Few-shot Compositional Font Generation with Dual Memory

Junbum Cha, Sanghyuk Chun, Gayoung Lee,
Bado Lee, Seonghyeon Kim, and Hwalsuk Lee

Clova AI Research, NAVER Corp.
{junbum.cha, sanghyuk.c, gayoung.lee,
bado.lee, kim.seonghyeon, hwalsuk.lee}@navercorp.com

Abstract. Generating a new font library is a very labor-intensive and time-consuming job for glyph-rich scripts. Despite the remarkable success of existing font generation methods, they have significant drawbacks; they require a large number of reference images to generate a new font set, or they fail to capture detailed styles with a few samples. In this paper, we focus on compositional scripts, a widely used letter system in the world, where each glyph can be decomposed by several components. By utilizing the compositionality of compositional scripts, we propose a novel font generation framework, named Dual Memory-augmented Font Generation Network (DM-Font), which enables us to generate a high-quality font library with only a few samples. We employ memory components and global-context awareness in the generator to take advantage of the compositionality. In the experiments on Korean-handwriting fonts and Thai-printing fonts, we observe that our method generates a significantly better quality of samples with faithful stylization compared to the state-of-the-art generation methods in quantitatively and qualitatively.

1 Introduction

Advances of web technology lead people to consume more and more text content on the web instead of their handwriting. Meanwhile, designing a new font style, such as personalized handwriting, is getting important for better user experience. However, because traditional methods to make a font library heavily rely on expert designers by manually design each glyph, creating a font library is extremely expensive for glyph-rich scripts such as Chinese (more than 50,000 glyphs), Korean (11,172 glyphs), or Thai (11,088 glyphs).

Recently, end-to-end automatic font generation methods [1,14,15,26,7,6] have been proposed to build a font set without human experts. The methods solve image-to-image translation tasks between various font styles based on generative adversarial networks (GANs) to generate a new font library. While the automatic font generation methods have shown the remarkable achievement in font generation tasks, they still require a large number of samples, *e.g.*, 775 samples [14,15]

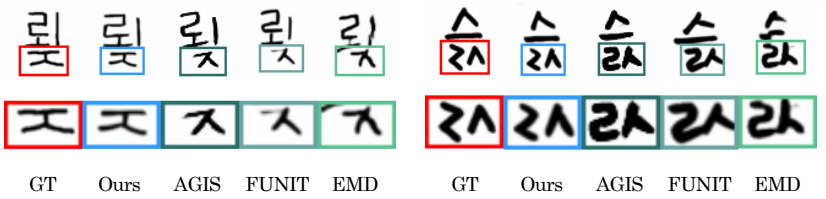


Fig. 1: **Few-shot font generation results.** While previous few-shot font generation methods (AGIS [8], FUNIT [24], and EMD [38]) are failed to generate unseen font, our model successfully transfer the font style and details.

to generate a new font set. Moreover, they require additional training procedures for each font style to create a new glyph set, *i.e.*, they need to finetune the pretrained model on the given new glyph subset, usually larger than 500. Thus, these finetune-based methods are rarely practical if collecting the target glyphs is extremely expensive, *e.g.*, human handwriting, or computing resources are limited, *e.g.*, on mobile devices.

Several recent studies attempt to generate a font set without additional training with a large number of glyphs, but using only a few samples [2,32,38,8,31]. Despite their successful few-shot generation performances on in-distributed styles, existing few-shot font generation methods often fail to generate high-quality font library using unseen style few-shot samples. We report example failure modes of existing few-shot methods in Figure 1 and the experiment section. We conjecture that it is because previous works rarely consider the inherent glyph characteristics, but solve the problem by end-to-end data-driven manner without any human prior. A few researchers have considered characteristics of glyphs to improve font generation methods [32,15], but their approaches are either still requiring more than 700 samples [15], or only designed for memory efficiency [32].

In this paper, we focus on a famous family of scripts, called *compositional scripts*, which are composed of a combination of sub-glyphs or components. For example, the Korean script has 11,172 valid glyphs with only 68 components. In this case, one can build the entire font library by designing only sub-glyphs and combine them by the rule. However, this rule-based method has a significant limitation; a sub-glyph changes its shape and position diversely depending on the combination, as shown in Figure 2. Hence, even if a user has a complete sub-glyphs, generating a full font set is impossible without the combination rule of components. Due to the limitations, compositional scripts have been manually designed for each glyph despite its compositionality.

Our framework for the few-shot font generation tasks explicitly utilizes the compositionality to more efficient and effective font generation. Our model, named Dual Memory-augmented Font Generation Network (DM-Font), learns the global combination recipe and the local component-wise styles from data. Unlike previous methods, we let DM-Font directly utilize local component-wise information from compositionality. In particular, we employ the dual-memory structure (*persistent memory* and *dynamic memory*) to efficiently capture the global glyph structure and the local component-wise styles, respectively. This

strategy enables us to generate a new high-quality font library with only a few samples, *e.g.*, 28 samples and 44 samples for Korean and Thai, respectively. Our experimental results on generating Korean handwritten and Thai printing fonts show both quantitatively better visual quality in various metrics and qualitatively being preferred in the user study.

2 Related Works

2.1 Few-Shot Image-to-Image Translation

Image-to-image translation [13,39] is a task which aims to learn the mapping between different domains. This mapping preserves the content in the source domain while changing the style in the source domain and applies the style of the target domain. Mainstream image-to-image translation works assumed an abundance of target training samples which is oftentimes, not plausible. To deal with more realistic scenarios where the target sample is scarce, few-shot image-to-image translation works appeared recently [24]. Image-to-image translation method can be directly applied as we view the font generation task as a translation task between the reference font domain and target font domain. We directly compare our method to FUNIT [24].

As an independent line of research, style transfer methods for artistic [9,20,12] and photorealistic [25,21,35] scenarios have been proposed to transfer styles of an unseen reference image while preserving the original content. Unlike image-to-image translation tasks, style transfer methods can not be directly transformed to font generation tasks, because they usually define the style as the set of textures and colors. However, in font generation tasks, the style of font is usually defined as discriminative local property of the font. Hence, our work does not concern style transfer methods as our baseline.

2.2 Automatic Font Generation

Automatic font generation task is an image-to-image translation task between different font domains, *i.e.*, styles. In the inference stage, the model is asked to generate a font with unseen style during the training. We categorize the automatic font generation methods into two classes according to way to generate a new font set many-shot and few-shot methods. Many-shot font generation methods [1,14,26,7,6,15] directly finetune the model on the subset of the target font set. The size of the subset is usually very large, *e.g.*, 755 samples. It is impractical in many real-world scenarios when collecting new glyphs is costly, *e.g.*, handwriting, or computing resources are limited, *e.g.*, mobile devices.

In contrast, few-shot font generation methods [38,31,2,8,32] does not require additional finetuning and large number of reference images. However, the existing few-shot methods have significant drawbacks. For example, some methods designed to generate a whole font set at the single forward path requires a huge model capacity [2,31]. Hence, they cannot be applied to glyph-rich scripts but

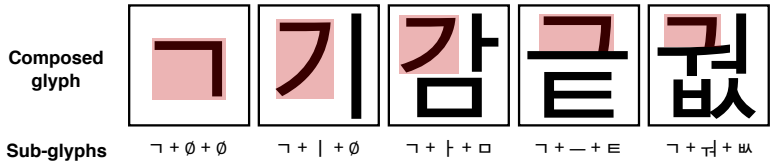


Fig. 2: **Examples of compositionality of Korean script.** Even if we choose the same sub-glyph, *e.g.*, "ㄱ", the shape and position of each sub-glyph are varying depending on the combination, as shown in red boxes.

scripts with a few glyphs, *e.g.*, Latin alphabet. SA-VAE [32] keep the model size small by introducing the Chinese character structure database. Some previous works, such as EMD [38] and AGIS-Net [8], proposed general image-to-image translation frameworks which can be applied to any general scripts. However, they show worse synthesizing quality to unseen style fonts, as observed in our experimental results.

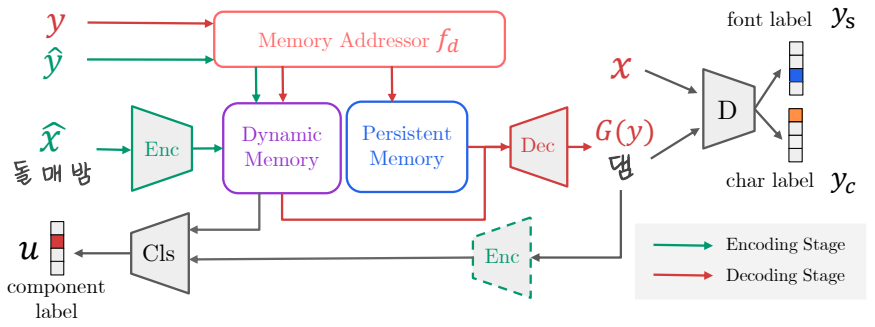
3 Preliminary: Complete Compositional Scripts

Compositional script is a widely-used glyph-rich script, where each glyph can be decomposed by several components as shown in Fig. 2. These scripts account for 24 of the top 30 popular scripts, including Chinese, Hindi, Arabic, Korean, Thai and so on. A compositional script is either *complete* or not, where each glyph in *complete compositional scripts* can be decomposed to fixed number sub-glyphs. For example, every Korean glyph can be decomposed by three sub-glyphs. Similarly, a Thai character has four components. Furthermore, complete compositional letters have specific sub-glyph sets for each *component type*. For example, the Korean alphabet has three component types where each component type has 19, 21, 28 sub-glyphs. By combining them, Korean letter has $19 \times 21 \times 28 = 11,172$ valid characters. Note that the minimum number of glyphs to get the entire sub-glyph set is 28. Similarly, Thai letter can represent $44 \times 7 \times 9 \times 4 = 11,088$ characters, and 44 characters are required to cover whole sub-glyphs.

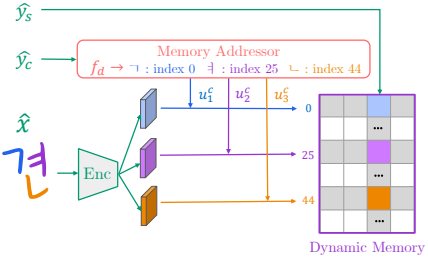
Some compositional scripts are not complete. For example, each character of the Chinese letter can be decomposed into a diverse number of sub-glyphs. Although we mainly validate our method on Korean and Thai scripts, our method can be easily extended to other compositional scripts, *e.g.*, Chinese letter system.

4 Dual Memory-augmented Font Generation Network

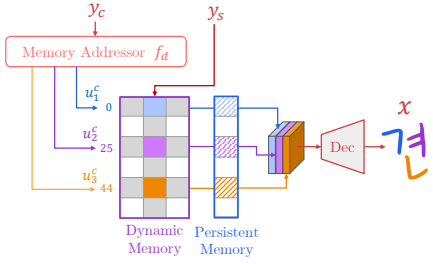
In this section, we introduce a novel architecture, Dual Memory-augmented Font Generation Network (DM-Font), which utilizes the compositionality of a script by the augmented dual memory structure. DM-Font disentangles global composition information and local styles, and writes them into persistent and dynamic memory, respectively. It enables to make a high-quality full glyph library only with very few references, *e.g.*, 28 samples for Korean, 44 samples for Thai.



(a) Architecture overview.



(b) Encoding phase detail.



(c) Decoding phase detail.

Fig. 3: **Visual descriptions of the proposed DM-Font.** (a) The model encodes the reference style glyphs and stores it into the memory module. After the encoder fills up the memory, the decoder generates the glyphs of the target style. (b) The encoder produces the component-wise features and stores it into the dynamic memory using the component addresses u_i^c and the style label $\hat{y}_s$. (c) The memory addressor loads the component-wise features by the character label y_c and feeds them to the decoder.

4.1 Architecture Overview

We illustrate the architecture overview of DM-Font in Fig. 3a. We also provide more details in Appendix. The generation process of the model consists of two stages: an encoding stage and a decoding stage. In the encoding stage, the reference style glyphs are encoded to the component features and stored into the dynamic memory. After the encoder fills up the memory, the decoder fetches the component features and generates the target glyph according to the target character label y_c. The detail of each module is described below.

Encoder *Enc* disassembles a source glyph into the several components and encodes them to the component features. We adopt multi-head structure, one head per one component type. The encoded component-wise features are written into the dynamic memory as shown in Figure 3b.

We employ two memory modules, where **persistent memory** is a component-wise learned embedding that represents the intrinsic shape of each component and the global information of the script such as the compositionality, while **dynamic memory** stores encoded component features of the given reference glyphs. Hence, persistent memory captures the global information of sub-glyphs independent to each font style, while encoded features in dynamic memory learn unique local styles depending on each font. We provide detailed analysis of each memory in the experiments. **Memory addressor** provides the access address of both dynamic and persistent memory based on the given character label y_c as shown in Figure 3b and Figure 3c. We use pre-defined decomposition function $f_d : y_c \mapsto \{u_i^c \mid i = 1 \dots M_c\}$ to get the component-wise address, where u_i^c is the label of i-th component of y_c, and M_c is the number of sub-glyphs for y_c. More detailed examples are given in Appendix.

The component-wise encoded features for the reference $\hat{x}$, whose character label is $\hat{y}_c$ and style label is $\hat{y}_s$, are stored into dynamic memory during the encoding stage. In our scenario, the encoder Enc is a multi-head encoder, and $\hat{y}_c$ can be decomposed by $f_d(\hat{y}_c)$ to sub-glyph labels $\hat{u}_i^c$. Hence, the features in dynamic memory at address $(\hat{u}_i^c, \hat{y}_s)$, $DM(\hat{u}_i^c, \hat{y}_s)$ can be computed by $Enc_i(\hat{x})$, where i is the index of the component type of $\hat{u}_i^c$, and Enc_i is the encoder output corresponding to i.

In the decoding stage, **decoder** Dec generates a target glyph $G(y_c, y_s)$ with the target character y_c and the reference style y_s by reading the target component-wise features from the dynamic memory DM and the persistent memory PM as the following:

$$G(y_c, y_s) = Dec\big(\big[DM(u_i^c, y_s),\, PM(u_i^c) \mid u_i^c \in f_d(y_c)\big]\big), \tag{1}$$

where $[x_0, \dots, x_n]$ refers to the concatenation operation. Figure 3c shows an example of the decoding phase.

For the better generation quality, we also employ a discriminator and a component classifier. For **discriminator** D, we use a multitask discriminator [27,24] with the font condition and the character content condition. The multitask discriminator has independent branches for each target class and each branch performs binary classification. Considering two types of conditions, we use two multitask discriminator, one for character classes and the other for font classes, with a shared backbone. We further use **component classifier** Cls to ensure the model to fully utilize the compositionality. The component classifier provides additional supervision to the generator that stabilizes the training process.

Moreover, we incorporate the global-context awareness and local-style preservation into the generator, called **compositional generator**. Specifically, self-attention blocks [5,36] are used in the encoder to facilitate relational reasoning between components, and the hourglass block [30,23] is attached to the decoder to aware global-context while preserving locality. The dynamic memory handles the low-level features in addition to the high-level features to capture the details of local style from the reference images. In the experiment section, we analyze the

impact of the architectural improvements on the final performance. We further provide the implementation details in Appendix.

4.2 Learning

We train DM-Font from font sets $(x, y_c, y_f) \sim \mathcal{D}$, where x is a target glyph image, y_c and y_f is a character and font label, respectively. During the training, we assume that different font labels represent different style, *i.e.*, we set $y_s = y_f$ in equation (1). Also, for the efficiency, we only encode a core component subset to compose the target glyph x into the dynamic memory, instead of the full component set. For example, the Korean script has the full component set with size 68, but to construct a single character, only 3 components are required.

We use the **adversarial loss** to let the model generate plausible images.

$$\mathcal{L}_{adv} = \mathbb{E}_{x,y}\left[\log D_y(x)\right] + \mathbb{E}_{x,y}\left[\log(1 - D_y(G(y_c, y_f))\right], \tag{2}$$

where G generates an image $G(y_c, y_f)$ from the given image x and target label y by equation (1). The discriminator D_y is conditional on the target label y. We employed two types of the discriminator to solve the problem. The font discriminator is a conditional discriminator on the source font index, while the character discriminator aims to classify what is the given character.

$\mathbf{L_1}$ loss is employed to add supervision from the ground truth target glpyh x as the following:

$$\mathcal{L}_{l1} = \mathbb{E}_{x,y}\left[\|x - G(y_c, y_f)\|_1\right]. \tag{3}$$

We also use the discriminator **feature matching loss** to improve the stability of the training. The feature matching loss is constructed using the output from the l-th layer of the discriminator, $D_f^{(l)}$.

$$\mathcal{L}_{feat} = \mathbb{E}_{x,y}\left[\frac{1}{L}\sum_{l=1}^{L}\|D_f^{(l)}(x) - D_f^{(l)}(G(y_c, y_f))\|_1\right], \tag{4}$$

where L denotes the number of feature layers of the discriminator.

Lastly, to let the model fully utilize the compositionality, we train the model with additional **component-classification loss**. For the given input x, we extract the component-wise features using the encoder Enc, and train them with cross-entropy loss (CE) using component labels $u \in f_d(y_c)$, where f_d is the component decomposition function to the given character label y_c.

$$\mathcal{L}_{cls} = \mathbb{E}_{x,y}\left[\sum_{u_i^c \in f_d(y_c)} \mathrm{CE}(Enc_i(x), u_i^c)\right] + \mathbb{E}_{y}\left[\sum_{u_i^c \in f_d(y_c)} \mathrm{CE}(Enc_i(G(y_c, y_f)), u_i^c)\right]. \tag{5}$$

The final objective function to optimize the generator G, the discriminator D, the component classifier C as the following:

$$\min_{G,C}\max_{D} \mathcal{L}_{adv(font)} + \mathcal{L}_{adv(char)} + \lambda_{l1}\mathcal{L}_{l1} + \lambda_{feat}\mathcal{L}_{feat} + \lambda_{cls}\mathcal{L}_{cls}, \tag{6}$$

where $\lambda_{l1}, \lambda_{feat}, \lambda_{cls}$ are control parameters to importance of each objective compared to the adversarial loss.

5 Experiments

We empirically validate DM-Font on Korean and Thai scripts and compare with state-of-the-art few-shot font generation methods.

5.1 Datasets

Korean-handwriting Dataset. Due to its diversity and data sparsity, generating a handwritten font with a few samples is challenging. We validate the models using 86 Korean-handwriting fonts[1], refined by the expert designer. Each font library contains $2,448$ widely-used Korean glyphs which is only $2,448/11,172 = 21\%$ of the whole characters. We train the models using 80% fonts and 90% characters, and validate the models on the remaining 20% font split. Also, to measure the generalizability to the unseen characters, we separately evaluate the models on the 90% seen characters and the 10% unseen characters for the validation font split. We choose 30 samples to recover a whole font set for all experiments. As a final product, we generate $11,172$ characters for each font library.

Thai-printing Dataset. We also demonstrate the models on 105 Thai printing fonts[2]. Thai letters are much complex than Korean letters, because Thai character systems are composed of four sub-glyphs while Korean letters have three components. We apply the same train-evaluation split strategy of the Korean-handwriting dataset, and 44 samples are used for the few-shot generation.

Korean-unrefined Dataset. We also gather unrefined Korean handwriting dataset from 88 non-experts, letting each applicant write 150 characters. This dataset is extremely diverse and not refined by expert artists different from the Korean-handwriting dataset. We use the Korean-unrefined dataset as the validation of the models trained on the Korean-handwriting dataset, *i.e.*, the Korean-unrefined dataset is not visible during the training, but only a few samples are visible for the evaluation. In our experiments, we measure the robustness to out-of-distributed styles for each few-shot font generation methods. We use 30 samples for the generation as well as the Korean-handwriting dataset.

5.2 Comparison Methods and Evaluation Metrics

Comparison Methods. We compare our model with state-of-the-art few-shot font generation methods, including EMD [38], AGIS-Net [8], and FUNIT [24]. We exclude the methods which are Chinese-specific [32] or not applicable to

[1] We collect public fonts from http://uhbeefont.com/.
[2] We use https://github.com/jeffmcneill/thai-font-collection.

Table 1: **Quantatitive Evaluation on the Korean-handwriting dataset.** We evaluate the methods on the seen and unseen character sets. Higher is better, except perceptual distance (PD) and mFID.

	Pixel-level		Content-aware			Style-aware		
	SSIM	MS-SSIM	Acc(%)	PD	mFID	Acc(%)	PD	mFID
Evaluation on the **seen** character set during training								
EMD [38]	0.691	0.361	80.4	0.084	138.2	5.1	0.089	134.4
FUNIT [24]	0.686	0.369	94.5	0.030	42.9	5.1	0.087	146.7
AGIS-Net [8]	0.694	0.399	**98.7**	**0.018**	23.9	8.2	0.088	141.1
DM-Font (ours)	**0.704**	**0.457**	98.1	**0.018**	**22.1**	**64.1**	**0.038**	**34.6**
Evaluation on the **unseen** character set during training								
EMD [38]	0.696	0.362	76.4	0.095	155.3	5.2	0.089	139.6
FUNIT [24]	0.690	0.372	93.3	0.034	48.4	5.6	0.087	149.5
AGIS-Net [8]	0.699	0.398	98.3	0.019	25.9	7.5	0.089	146.1
DM-Font (ours)	**0.707**	**0.455**	**98.5**	**0.018**	**20.8**	**62.6**	**0.039**	**40.5**

glyph-rich scripts [31]. Here, we slightly modified FUNIT, originally designed for unsupervised translation, by changing its reconstruction loss to L_1 loss with ground truths and conditioning the discriminator to both contents and styles.

Evaluation Metrics. Assessing a generative model is difficult because of its non-tractability. Several quantitative evaluation metrics [16,11,37,29] have attempted to measure the performance of the trained generative model with different assumptions, but it is still controversial what it the best evaluation methods for generative models. In this paper, we consider three diverse levels of evaluation metrics; pixel-level, perceptual-level and human-level evaluations.

Pixel-level evaluation metrics assess the pixel structural similarity between the ground truth image and the generated image. We employ the structural similarity index (SSIM) and multi-scale structural similarity index (MS-SSIM).

However, pixel-level metrics often disagree with human perceptions. Thus, we also evaluate the models with **perceptual-level evaluation metrics**. We trained four ResNet-50 [10] models on the Korean-handwriting dataset and Thai-printing dataset to classify style and character label. Here, unlike the generation task, we use the whole fonts to the training. We denote a metric is *context-aware* if the metric is performed using the content classifier, and *style-aware* is defined similarly. Note that these classifiers are not accessible from the font generation models during the training, but only used for the evaluation. We report the top-1 accuracy, perceptual distance (PD) [16,37], and mean FID (mFID) [24] using the trained classifiers. The perceptual distance is computed by L_2 distance of the features between generated glyph and GT glyph, and mFID is a conditional Fréchet Inception Distance (FID) [11] by averaging FID for each target class.

Finally, we conduct a user study on the Korean-unrefined dataset for measuring **human-level evaluation metric**. We ask users about three types of

Table 2: **Quantatitive Evaluation on the Thai-printing dataset.** We evaluate the methods on the seen and unseen character sets. Higher is better, except perceptual distance (PD) and mFID.

	Pixel-level		Content-aware			Style-aware		
	SSIM	MS-SSIM	Acc(%)	PD	mFID	Acc(%)	PD	mFID
Evaluation on the **seen** character set during training								
EMD [38]	0.773	0.640	86.3	0.115	215.4	3.2	0.087	172.0
FUNIT [24]	0.712	0.449	45.8	0.566	1133.8	4.6	0.084	167.9
AGIS-Net [8]	0.758	0.624	**87.2**	**0.091**	**165.2**	15.5	0.074	145.2
DM-Font (ours)	**0.776**	**0.697**	87.0	0.103	198.7	**50.3**	**0.037**	**69.4**
Evaluation on the **unseen** character set during training								
EMD [38]	0.770	0.636	85.0	0.123	231.0	3.4	0.087	171.6
FUNIT [24]	0.708	0.442	45.0	0.574	1149.8	4.7	0.084	166.9
AGIS-Net [8]	0.755	0.618	85.4	0.103	**188.4**	15.8	0.074	145.1
DM-Font (ours)	**0.773**	**0.693**	**87.2**	**0.101**	195.9	**50.6**	**0.037**	**69.6**

preference: content preference, style preference, and user preference considering both content and style. The questionnaire is made of 90 questions, 30 for each preference. Each question shows 40 glyphs, consisting of 32 glyphs generated by four models and 8 GT glyphs. We collect total 3,420 responses from 38 Korean natives.

5.3 Main Results

Quantitative Evaluation. The main results on Korean-handwriting and Thai-printing datasets are reported in Table 1 and Table 2, respectively. We also report the evaluation results on the Korean-unrefined dataset in Appendix. We follow the dataset split introduced in Section 5.1. In the experiments, DM-Font remarkably outperforms the previous methods in most of evaluation metrics, especially on style-aware benchmarks. Baseline methods show slightly worse content-aware performances on unseen characters than seen characters, *e.g.*, AGIS-Net shows worse content-aware accuracy (98.7 $\rightarrow$ 98.3), PD (0.018 $\rightarrow$ 0.019), and mFID (23.9 $\rightarrow$ 25.9) in Table 1. In contrast, DM-Font consistently shows better generalizability to the unobserved characters during the training for both datasets. It is because our model interprets a glyph at the component level, the model easily extrapolates the unseen characters from the learned component-wise features stored in memory modules.

Our method shows significant improvements in style-aware metrics. DM-Font achieves 62.6% and 50.6% accuracy while other methods show much less accuracy, *e.g.*, about 5% for Korean unseen and Thai unseen character sets, respectively. Likewise, the model shows dramatic improvements in perceptual distance and mFID as well as the accuracy measure. In the latter section, we provide

(a) Seen character set during training.

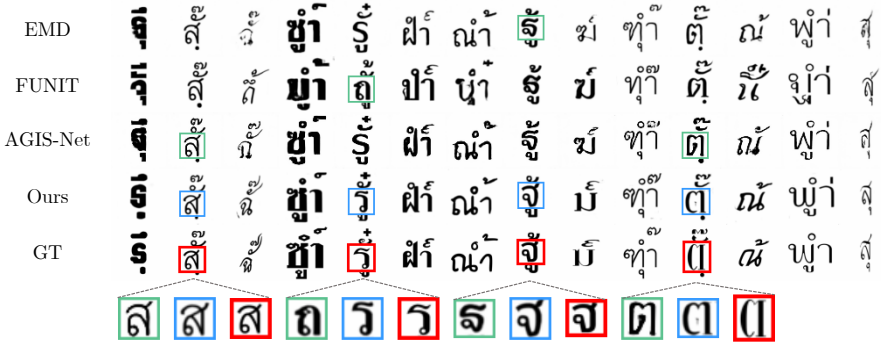
(b) Unseen character set during training.

Fig. 4: **Qualitative comparison on the Korean-handwriting dataset.** Visualization of generated samples with seen and unseen characters. We show insets of baseline results (green box), ours (blue box) and ground truth (red box). Ours successfully transfers the detailed style of the target style, while baselines fail to generate glyphs with the detailed reference style.

more detailed analysis that the baseline methods are overfitted to the training styles and failed to generalize to unseen styles.

Qualitative Comparison. We also provide visual comparisons in Figure 4 and Figure 5, which contain various challenging fonts including thin, thick, and curvy fonts. Our method generates glyphs with consistently better visual quality than the baseline methods. EMD [38] often erases thin fonts unintentionally, which causes low content scores compared to the other baseline methods. FUNIT [24] and AGIS-Net [8] accurately generate the content of glyphs and capture global styles well including overall thickness and font sizes. However, the detailed styles of the components in their results look different from the ground truths. Moreover, some generated glyphs for unseen Thai style lose the original content (see the difference between green boxes and red boxes in Figure 4 and Figure 5 for more details). Compared to the baselines, our method generates the most plau-

(a) Seen character set during training.



(b) Unseen character set during training.

Fig. 5: **Qualitative comparison on the Thai-printing dataset.** Visualization of generated samples with seen and unseen characters. We show insets of baseline results (green box), ours (blue box) and ground truth (red box). Overall, ours faithfully transfer the target style, while other methods even often fail to preserve contents in unseen character sets.

sible images in terms of global font styles and detailed component styles. These results show that our model preserves details in the components using the dual memory and reuse them to generate a new glyph.

User Study. We conduct a user study to further evaluate the methods in terms of human preferences using the Korean-unrefined dataset. Some example generated glyphs are illustrated in Figure 6 and Appendix. Users are asked to choose the most preferred generated samples in terms of content preserving, faithfulness to the reference style, and personal preference. The results are shown in Table 3, which present similar intuitions with Table 1; AGIS-Net and our method are comparable in the content evaluation, and our method is dominant in the style preference.

Table 3: **User study results on the Korean-unrefined dataset.** Each number is the preferred model output out of 3,420 responses.

	EMD [38]	FUNIT [24]	AGIS-Net [8]	DM-Font (ours)
Best content preserving	1.33%	9.17%	**48.67%**	40.83%
Best stylization	1.71%	8.14%	17.44%	**72.71%**
Most preferred	1.23%	9.74%	16.40%	**72.63%**

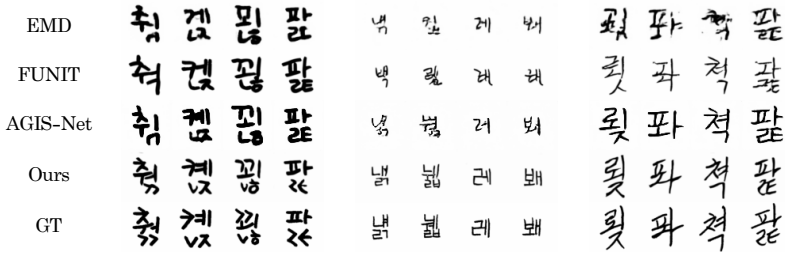


Fig. 6: Samples for the user study. The Korean-unrefined dataset is used.

Table 4: **Ablation studies on the Korean-handwriting dataset.** Each content and style score is an average of the seen and unseen accuracies. Hmean denotes the harmonic mean of content and style scores.

(a) Impact of the memory modules.

	Content	Style	Hmean
Baseline	96.6	6.5	12.2
+ Dynamic memory	**99.8**	32.0	48.5
+ Persistent memory	97.6	46.2	62.8
+ Compositional G	98.3	**63.3**	**77.0**

(b) Impact of the objective functions.

	Content	Style	Hmean
Full	**98.3**	**63.3**	**77.0**
Full $-\mathcal{L}_{l1}$	97.3	53.8	69.3
Full $-\mathcal{L}_{feat}$	97.8	51.3	67.3
Full $-\mathcal{L}_{cls}$	3.1	16.0	5.2

5.4 More Analysis

Ablation Study. We investigate the impact of our design choices by ablative studies. Table 4a shows that the overall performances are improved by adding proposed components such as the dynamic memory, persistent memory, and global-context awareness to the generator.

Here, the baseline method is similar to FUNIT whose content and style accuracies are 93.9 and 5.4, respectively. The baseline suffers from the failure of style generalization like previous methods. We observe that the dynamic memory and persistent memory dramatically improves style scores while almost preserving content scores. Finally, our architectural improvements bring the best performance.

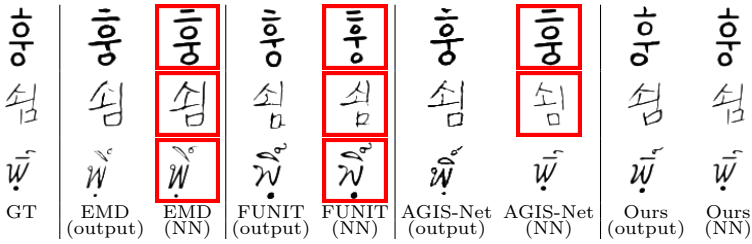


Fig. 7: **Nearest neighbor analysis.** We report the generated images by each model (output) with the given unseen reference style (GT) and the ground truth samples whose label is predicted by the style classifier (NN). Red boxed samples denote training samples. We can conclude that the baseline methods are overfitted to the training style while ours easily generalizes to unseen style.

We also explore the performance influence of each objective. As shown in Table 4b, removing L_1 loss and feature matching loss slightly degrades performances. The component-classification loss, which enforces the compositionality to the model, is the most important factor for successful training.

Style Overfitting of Baselines. We further analyze the generated glyphs using our style classifier to investigate the style overfitting of the baseline methods. Figure 7 shows the predicted classes for each model output. We observe that the baseline methods often generate samples similar to the training samples, which clearly indicates the overfitting in the font styles. On the other hand, our model avoids the overfitting in the font styles by learning the compositionality of glyphs and directly reusing components of inputs. Consequently, as supported by previous quantitative and qualitative evaluations, our model is robust to the out-of-distributed font generation compared to the existing methods.

Component-wise Style Mixing. In Figure 8, we demonstrate our model can interpolate styles in component-wise. It supports that our model fully utilizes the compositionality to generate a glyph.

6 Conclusions

Previous few-shot font generation methods often fail to generalize to unseen styles. In this paper, we propose a novel few-shot font generation framework for compositional scripts, named Dual Memory-augmented Font Generation Network (DM-Font). Our method effectively incorporates the prior knowledge of compositional script into the framework via two external memories: the dynamic memory and the persistent memory. The experimental results showed that the existing methods fail in stylization on unseen fonts, while DM-Font remarkably and consistently outperforms the existing few-shot font generation methods on

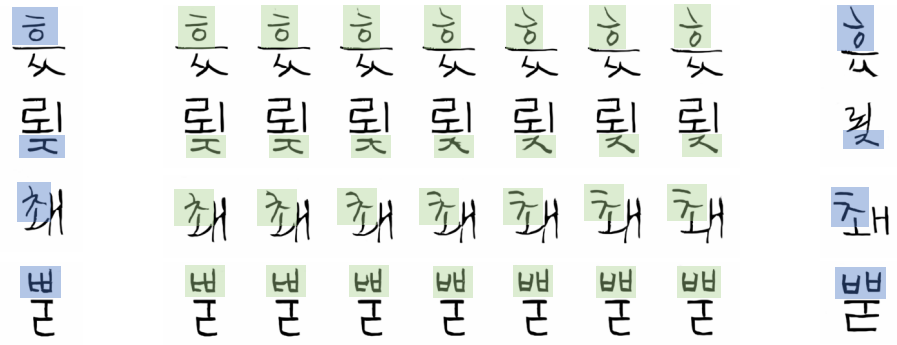


Fig. 8: **Component-wise style mixing.** We interpolate only one component (marked by blue boxes) between two glyphs (the first column and the last column). The interpolated sub-glyphs are marked by green boxes. Our model successfully interpolates two sub-glyphs, while preserving other local styles.

Korean and Thai letters. Extensive empirical evidence support that our framework lets the model fully utilize the compositionality so that the model can produce high-quality samples with only a few samples.

References

1. zi2zi: Master chinese calligraphy with conditional adversarial networks, https://github.com/kaonashi-tyc/zi2zi 1, 3
2. Azadi, S., Fisher, M., Kim, V.G., Wang, Z., Shechtman, E., Darrell, T.: Multi-content gan for few-shot font style transfer. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018) 2, 3
3. Bello, I., Zoph, B., Vaswani, A., Shlens, J., Le, Q.V.: Attention augmented convolutional networks. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV). pp. 3286–3295 (2019) 19
4. Brock, A., Donahue, J., Simonyan, K.: Large scale GAN training for high fidelity natural image synthesis. In: International Conference on Learning Representations (ICLR) (2019) 20
5. Cao, Y., Xu, J., Lin, S., Wei, F., Hu, H.: Gcnet: Non-local networks meet squeeze-excitation networks and beyond. In: IEEE International Conference on Computer Vision Workshops (ICCVW) (2019) 6, 18
6. Chang, B., Zhang, Q., Pan, S., Meng, L.: Generating handwritten chinese characters using cyclegan. In: IEEE Winter Conference on Applications of Computer Vision (WACV) (2018) 1, 3
7. Chang, J., Gu, Y., Zhang, Y., Wang, Y.F.: Chinese handwriting imitation with hierarchical generative adversarial network. In: British Machine Vision Conferenc (BMVC) (2018) 1, 3
8. Gao, Y., Guo, Y., Lian, Z., Tang, Y., Xiao, J.: Artistic glyph image synthesis via one-stage few-shot learning. ACM Transactions on Graphics (TOG) (2019) 2, 3, 4, 8, 9, 10, 11, 13, 20

9. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2414–2423 (2016) 3
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 9
11. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: Advances in Neural Information Processing Systems (NeurIPS) (2017) 9, 20
12. Huang, X., Belongie, S.J.: Arbitrary style transfer in real-time with adaptive instance normalization. In: IEEE International Conference on Computer Vision (ICCV) (2017) 3
13. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017) 3
14. Jiang, Y., Lian, Z., Tang, Y., Xiao, J.: Dcfont: an end-to-end deep chinese font generation system. In: SIGGRAPH Asia (2017) 1, 3
15. Jiang, Y., Lian, Z., Tang, Y., Xiao, J.: Scfont: Structure-guided chinese font generation via deep stacked networks. In: AAAI Conference on Artificial Intelligence (AAAI) (2019) 1, 2, 3
16. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: European Conference on Computer Vision (ECCV) (2016) 9
17. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of GANs for improved quality, stability, and variation. In: International Conference on Learning Representations (ICLR) (2018) 20
18. Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019) 20
19. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (ICLR) (2015) 20
20. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.H.: Universal style transfer via feature transforms. In: Advances in Neural Information Processing Systems (NeurIPS) (2017) 3
21. Li, Y., Liu, M.Y., Li, X., Yang, M.H., Kautz, J.: A closed-form solution to photorealistic image stylization. In: European Conference on Computer Vision (ECCV) (2018) 3
22. Lim, J.H., Ye, J.C.: Geometric gan. arXiv e-prints (2017) 20
23. Lin, T.Y., Dollar, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017) 6, 18
24. Liu, M.Y., Huang, X., Mallya, A., Karras, T., Aila, T., Lehtinen, J., Kautz, J.: Few-shot unsupervised image-to-image translation. In: IEEE International Conference on Computer Vision (ICCV) (2019) 2, 3, 6, 8, 9, 10, 11, 13, 20
25. Luan, F., Paris, S., Shechtman, E., Bala, K.: Deep photo style transfer. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017) 3
26. Lyu, P., Bai, X., Yao, C., Zhu, Z., Huang, T., Liu, W.: Auto-encoder guided gan for chinese calligraphy synthesis. In: International Conference on Document Analysis and Recognition (ICDAR) (2017) 1, 3
27. Mescheder, L., Geiger, A., Nowozin, S.: Which training methods for gans do actually converge? In: International Conference on Machine Learning (ICML) (2018) 6

28. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. In: International Conference on Learning Representations (ICLR) (2018) 20
29. Naeem, M.F., Oh, S.J., Uh, Y., Choi, Y., Yoo, J.: Reliable fidelity and diversity metrics for generative models. arXiv e-prints (2020) 9
30. Newell, A., Yang, K., Deng, J.: Stacked hourglass networks for human pose estimation. In: European Conference on Computer Vision (ECCV) (2016) 6, 18
31. Srivatsan, A., Barron, J., Klein, D., Berg-Kirkpatrick, T.: A deep factorization of style and structure in fonts. In: Conference on Empirical Methods in Natural Language Processing (EMNLP) (2019) 2, 3, 9
32. Sun, D., Ren, T., Li, C., Su, H., Zhu, J.: Learning to write stylized chinese characters by reading a handful of examples. In: International Joint Conference on Artificial Intelligence (IJCAI) (2018) 2, 3, 4, 8
33. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems (NeurIPS). pp. 5998–6008 (2017) 18
34. Yazıcı, Y., Foo, C.S., Winkler, S., Yap, K.H., Piliouras, G., Chandrasekhar, V.: The unusual effectiveness of averaging in GAN training. In: International Conference on Learning Representations (ICLR) (2019) 20
35. Yoo, J., Uh, Y., Chun, S., Kang, B., Ha, J.W.: Photorealistic style transfer via wavelet transforms. In: International Conference on Computer Vision (ICCV) (2019) 3
36. Zhang, H., Goodfellow, I., Metaxas, D., Odena, A.: Self-attention generative adversarial networks. In: International Conference on Machine Learning (ICML) (2019) 6, 18, 20
37. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018) 9
38. Zhang, Y., Zhang, Y., Cai, W.: Separating style and content for generalized style transfer. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018) 2, 3, 4, 8, 9, 10, 11, 13, 20
39. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: IEEE International Conference on Computer Vision (ICCV) (2017) 3

A Network Architecture and Details

A.1 Memory Addressors

Algorithm 1: Unicode-based Korean letter decomposition function

Input: A character label y_c
Output: Component labels u_1^c, u_2^c, and u_3^c
Data: The number of components for each i-th component type N_i.
unicode = `ToUnicode(`y_c`)`
`// 0xAC00 is the initial Korean Unicode`
code = unicode - 0xAC00
u_3^c = code `mod` N_3
u_2^c = (code `div` N_3) `mod` N_2
u_1^c = code `div` ($N_3 \times N_2$)

The memory addressor converts character label y_c to the set of component labels u_i^c by the pre-defined decomposition function $f_d : y_c \mapsto \{u_i^c \mid i = 1 \dots M_c\}$, where u_i^c is the label of i-th component of y_c and M_c is the number of sub-glyphs for y_c. In this paper, we employ Unicode-based decomposition functions specified to each language. For example, the Korean decomposition function is described in Algorithm 1. The Korean decomposition function disassembles a character Unicode into component labels by the pre-defined rule. On the other hand, each Thai character consists of several Unicodes, each of which corresponds to one component. Therefore, each Unicode constituting the letter is a label itself. The Thai decomposition function only needs to determine the component type of each Unicode.

A.2 Network Architecture

We design the network architecture focusing on two properties: global-context awareness and local-style preservation. Global-context awareness allows the relational reasoning between components to the network, boosting to disassemble source glyphs into sub-glyphs and assemble the sub-glyphs to the target glyph. Local-style preservation indicates that the local style of source glyph is reflected in the target glyph.

For the global-context awareness, the encoder adopts global-context block (GCBlock) [5] and self-attention block (SABlock) [36,33], and the decoder employs hourglass block (HGBlock) [30,23]. These blocks extend the receptive field globally and facilitate relational reasoning between components while preserving locality. For the local-style preservation, the network handles multi-level features based on the dual memory framework. The specific architecture overview is described visually in Figure 9.

Concretely, the generator consists of five types of blocks; convolution block (ConvBlock), residual block (ResBlock), self-attention block, global-context block, and hourglass block. Our self-attention block is adopted from Transformer [33] instead of SAGAN [36]. That is, the block consists of multi-head self-attention

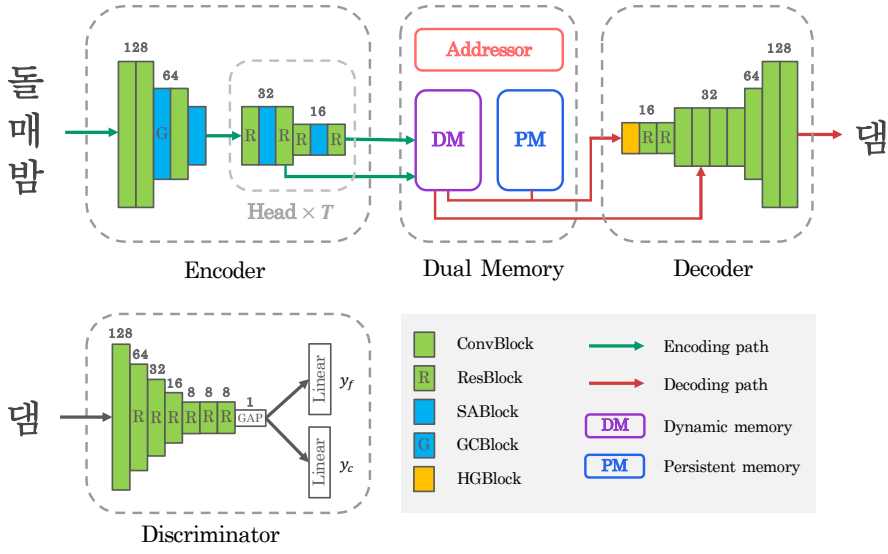


Fig. 9: DM-Font architecture overview. The encoder holds multiple heads according to the number of component types T. The number above each block denotes the spatial size of the input feature map.

and position-wise feed-forward. We also use two-dimensional relative positional encoding from [3]. The hourglass block consists of multiple convolution blocks and downsampling or upsampling operation follows each block. Through hourglass structure, the spatial size of the feature map is reduced to 1×1 and restored to the original size, which extends the receptive field globally preserving locality. The channel size starts at 32 and doubles as blocks are added, up to 256 for the encoder and 512 for the decoder.

The discriminator structure is relatively simple. Several residual blocks follow the first convolution block. Similar to the generator, the channel size starts at 32 and doubles as blocks are added, up to 1024. The output feature map of the last residual block is spatially squeezed to 1×1 size and it is fed to the two linear, font and character discriminators. Each discriminator is a multi-task discriminator that performs binary classification for each target class. Therefore, the font discriminator produces $|\mathbb{Y}_f|$ binary outputs and the character discriminator produces $|\mathbb{Y}_c|$ binary outputs, where $|\mathbb{Y}|$ denotes the number of target classes.

Since the persistent memory is independent to local styles, we set the size of persistent memory identical to the size of high-level features, the final output of the encoder, *i.e.*, 16×16. The learned embedding is refined via three convolution blocks, added to the high-level features of dynamic memory, and then fed to the decoder. The component classifier comprises two residual blocks and one linear layer and identifies the class of the high-level component features from the dynamic memory.

A.3 Implementation Details

The optimization coefficients are set to $\lambda_{l1} = 0.1$, $\lambda_{feat} = 1.0$, and $\lambda_{cls} = 0.1$. We use the Adam optimizer [19] with a learning rate of 2e-4 for the generator and 8e-4 for the discriminator, following the two time-scale update rule [11]. The component classifier use same learning rate with the generator. The discriminator adopts spectral normalization [28] for the regularization. We train the model with hinge GAN loss [36,28,4,24,22] during a total of 200K iterations. For evaluation, we employ exponential moving average of the generator [17,34,24,18]. For the Thai-printing dataset, we only adjust the learning rates and the training iterations. We use a learning rate of 5e-5 for the generator and 1e-4 for the discriminator with 250K training iterations.

B Additional Results

B.1 Korean-unrefined Dataset

Table 5: **Quantatitive Evaluation on the Korean-unrefined dataset.** Higher is better, except perceptual distance (PD) and mFID.

	Pixel-level		Content-aware		Style-aware	
	SSIM	MS-SSIM	PD	mFID	PD	mFID
EMD [38]	0.716	0.340	0.106	99.2	0.079	93.3
FUNIT [24]	0.711	0.311	0.080	87.0	0.066	79.4
AGIS-Net [8]	0.708	0.334	0.052	67.2	0.089	134.5
DM-Font (ours)	**0.726**	**0.387**	**0.048**	**46.2**	**0.046**	**31.5**

Table 5 shows the quantitative evaluation results of the Korean-unrefined dataset used for the user study. Since the Korean-unrefined dataset is an evaluation only dataset, the classifiers trained with the Korean-handwriting dataset are reused to compute the perceptual distance and mFID. DM-Font consistently shows the remarkable performance, despite the different characteristics of datasets. The visual samples are shown in Figure 10.

B.2 Ablation Study

Table 6 shows the full ablation study results. We report only averaged accuracy in the body text for the sake of brevity, but every result shows similar behavior.

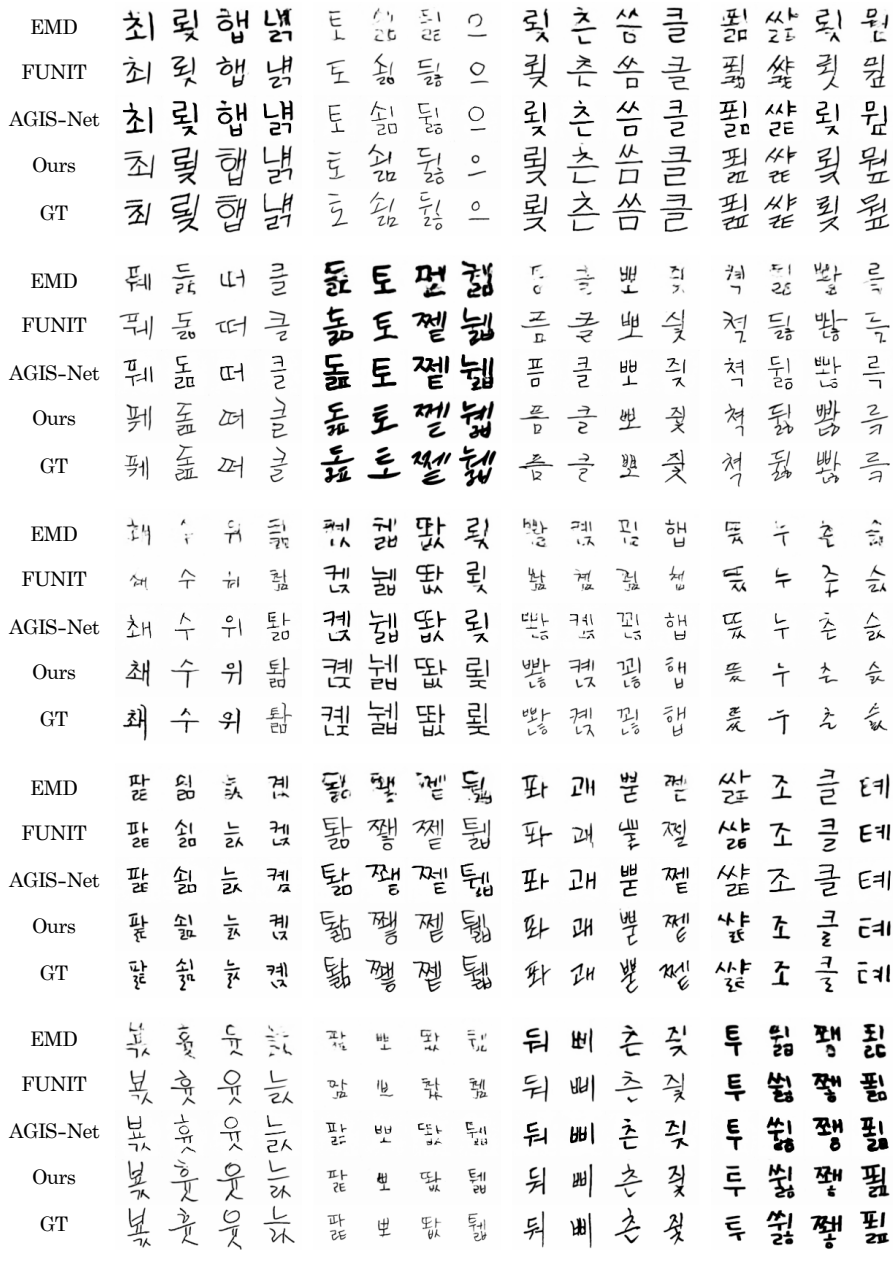


Fig. 10: Additional samples for the user study. The Korean-unrefined dataset is used.

Table 6: **Ablation studies on the Korean-handwriting dataset.** Higher is better, except perceptual distance (PD) and mFID.

(a) Impact of components. DM, PM, and Comp. G denote dynamic memory, persistent memory, and compositional generator, respectively.

	Pixel-level		Content-aware			Style-aware		
	SSIM	MS-SSIM	Acc(%)	PD	mFID	Acc(%)	PD	mFID
Evaluation on the **seen** character set during training								
Baseline	0.689	0.373	96.7	0.026	33.6	6.5	0.084	132.7
+ DM	0.702	0.424	**99.7**	**0.015**	**19.5**	31.8	0.060	77.6
+ PM	**0.704**	0.435	97.7	0.020	26.9	46.6	0.049	57.1
+ Comp. G	**0.704**	**0.457**	98.1	0.018	22.1	**64.1**	**0.038**	**34.6**
Evaluation on the **unseen** character set during training								
Baseline	0.693	0.375	96.6	0.027	34.3	6.5	0.084	134.8
+ DM	0.705	0.423	**99.8**	**0.015**	**19.5**	32.3	0.060	81.0
+ PM	**0.707**	0.432	97.6	0.022	28.9	45.9	0.050	61.4
+ Comp. G	**0.707**	**0.455**	98.5	0.018	20.8	**62.6**	**0.039**	**40.5**

(b) Impact of objective functions.

	Pixel-level		Content-aware			Style-aware		
	SSIM	MS-SSIM	Acc(%)	PD	mFID	Acc(%)	PD	mFID
Evaluation on the **seen** character set during training								
Full	**0.704**	**0.457**	**98.1**	**0.018**	**22.1**	**64.1**	**0.038**	**34.6**
Full $-\mathcal{L}_{l1}$	0.695	0.407	97.0	0.022	27.9	53.4	0.046	48.3
Full $-\mathcal{L}_{feat}$	0.699	0.427	97.8	0.020	23.8	51.4	0.047	51.4
Full $-\mathcal{L}_{cls}$	0.634	0.223	3.0	0.488	965.3	16.2	0.082	118.9
Evaluation on the **unseen** character set during training								
Full	**0.707**	**0.455**	**98.5**	**0.018**	**20.8**	**62.6**	**0.039**	**40.5**
Full $-\mathcal{L}_{l1}$	0.697	0.401	97.5	0.023	26.8	54.3	0.046	52.3
Full $-\mathcal{L}_{feat}$	0.701	0.423	97.8	0.020	24.1	51.2	0.048	56.0
Full $-\mathcal{L}_{cls}$	0.636	0.220	3.2	0.486	960.7	15.9	0.082	123.7