively. The AutoStegaFonts can guarantee high visual quality in all cases. We also provide the quantitative results in Table 1, which further demonstrate that the encoded glyph images are hard to be distinguished from the original ones.

Font	Times	Helvetica	Song
PSNR ↑	35.52	33.53	35.93
SSIM ↑	0.9962	0.9948	0.9977

Table 1: Quantitative evaluation on the visual quality.

[1]https://fontforge.org/

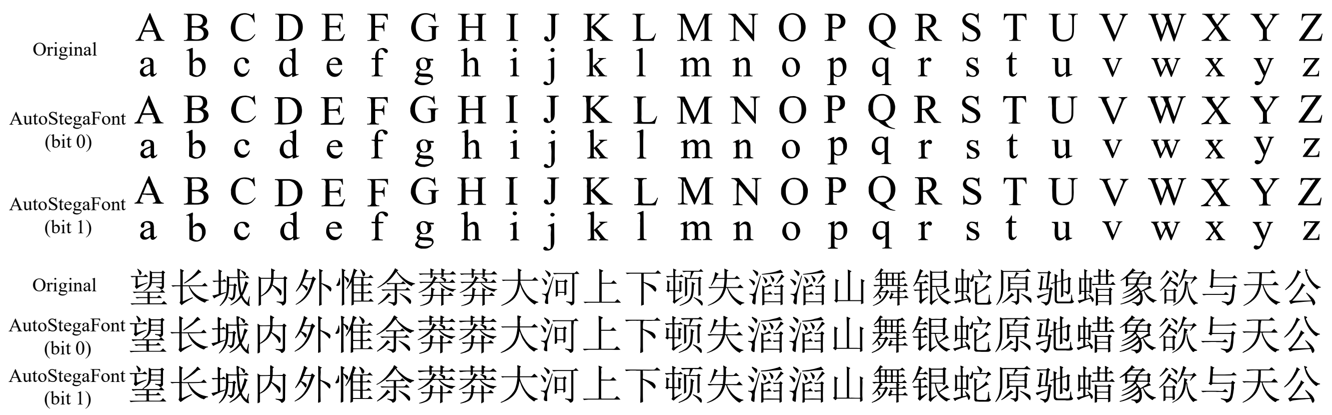

Figure 3: Examples of the synthesized vector format AutoStegaFonts for Times New Roman (English) and Song (Chinese), where glyphs in the first line come from the original font, each glyph in the second line is encoded with bit 0, and each glyph in the third line is encoded with bit 1.

Font	Method	16 px	20 px	24 px	32 px	40 px
Times	Ours	86.93	98.56	99.52	100	100
	Baseline	51.92	61.53	65.38	67.31	71.15
Helvetica	Ours	93.46	99.13	99.62	99.90	100
	Baseline	59.62	63.08	67.50	70.96	75
Song	Ours	83.44	92.88	95.77	96.54	97.60
	Baseline	50.19	53.85	57.69	65.38	68.65

Table 2: Robustness ($\overline{ACC}$ (%)) against screenshots with three different fonts under different resolutions (*i.e.*, font sizes).

Robustness against Screenshots. Documents are often propagated via screenshots, where the distortions mainly come from the down-sampling of the screen. We first print all the encoded glyphs (repeated 10 times) in a PDF file and take screenshots at different resolutions (i.e., font sizes). Then, we calculate the average extraction accuracy ($\overline{ACC}$) on the extracted information. As shown in Table 2, we achieve better robustness on English fonts than Chinese fonts. Nevertheless, in most cases, we obtain above 90% successful extraction, which outperforms the baseline method by a large margin. Even at font size of 16px, an $\overline{ACC}$ largger than 80% is still guaranteed. This means that, even at very low resolutions, we can still achieve correct extraction by using error-correcting codes like BCH codes.

Robustness against Print-Camera Shooting. A more common scenario is using a cellphone camera to capture printed paper documents to extract information. Therefore, we print all the encoded glyphs (repeated 10 times) on A4 papers at different font sizes and calculate the average extraction accuracy ($\overline{ACC}$) after taking photos at a distance of 30 cm with a handheld cellphone. As Table 3 shows, we can still achieve above 90% extraction accuracy when the font size is larger than 20px. As mentioned above, to compare with FontCode, we align the experimental settings provided

Font	Method	16 px	20 px	24 px	32 px	40 px
Times	Ours	82.24	89.90	91.54	95.69	96.35
	Baseline	53.85	55.77	61.54	63.46	69.23
	FontCode	-	-	-	-	40
Helvetica	Ours	88.64	94.17	96.25	97.79	97.88
	Baseline	60.96	62.12	66.92	69.04	72.88
Song	Ours	79.84	88.92	90.76	92.11	95.77
	Baseline	55.76	63.46	65.38	67.50	67.69

Table 3: Robustness ($\overline{ACC}$ (%)) against print-camera shooting with three different fonts under different resolutions.

Font	Method	16 px	20 px	24 px	32 px	40 px
Times	Ours	68.19	79.43	86.92	95.67	96.44
	Baseline	55	57.12	61.15	65.77	69.42
Helvetica	Ours	71.27	77.98	81.25	93.90	95.47
	Baseline	55.76	57.69	59.81	62.69	67.12
Song	Ours	81.92	87.40	89.10	93.27	93.67
	Baseline	52.12	64.04	65.58	65.96	68.08

Table 4: Robustness ($\overline{ACC}$ (%)) against screen-camera shooting with three different fonts under different resolutions.

in their paper and directly use the released results therein. Since FontCode is designed only for high-resolution scenarios such as posters, the embedded information cannot be extracted at regular font sizes.

Robustness against Screen-Camera Shooting. We also demonstrate robustness against the screen-camera shooting distortions, which are mainly caused by screen down-sampling, light sources, moiré patterns, etc. As shown in Table 4, compared with the baseline, our method is more robust against the combined distortion. When the font size is small, Chinese glyphs perform better than English, which we at-

Display	Angles					Distances				
	-60°	-30°	0°	+30°	+60°	20 cm	30 cm	40 cm	50 cm	60 cm
Screen	89.68	93.27	94.23	93.99	90.17	75.38	95.45	90.59	93.03	94.48
Paper	94.31	94.02	96.20	93.96	93.41	96.70	94.23	94.62	92.85	92.03

Table 5: The average extraction accuracy ($\overline{ACC}$ (%)) on different shooting angles and distances.

Font	16 px	20 px	24 px	32 px	40 px
Times	83.76	92.96	99.21	99.97	99.98

Table 6: The extraction accuracy of our method on natural text in different resolutions. we use the text content from *The Lord of The Rings, Chapter 1* for the evaluation.

tribute to their complex structures, while English glyphs with simpler structures are susceptible to moiré patterns.

Impact of Camera Shooting Angles and Distances. In the former experiments, we fix the camera shooting angle (0°) and distance (30 cm), to evaluate the robustness against print/screen-camera shooting with different fonts under different font sizes. Here, we further conduct controlled experiments to comprehensively evaluate the impact of different camera shooting angles and distances. Specifically, we fix the font size as 32px and utilize Times New Roman for evaluation. Due to the noise-aware training in the first stage, we do not need to restore the perspective transformation introduced by the shooting process. Therefore, we first fix the shooting distance to 30 cm and calculate the $\overline{ACC}$ in different shooting angles. Then, we fix the shooting angle to 0° and calculate the $\overline{ACC}$ in different shooting distances. As Table 5 shows, our method performs well under different shooting angles and different distances in most cases. It is worth noting that a much closer distance to the screen will cause severe moiré patterns, inducing a performance degradation.

Performance on Natural Text. We further conduct experiments on natural text, where different glyphs appear with different frequencies. Similar to FontCode, we adopt the text content from *The Lord of The Rings, Chapter 1* to embed a randomly generated multi-bit message and calculate the extraction accuracy under screenshots. As shown in Table 6, the extraction accuracy on natural text is comparable with that tested by printing all the encoded glyphs in the same frequency, whose results are provided in Table 2.

Ablation Study

Importance of the Noise Layer. Here, we compare the encoded glyph images synthesized by the encoders trained with and without the noise layer. As shown in Figure 4, the encoder trained with the noise layer exactly modifies the glyph outline to hide information. We explain that the modifications on the glyph outline rather than in pixel values are more likely to be preserved when they pass through the noise

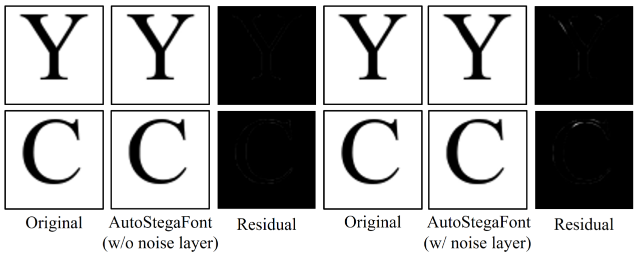
Original
AutoStegaFont (w/o noise layer)
Residual
Original
AutoStegaFont (w/ noise layer)
Residual

Figure 4: Glyph images encoded by the encoder trained with and without noise layer.

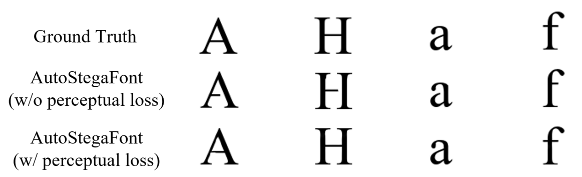
Ground Truth
AutoStegaFont (w/o perceptual loss)
AutoStegaFont (w/ perceptual loss)

Figure 5: Examples of the glyph images synthesized by the encoder trained with/without perceptual loss.

layer. In contrast, without the noise layer, the encoded information even cannot be transferred to the vector in Stage II. In other words, the noise layer not only ensures robustness against the distortions but also facilitates the mapping of the modifications on the image modality to the vector modality, killing two birds with one stone. We hope this perspective on the noise layer can be instructive for other tasks.

Importance of Perceptual Loss. Since glyph images are stylized, only using MSE loss to constrain the pixel-wise image quality may make the encoded glyph images less natural. Thus, we adopt perceptual loss to guarantee the style-level consistency between the encoded images and the original images. As Figure 5 shows, the encoder trained with perceptual loss can generate glyph images more naturally.

Conclusion

In this paper, we propose AutoStegaFont, a two-stage and dual-modality framework to automatically synthesize vector fonts for hiding information in text documents. Compared with existing methods based on manually designed fonts, our framework satisfies the requirements (*i.e.*, the automaticity, generalizability, and robustness) well and simultaneously for the first time, making it more practical to hide information in documents.

Acknowledgments

This work was supported in part by the Natural Science Foundation of China under Grant 62072421, 62002334, 62102386, 62121002 and U20B2047, and Students' Innovation and Entrepreneurship Foundation of USTC under Grant XY2022X01CY.

References

Abdelnabi, S.; and Fritz, M. 2021. Adversarial Watermarking Transformer: Towards Tracing Text Provenance with Data Hiding. In *42nd IEEE Symposium on Security and Privacy*.

Baek, Y.; Lee, B.; Han, D.; Yun, S.; and Lee, H. 2019. Character Region Awareness for Text Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9365–9374.

Baluja, S. 2017. Hiding images in plain sight: Deep steganography. *Advances in neural information processing systems*, 30.

Brassil, J. T.; Low, S.; and Maxemchuk, N. F. 1999. Copyright protection for the electronic distribution of text documents. *Proceedings of the IEEE*, 87(7): 1181–1196.

Campbell, N. D. F.; and Kautz, J. 2014. Learning a manifold of fonts. *ACM Transactions on Graphics (TOG)*, 33(4): 91:1–91:11.

Carlier, A.; Danelljan, M.; Alahi, A.; and Timofte, R. 2020. Deepsvg: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems*, 33: 16351–16361.

Gao, Y.; Guo, Y.; Lian, Z.; Tang, Y.; and Xiao, J. 2019. Artistic glyph image synthesis via one-stage few-shot learning. *ACM Transactions on Graphics (TOG)*, 38(6): 1–12.

Johnson, J.; Alahi, A.; and Fei-Fei, L. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, 694–711. Springer.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, T.-M.; Lukáč, M.; Michaël, G.; and Ragan-Kelley, J. 2020. Differentiable Vector Graphics Rasterization for Editing and Learning. *ACM Transactions on Graphics*, 39(6): 193:1–193:15.

Lopes, R. G.; Ha, D.; Eck, D.; and Shlens, J. 2019. A learned representation for scalable vector graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 7930–7939.

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*, 16 4: 2233–2249.

Rizzo, S.; Bertini, F.; and Montesi, D. 2016. Content-preserving Text Watermarking through Unicode Homoglyph Substitution. In *Proceedings of the 20th International Database Engineering & Applications Symposium*, 97–104.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Tancik, M.; Mildenhall, B.; and Ng, R. 2020. Stegastamp: Invisible hyperlinks in physical photographs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2117–2126.

Ueoka, H.; Murawaki, Y.; and Kurohashi, S. 2021. Frustratingly Easy Edit-based Linguistic Steganography with a Masked Language Model. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics*.

Wang, Y.; Gao, Y.; and Lian, Z. 2020. Attribute2font: Creating fonts you want from attributes. *ACM Transactions on Graphics (TOG)*, 39(4): 69–1.

Wang, Y.; and Lian, Z. 2021. DeepVecFont: synthesizing high-quality vector fonts via dual-modality learning. *ACM Transactions on Graphics (TOG)*, 40(6): 1–15.

Wengrowski, E.; and Dana, K. 2019. Light field messaging with deep photographic steganography. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1515–1524.

Xiao, C.; Zhang, C.; and Zheng, C. 2018. FontCode: Embedding Information in Text Documents using Glyph Perturbation. *ACM Transactions on Graphics (TOG)*, 37: 15:1–15:16.

Yang, X.; Zhang, J.; Chen, K.; Zhang, W.; Ma, Z.; Wang, F.; and Yu, N. 2022. Tracing Text Provenance via Context-Aware Lexical Substitution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 11613–11621.

Yoo, J.-C.; and Han, T. H. 2009. Fast normalized cross-correlation. *Circuits, systems and signal processing*, 28(6): 819–843.

Zhu, J.; Kaplan, R.; Johnson, J.; and Fei-Fei, L. 2018. HiDDeN: Hiding Data With Deep Networks. In *Proceedings of the 15th European Conference on Computer Vision*, 682–697.