LoGAN: Multilingual Font Localization with Generative Agents

Zhuoning Yuan[1], Ta-Ying Cheng[1] , Benjamin Klein[1]

[1]Netflix, 121 Albright Way, Los Gatos, CA 95032
{zyuan,tayingc,bklein}@netflix.com

Abstract. Localizing a font into new languages is a highly intricate task requiring precise design adaptation of glyphs, color/texture, and spacing/kerning, from source to target languages. Most existing methods focus on single glyph generation with limited capability in handling multilingual font rendering. In this work, we propose LoGAN, a VLM-based agentic framework for few-shot multilingual font localization, which takes in a small number of individual glyphs from a font or letters from a logo and uses them to generate complete character sets in other languages. LoGAN breaks down this task into multiple components: a glyph-level diffusion model, a style finetuning module, a spacing and kerning transfer algorithm, and a texture expansion model, with a VLM agent coordinator. LoGAN achieves broad language coverage for font localization with various styles, including Chinese/Korean/Japanese (CJK). We evaluate our approach on both font and real-world logo datasets spanning more than 27 languages and compare it against both specialized font generation and state-of-the-art image editing models with strong text rendering capabilities (e.g., FLUX, Nano-Banana). Our approach yields higher glyph fidelity while maintaining better style, texture, and kerning consistency according to both quantitative and qualitative evaluations.

Keywords: Font Localization · VLM Agent · Generative Modeling

1 Introduction

Fonts are fundamental visual elements in graphic and logo design. However, most highly stylized fonts support a limited set of languages, predominantly offering only Latin character sets. Localizing these designs to new languages poses a significant challenge due to the immense creative freedom inherent in font design. Because artistic fonts frequently employ idiosyncratic geometric motifs, complex textures, and exaggerated stroke variations, transferring these unconstrained styles to entirely different character sets is highly non-trivial. Designers must navigate severe structural discrepancies—such as mapping simple Latin strokes to dense, complex CJK ideograms without sacrificing legibility—while preserving local stylistic motifs and re-engineering spatial dynamics like kerning. Consequently, in real-world workflows like movie logo localization, finding a matching multilingual font is a recurring bottleneck. When none is available, designers are forced into a difficult trade-off: either substitute a visually similar font and compromise stylistic coherence, or painstakingly synthesize the missing characters by hand. While movie logo creation involves graphical elements beyond basic fonts, our sample study of a 400-logo movie dataset (Section 4.5) shows that 44% of logos are created using fonts. Therefore, we focus specifically on the core challenge of font localization and use the movie logo localization task to evaluate our approach.

Recent advances in generative modeling—such as Nano-Banana, GPT-Image, FLUX, and Qwen-Image [25,34,59]—open up the possibility of applying general image diffusion models to tackle such

Fig. 1: Overview. We present LoGAN, an agentic framework that localizes fonts into various languages. LoGAN is able to generalize to various font styles, spacing/kernings, colors, and languages spanning across Latin, Greek, Cyrillic and CJK.

challenges. A few pioneering works have explored using recent Text-to-Image diffusion models for creative design tasks, ranging from general image editing and style transfer [23, 33, 64], poster-level generation [10, 22, 56], logo-level generation [11, 35, 43, 53], to font-level generation [18, 28, 36, 37]. Although these methods show promising results, they are not readily adaptable to the challenges posed by font localization tasks due to the following: *(1) Existing font generation models primarily focus on glyph generation.* Most prior work (e.g., FontDiffuser, VecGlypher) focuses on generating single glyph images or SVGs and **lacks the ability to render a complete font** (e.g., generating multiple characters with proper spacing/kerning and consistent color/texture, as shown in Figure 2 left column). This limitation makes these methods unsuitable for end-to-end practical use and more appropriate as intermediate steps. *(2) Many end-to-end models can render complete text but lack controllability and stability.* For example, approaches such as FLUX and Nano-Banana often struggle with long-text rendering in font generation settings. When the target layout is complex, they may rescale letters improperly and produce incorrect arrangements (Figure 2 middle column). *(3) Many existing approaches lack multilingual support.* Most recent image generation models, including font generation models, are primarily trained on a single language or only a few languages (e.g., English/Latin) and often fail to produce high-quality, consistent results for other scripts, such as Cyrillic, Greek, and CJK (Figure 2 right column). Additionally, they often struggle with cross-language localization tasks due to limited training data and/or insufficient cross-lingual modeling.

To address these challenges, we introduce a Multilingual Font **Lo**calization with **G**enerative **A**gen**t** (**LoGAN**). The key ideas behind LoGAN are twofold: **First**, while end-to-end models are impressive in one-shot generation, they struggle with output precision and controllability for

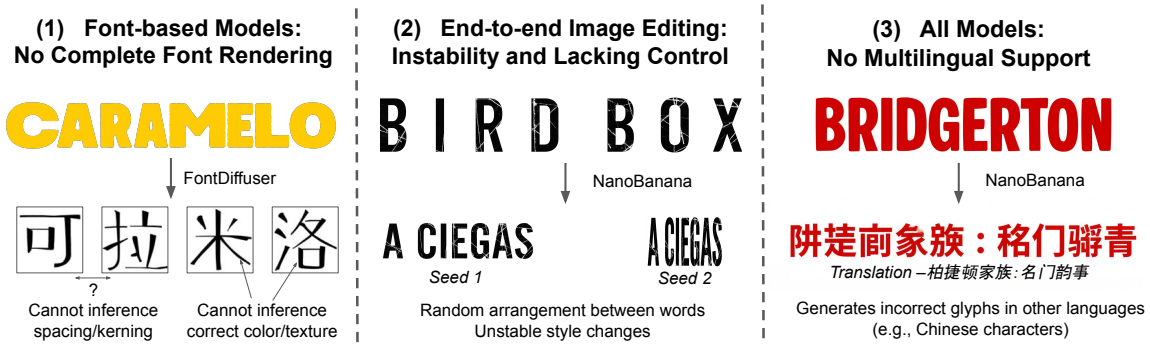

Fig. 2: Challenges of current image generation models. (1) Font-based models primarily focus on glyph generation and fails to render multiple characters with proper spacing/kerning and consistent color/-textures. (2) End-to-end image editing models introduces randomness towards style changes and arrangements. (3) All current models lack a wide range of multilingual support.

font rendering tasks. Multiple design elements, such as glyphs, spacing/kerning, color/texture, and layout, are coupled in a one-pass generation, which often leads to hallucinations of certain design attributes and requires re-prompting and regenerating until the results are correct. Inspired by the strong visual understanding capabilities of recent SOTA VLMs (e.g., `GPT-5`, `Gemini-3-Pro` [24,45]), we propose a divide-and-conquer approach by decomposing the complex font localization task into basic design elements. We further develop *expert models* to fulfill each of these steps—such as using a VLM to detect color, spacing, and layout, using an image generation model to generate glyphs, and using a VLM agent to coordinate the complete font rendering process. **Second**, we focus explicitly on glyph-level generation followed by a spacing/kerning transfer module instead of word- or sentence-level text generation, which usually carries the risk of hallucinating certain prompts in underrepresented languages or new words created by the internet that do not frequently occur in the training data, as well as the risk of characters being rescaled during long-text rendering. We develop a strong multilingual glyph-level generation model that supports 27 languages for both same-language and cross-language generation, especially for CJK (see examples in Figure 1), as part of the proposed font localization pipeline.

To the best of our knowledge, this is the first work to propose an agent-based end-to-end pipeline for multilingual font localization and rendering. The design overview of LoGAN is shown in Figure 3. Our main contributions are summarized as follows: We develop a novel framework, LoGAN, that utilizes a VLM as a design agent to coordinate complex font localization and rendering tasks through three stages: design decomposition, design transfer, and design generation. We further develop a multilingual glyph generation module—including glyph generation, arrangement, and texturing models—for high-quality vector font rendering. We conduct a comprehensive evaluation to validate LoGAN, spanning Latin, Cyrillic, Greek, and CJK languages on font and real movie logo datasets. We present both qualitative and quantitative results and compare LoGAN against specialized font-generation methods and state-of-the-art image editing/generation models, achieving the best performance across all metrics.

2 Related Work

Generative models have advanced substantially in the quality of image generation [17, 41, 46, 48, 50] and editing [8, 23, 33, 59].

Image Diffusion for Font Generation/Editing. With these improvements, a plethora of works has started to focus on the controllability of diffusion models. These controls span structural guidance [6, 55, 66], style transfer [15, 54, 64], and visual characteristics/textural transfer [13, 14, 68]. However, these models fall short in the task of logo localization, where the target style must be precisely transferred while being disentangled from the original content. For example, direct style transfer or prompt-based image editing often entangles glyph content with stylistic attributes, bringing unwanted stroke structures and character-specific features into the newly localized logo. Several works aimed specifically at poster design [10, 22, 56], text rendering [35, 43, 53] or font generation [19, 42, 63]. Nevertheless, their style transfers tend to be comparatively coarse and are less capable in generalizing across scripts with drastically different typographic and morphological properties, e.g., Latin-based scripts to Korean.

VLM for Image Editing. Guiding diffusion models for optimal image generation remains an ongoing challenge. With the rise of LLMs and VLMs, many works leverage language model feedback to enhance prompts for generation [21, 29, 40]. These are all different approaches aiming to improve the prompt for better image generation. [65] This method decomposes an abstract prompt into sequences of simpler editing instructions for diffusion models. [61] Another method improves the generated output based on the VLM feedback loop. [59] This approach feeds the prompt into a VLM and uses the hidden features as inputs into the diffusion model for greater expressibility. There are also approaches that treat VLM as an agent to call a library of model tools to iteratively refine a generated image [57].

In this work, we leverage VLM as a design agent that decomposes complex font localization into simpler tasks and coordinates specialized models to produce detailed, high-quality logos.

3 LoGAN

We present the overall design and workflow of LoGAN, as shown in Figure 3, and introduce its key components in the subsequent sections. LoGAN takes a base prompt of `Localize this image to [Translation]` and a reference font image, then executes the following actions: *design extraction*, *design transfer*, and *design generation*, and finally returns a localized rendered font image that matches the style of the reference and target translation. For each stage, the details are as follows:

- **Stage 1: Design Extraction**. LoGAN first decomposes users' input into: reference image, translation, and style finetuning trigger. Then, it calls VLM-Agent to analyze and extract design attributes from the reference image, including OCR text, font style, color/texture, line breaks, letter case, and text alignment, where texture and font style are in abstract form, e.g., *style1*, *texture1*. Next, it calls the AutoCrop module to crop the reference image into glyphs and measures the spacing between adjacent cropped glyph pairs. Finally, LoGAN summarizes the extracted data as a *character-attribute* mapping, key attributes, a list of cropped glyphs, and the trigger for style finetuning.
- **Stage 2: Design Transfer**. LoGAN processes the reference attributes and triggers extracted in Stage 1. It then calls the VLM-Agent to infer a new *character–attribute* mapping for the target translation and, optionally, triggers the Style-Finetuning module (based on the user's

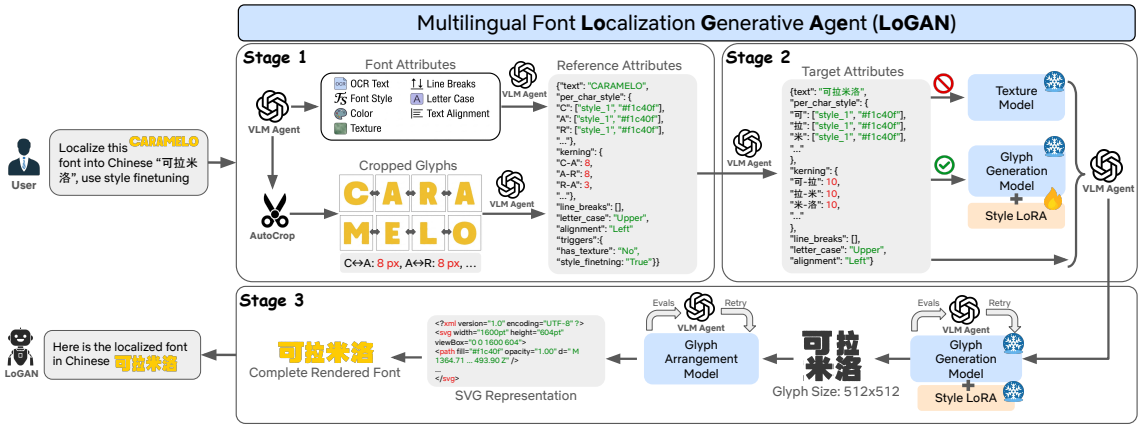

Fig. 3: Method Overview. LoGAN takes a reference image and a target translation generates translated font with three stages: (1) Design Extraction, (2) Design Transfer, and (3) Design Generation. Style LoRA is trained during (2) and then frozen for inference during (3).

request) to fine-tune the pre-trained glyph generation model with LoRA [27], improving style transfer quality. Next, LoGAN determines whether to call the Texture-Expansion module based on the texture detected in Stage 1.

– **Stage 3: Design Generation**. With all required inputs, LoGAN triggers the Glyph-Generation model, optionally with LoRA, using the translation to generate the target glyphs at a 512x512 resolution. It then calls the Glyph-Arrangement model to arrange the generated glyphs into a fully rendered font. Following this, LoGAN calls VLM-Agent to inspect the outputs (e.g., glyphs, line breaks, colors), and triggers a retry if errors are detected. Finally, Glyph-Arrangement model vectorizes the generated font. If generated texture was detected, LoGAN will overlay this texture map onto the final vectorized font.

3.1 Key Components

VLM Agent. This is the coordinator of LoGAN with three different roles: (1) *Interpreter*: decompose the user request, extract design attributes from the reference image (e.g., OCR text, layout, style, color/texture, spacing/kerning, etc.), and assist with glyph cropping. (2) *Planner*: localize the reference attributes to target languages, decide whether to trigger optional modules (e.g., style LoRA, Texture-Expansion) and trigger retries if evaluation fails. (3) *Judge*: verify the correctness of the generated glyphs and the overall quality of font rendering (e.g., kerning/layout, color/texture consistency). In our implementation, we use `GPT-5.2` [45] as the default VLM-Agent and validate that open-source Qwen3-VL-32B-Instruct [5] achieves comparable performance.

Glyph Generation. Most existing glyph/font generation approaches focus on only a few languages [28, 63]. To address this, we develop a glyph generation model spanning 27 languages based on MMDiT (SD3.5-medium) [1]. Specifically, we modify the joint attention in each block to add a third modality that encodes reference glyphs: we apply modality-specific projections (and RMSNorm), concatenate all modalities' Q/K/V, followed by a single unified scaled dot-product atten-

Table 1: Abstract Shapes for Kerning Heuristics.

Shape (Symbol)	**Description**
Straight (|)	Vertical or implied vertical boundaries that restrict close kerning (e.g., both sides of "H", right side of "E", left side of "B", etc. All CJK letters are also assumed to fall within this shape).
Circular (∘)	Curved boundaries that naturally accommodate tighter kerning (e.g., both sides of "O" or the left side of "C").
↗ Diagonal (/)	Ascending diagonal strokes from bottom-left to top-right (e.g., the left side of "A" or the right side of "V". Right side of "T" also belongs to this shape as you can draw an implicit ↗ for tighter kerning).
↘ Diagonal (\)	Descending diagonal strokes from top-left to bottom-right (e.g., the right side of "A" or the left side of "V"). Certain lowercase letters, such as "e" or "w", are also treated as diagonal because their open contours provide spatial gaps for tighter kerning.

tion [38]. For the other dependent components, we adopt siglip2-so400m-patch16-256 [52] to encode reference glyphs, concatenate k glyph embeddings into $(N, k * 256, 1152)$, and pass them through Perceiver Sampler [2] to obtain a fixed-length embedding $(N, 256, 1152)$. We utilize ByT5-Large [62] for glyph-level prompt encoding and a 16-channel VAE from AuraDiffusion [3]. We finally train a 2B-parameter glyph generation model from scratch. Training details are included in Section 3.2. We also develop a Style-Finetuning module with LoRA finetuning on reference glyphs to ensure the generated glyphs preserve the exact character identity when seen in the reference (e.g., **Zootopi**a[EN] → **Zootropoli**s[TR]), as most zero-shot approaches often introduce subtle pixel shifts that can limit practical usability. Additionally, we find this fine-tuning can improve overall style transfer quality (Table 2).

Glyph Arrangement. Given the generated glyphs, the reference/target design attributes (e.g., line breaks, spacing pairs), LoGAN arranges the generated glyphs into a fully rendered font that matches the reference through four sub-tasks: (1) spacing/kerning transfer, (2) line break and alignment transfer, (3) glyph vectorization, and (4) color/texture transfer.

– **Spacing and Kerning Transfer.** Kerning and spacing refer to the gap between two adjacent letter pairs. Different character pairs require different spacing to achieve a similar perceived visual gap (e.g., "WA" is typically much tighter than "HO"). LoGAN predicts spacing for target character pairs based on the spacing of reference character pairs extracted by AutoCrop. The key challenge is to reuse the same character-pair spacings when they are present in the reference and to infer unseen pairs using typographic rules. Detailed steps are as follows:

  • We first categorize the left and right contour of each character (across languages) into four abstract shapes (Table 1): Straight (|), Circular (∘), ↗ Diagonal (/), and ↘ Diagonal (\). This abstraction maps character pairs to high-level shape combinations. For example, both "HO" and "EC" correspond to |∘. Based on a small study with typographic experts, we adopt the following heuristic ordering:

$$S(||) > S(|/) = S(|\backslash) > S(|\circ) = S(\circ\circ) > S(\backslash/) > S(\circ\backslash) = S(\circ/) > S(\backslash\backslash) = S(//), \quad (1)$$

where $S(\cdot)$ denotes the spacing/kerning for a boundary-shape pair (right side of the preceding character + left side of the succeeding character).

- For each character pair in the reference, we extract watertight, character-level bounding boxes using AutoCrop (Section 3.1). We then measure the pixel gap between bounding boxes for each character pair and store these values in a dictionary. If multiple character pairs in the reference share the same abstract shape pair, we randomly select one measured value and apply it to the corresponding target shape pair. During inference, we reuse the cached value when the same character pair appears in the reference; otherwise, we infer the spacing using Eq. 1. Finally, we refine the result by enforcing a minimum pixel gap between adjacent glyphs, bounded by the average spacing across the localized font.

– **Line Break and Alignment Transfer.** We apply line breaks based on positions – the character indices at which to insert line breaks, e.g., "Hello
 World", where the break index is 5 – and text alignment (left|center|right) from target design attributes inferred by VLM-Agent.
– **Glyph Vectorization.** We convert generated glyphs to SVG to make them editable for future use. Following [37], we use VectorMagic [9] and also explore an open-source alternative, Potrace [51], which has comparable performance. We compare different PNG-to-SVG approaches (see more examples in the Appendix) and find that directly generating accurate SVGs remains challenging [49].
– **Color and Texture Transfer.** To transfer monochrome colors, we directly apply the color code from the reference to the final vectorized font. For complex textures, we feed the image containing the textured text into the Texture-Expansion model to recover the texture map from the given reference font (Figure 8) and overlay it onto the final vectorized font.

Glyph Cropping. One key challenge in both glyph generation (constructing model inputs) and glyph arrangement (measuring inter-character spacing) is to reliably crop a complete rendered font into individual glyphs. Most prior text-detection methods operate at the word or line level [16, 39, 58]. Although CRAFT [4] performs character-level detection, it is trained on SynthText and ICDAR [26] which can lead to the domain shift for our setting. We therefore propose a training-free method for multilingual character-level detection and cropping named AutoCrop. Our method is robust to characters composed of multiple disconnected components (e.g., CJK characters or Latin characters with diacritics) [16]. AutoCrop consists of four steps: (1) we apply connectedComponentsWithStats [60] to extract connected components and their masks; (2) we render an index label on each component; (3) we use VLM (OCR) to group component indices by character identity

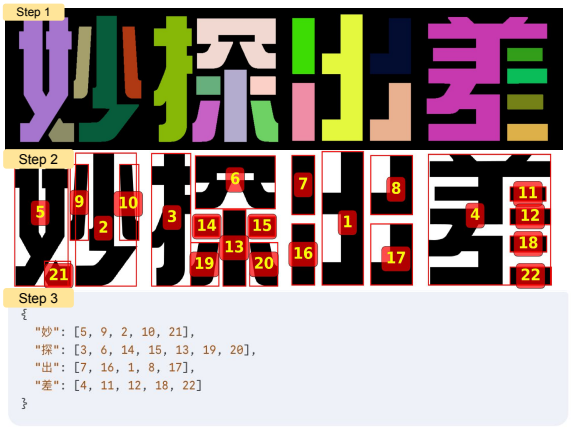

Fig. 4: Illustration of character-level cropping for Chinese characters "Beverly Hills Cop".

and then merge grouped components to produce a refined bounding box per character; and (4) we sort refined bounding boxes in reading order (top-to-bottom, left-to-right) to align with the OCR transcript. Figure 4 illustrates the overall pipeline.

3.2 Training Strategy

Cross-Lingual Sampling. Many existing generation models [23, 28, 33, 63] are specialized in a limited number of languages due to the scarcity of multilingual training data and limited cross-lingual modeling. To address these challenges, we first analyzed an internal font dataset of 15,000 popular fonts and found a drastically long-tailed distribution that favors Latin languages over others (Figure 5). We then propose a cross-lingual sampling algorithm, presented in Algorithm 1 and 2. The core idea is to mitigate the long-tail bias by explicitly balancing same-language and cross-language sampling pairs during training. For Latin scripts, we further introduce case- and diacritic-aware sampling to improve training diversity, e.g., mixing letter cases (hELlO $\rightarrow$ A) and mapping diacritic variants to their base forms (AÀÁÂÄÅ $\rightarrow$ A, ùúûüūū $\rightarrow$ u). Finally, we include some duplicated pairs (AAAAAA $\rightarrow$ A) to further increase training diversity.

Glyph Generation Training. We started with pre-training on 256×256 image resolution using Fully Sharded Data Parallel (FSDP) distributed across 8 A100 (80GB) GPUs with bfloat16 mixed precision and a global batch size of 1024. We freeze all encoders (image encoder, VAE) and train only the main transformer backbone. We follow the training settings from Stable Diffusion 3 [17]. We utilized AdamW optimizer with $\epsilon = 10^{-15}$. The learning rate was fixed at 1.0×10^{-4} using a constant scheduler, following a linear warmup of 1,000 steps. We applied gradient clipping with a maximum norm of 1.0. After training for 450k iterations, we switched to 512×512 image resolution for training another 450k iterations using the same learning parameters, with the batch size reduced to 640. Additionally, we used three types of glyph augmentation: zooming in or out, widening or narrowing either horizontally or vertically, and applying italic styles following [37]. The total training takes about 15 days to complete.

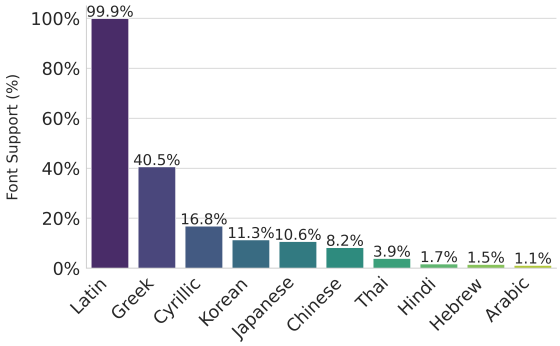

Fig. 5: Statistics of language distribution on our internal font dataset.

Style Finetuning. Starting from the pretrained 2B glyph generation model, we train a LoRA for Style-Finetuning. We first use AutoCrop to crop the reference font into square images for each glyph. Since the reference font can be of arbitrary sizes, we resize all cropped glyphs to 512×512. We run the LoRA fine-tuning on a single A100 (40GB). We use PEFT [44] to implement LoRA with rank 64 and alpha 64 with about 159.5M trainable parameters. The injected LoRA layers are included in the Appendix. We use the same training configurations as in pretraining, except with a smaller batch size (up to 8), depending on the number of available cropped glyphs in the reference. We train the LoRA for 800 steps, and the average training time is about 30–40 minutes.

Algorithm 1 Language-Aware Char Sampling

Input: Character c, Font f, Sample size k
Output: Set of k sampled characters
Notation for Algorithms 1 and 2: $L(c, f)$ returns the language group of character c in font f; $F(l, f)$ returns the set of all characters in language group l for font f; $\mathcal{L}$ = {Latin, Greek, Cyrillic, Chinese, Japanese, Korean}. IsUpper(c) and IsLower(c) return `true` if character c is uppercase or lowercase, respectively. Normalize(c) removes diacritics from character c. Denormalize(c) returns all diacritic variants of character c.
function SampleCharacters(c, f, k)
    $p_1, p_2 \leftarrow$ Random(), Random()
    $l_1 \leftarrow L(c, f)$
    if $p_1 < 0.05$ **then**
        return $[c] \times k$
    end if
    if $p_1 < 0.5$ **then**
        $\mathcal{C} \leftarrow F(l_1, f)$ ▷ Same-language sampling
    else
        $l_2 \leftarrow$ RandomSample($\mathcal{L} \setminus \{l_1\}, 1$)
        $\mathcal{C} \leftarrow F(l_2, f)$ ▷ Cross-language sampling
    end if
    return CaseSensitiveSample($c, k, \mathcal{C}, p_2$)
end function

Algorithm 2 Case-Sensitive Char Sampling

Input: Character c, Sample size k, Candidate set $\mathcal{C}$, Probability p_2
Output: Set of k sampled characters
function CaseSensitiveSample($c, k, \mathcal{C}, p_2$)
    if $p_2 < 0.25$ **then**
        if IsUpper(c) **then**
            $\mathcal{C} \leftarrow \{x \in \mathcal{C} \mid \text{IsLower}(x)\}$
        else if IsLower(c) **then**
            $\mathcal{C} \leftarrow \{x \in \mathcal{C} \mid \text{IsUpper}(x)\}$
        end if
    else if $0.25 \leq p_2 < 0.5$ **then**
        $\mathcal{C} \leftarrow \{c + x \mid x \in \mathcal{C}\}$ ▷ Must include c
    else if $0.5 \leq p_2 < 0.75$ **then**
        if HasDiacritic(c) **then**
            $c_{\text{base}} \leftarrow$ Normalize(c)
            $\mathcal{C} \leftarrow \{x + c_{\text{base}} \mid x \in \mathcal{C}\}$
        else
            $\mathcal{C} \leftarrow \{x + c \mid x \in \mathcal{C} \wedge \text{Denormalize}(x)\}$
        end if
    end if
    if $|\mathcal{C}| < k$ **then**
        return $[c] \times k$
    end if
    return RandomSample(C, k)
end function

Texture Expansion Model. We train a LoRA [27] on the FLUX.1-Fill-dev model [32] to recover the full texture given a reference font with an overlaid texture. We use PEFT to implement the LoRA fine-tuning. The trainable parameters are about 44.8M. Given the relatively small dataset size of PBR materials [7], we apply heavy augmentations to the materials, including zooming, changing brightness, color inversion, etc., to expand texture diversity. We also randomize font sizes and dynamically sample characters on-the-fly to further vary stroke widths and spatial layouts. We used a global batch size of 2 and a learning rate of 1e-4 for training 20,000 steps distributed across 2 A100 (40GB) GPUs using bfloat16 mixed precision. We use an image resolution of 1536x1536 for training. The total training takes about 12 hrs to complete.

4 Experiments

4.1 Dataset Creation

To train the glyph generation model, we use an internal dataset of 15,000 fonts, including 40 million glyphs. We construct two versions of the dataset at 256×256 and 512×512 resolutions for two-stage training. For the texture expansion model, we curate a dataset of 542 PBR materials [7], with 433 used for training and 109 for testing. We use Matplotlib [30] to render each glyph centered in a square image with a white background. For evaluating LoGAN, we focus on **word-level** assessment to assess complete font rendering rather than individual glyphs.

Our evaluation dataset has two parts: (1) we collect 50 unseen fonts to create 300 synthetic movie-title pairs in the format (*Movie1 in LanguageA*, *Movie1 in LanguageB*), and we use 50 of the 300 pairs to add random textures sampled from the PBR materials test set; and (2) we collect 700 real font-based movie logos with corresponding language pairs in various monochrome colors. To

Table 2: Qualitative Comparison. We measure PSNR, SSIM, LPIPS [67], DreamSim (DSim) [20], and CLIP similarity score [47]. Results are shown for Non-CJK (254 pairs) and CJK (42 pairs). Editing type indicates whether the model is trained for text rerendering. Red/yellow denote 1st/2nd.

Lang. (#)	Model	Editing Type	PSNR↑	SSIM↑	LPIPS↓	DSim↓	CLIP↑
CJK (42)	AnyText2 [53]	Text	6.894	0.329	0.825	0.510	0.811
	TextDiffuser-2 [12]	Text	6.460	0.449	0.882	0.604	0.666
	Flux-Text [35]	Text	6.900	0.580	0.706	0.374	0.858
	Flux-Kontext Pro [33]	Image	7.050	0.635	0.667	0.162	0.801
	Nano-Banana [23]	Image	7.658	0.622	0.588	0.101	0.920
	LoGAN w/o LoRA (GPT-5.2)	Text	9.052	0.657	0.449	0.084	0.932
	LoGAN (Qwen)	Text	8.186	0.671	0.536	0.081	0.929
	LoGAN (GPT-5.2)	Text	9.494	0.692	0.446	0.076	0.948
Non-CJK (254)	AnyText2 [53]	Text	7.123	0.458	0.806	0.524	0.812
	TextDiffuser-2 [12]	Text	6.875	0.541	0.862	0.631	0.658
	Flux-Text [35]	Text	6.973	0.560	0.706	0.444	0.838
	Flux-Kontext Pro [33]	Image	7.784	0.596	0.607	0.158	0.876
	Nano-Banana [23]	Image	8.988	0.638	0.481	0.084	0.922
	LoGAN w/o LoRA (GPT-5.2)	Text	9.094	0.654	0.444	0.085	0.932
	LoGAN (Qwen)	Text	9.162	0.664	0.451	0.109	0.927
	LoGAN (GPT-5.2)	Text	10.077	0.684	0.373	0.072	0.944

construct the synthetic movie-title pairs, we collect movie titles and their translations—spanning Latin, Cyrillic, Greek, and CJK scripts—from the IMDb Top Movies list [31]. We use this list to render the test fonts with random colors and textures (from the PBR test set) to increase the evaluation diversity. For the unseen test set, the selection criteria are as follows: (1) we choose fonts that were released substantially later than those in the training set (a gap of at least 1 year); (2) we compute similarity scores using SigLIP2 embeddings on the same set of 10 randomly sampled glyph images (A–Z), and keep only fonts whose average cosine similarity score is < 0.5.

4.2 Quantitative Evaluation

We compare two versions of LoGAN: one using `GPT-5.2` as the central VLM agent, and the other using the open-source Qwen3-VL-32B-Instruct model and compare against baselines specifically trained for texts (Flux-Text [35], AnyText2 [53]), TextDiffuser-2 [12], as well as general generative models (Nano-Banana [23], FLUX-Kontext Pro [33]). We compute pixel-wise metrics PSNR and SSIM, alongside perceptual metrics LPIPS [67], DreamSim [20], and CLIP Score [47].

Table 2 presents our results. We observe substantial improvements across all pixel-wise and perceptual metrics. Interestingly, models specifically trained on text rendering perform worse than recent general-purpose image editing models. All baselines perform significantly worse when the output language is CJK, likely due to the scarcity of multilingual training data. In contrast, LoGAN (with both open-source and proprietary VLMs) benefits from the strong Glyph-Generation model, leading to comparable performance across metrics for both CJK and non-CJK outputs. We also report results for LoGAN without a fine-tuned LoRA using `GPT-5.2`, highlighting the benefit of style fine-tuning in reducing the domain gap and improving overall style transfer quality while **requiring only the reference image itself**. Notably, LoGAN still outperforms the baselines even without LoRA.

4.3 Qualitative Evaluation

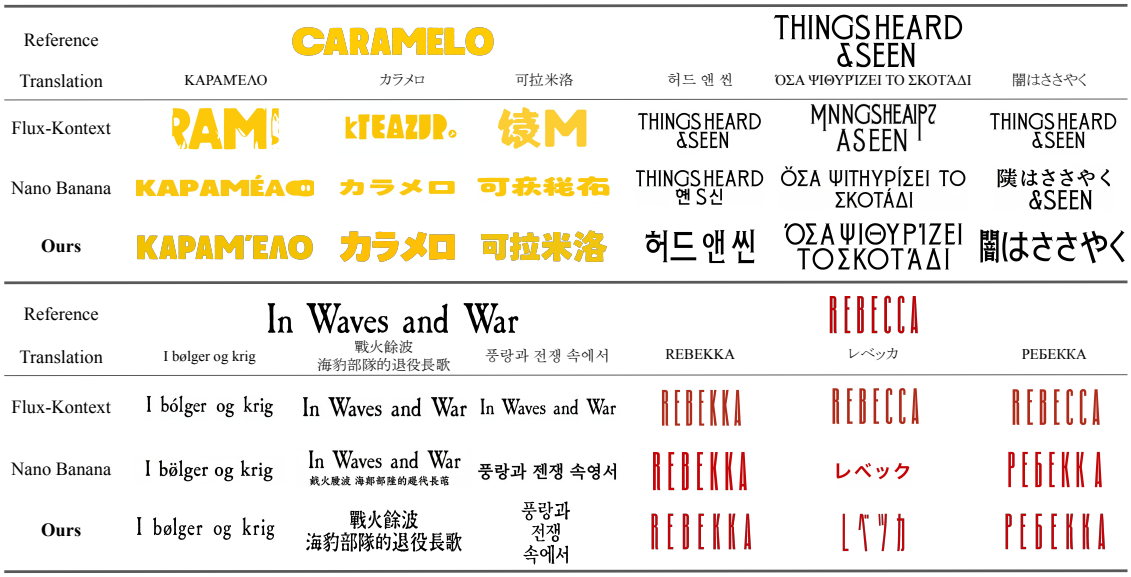

Fig. 6: Qualitative Comparison. We present the results of 12 localized logos spanning across 4 movie titles. Flux-Kontext often ignores the text guidance and does not perform any edits, especially for CJK languages. Nano-Banana often misses the translations and slightly alters the style. Our results performs accurate style and layout transfer while simultaneously translating the titles correctly.

Font Quality Comparison. Figure 6 presents 12 examples of 12 movies logos from four titles. We compare LoGAN (`GPT-5.2`) against two strong baselines: Nano-Banana [23] and FLUX Kontext Pro [33]. FLUX Kontext Pro often produces completely illegible results (e.g., all instances of "Caramelo") or fails to perform any edits (columns 1 and 3 for "Things Heard & Seen" and columns 2 and 3 for “In Waves and War”). Nano-Banana, while performing better in English (column 1 for "Rebecca"), frequently misses diacritics in other Latin-based languages (column 1 for "In Waves and War"), slightly changes the style (all instances of "Caramelo"), or produces incorrect translations for non-Latin scripts (column 2 for “Things Heard & Seen” and "In Waves and War"). In contrast, LoGAN accurately preserves the layout, spacing, and style of the original title and applies them consistently to both CJK and non-CJK languages. Column 2 of "Rebecca" is a clear example, where the tall, slim style is captured and correctly transferred to Japanese.

Glyph Quality Comparison. Existing font-based methods, such as FontDiffuser [63], cannot predict spacing/kerning and align multiple characters. We therefore provide an additional qualitative comparison to highlight LoGAN’s glyph generation quality. Since FontDiffuser only accepts and generates a single character, we use randomly cropped glyphs from a logo as references when generating characters. We show comparisons in Chinese (their strongest language) in Figure 7. LoGAN produces more accurate styles and colors.

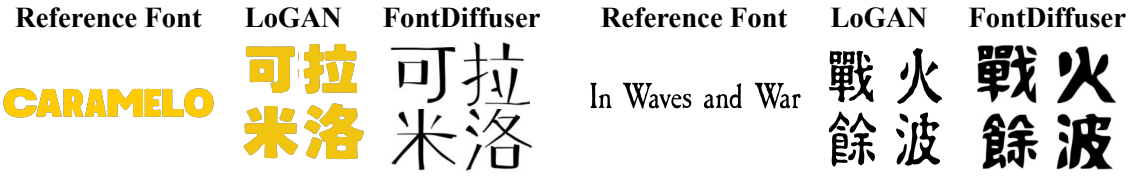

Fig. 7: Glyph Quality Comparison Against FontDiffuser. FontDiffuser fails to match the original style/color of the reference font.

Texture Expansion Quality Comparison. We also perform a qualitative comparison of our Texture-Expansion model. To the best of our knowledge, we are the first to train a model that recovers texture from a given text image with a burnt-in texture. Thus, our closest comparison is a general image editing model prompted with `Recover the texture from the given text image`.

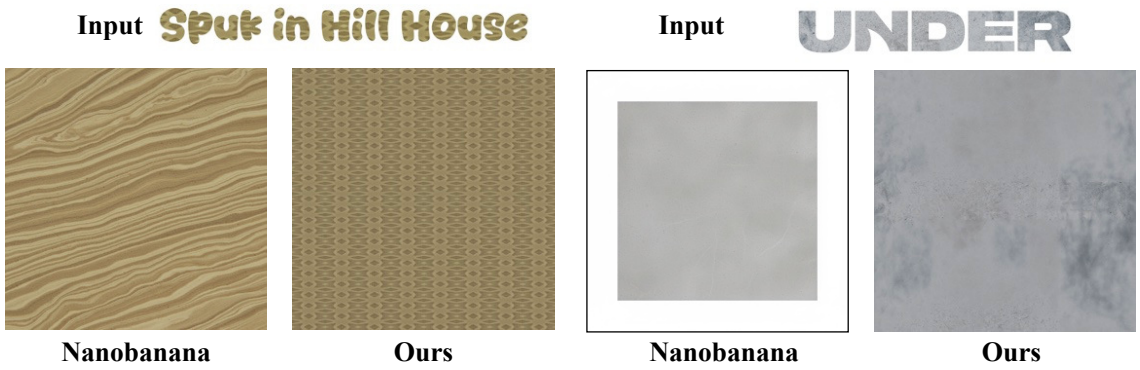

Fig. 8: Texture Expansion Qualitative Comparison. We present the qualitative results of our texture expansion approach against Nano-Banana [23].

Figure 8 shows qualitative comparisons between our model and Nano-Banana [23]. Since Nano-Banana is not specifically trained for texture expansion, it often generates incorrect textures (left) or loses fine-grained details (right). In contrast, our model better preserves fine details and more closely matches the original texture in the reference inputs.

4.4 Expert Evaluation

To better assess fine-grained details in generated fonts of LoGAN, we worked with a font designer rather than conducting a user study, because non-expert users often **overlook the intricate details of font design**. Specifically, the designer was asked to evaluate the generated fonts in terms of: (1) style transfer (whether the generated letters match the reference's visual styling, e.g., stroke shape/weight, curvature, color, and texture), (2) kerning/spacing (between adjacent characters), and (3) translation accuracy (whether the generated glyphs match the prompt). The evaluation had two parts: (1) evaluating 80 generated fonts (from five randomly selected reference

movie titles), comparing LoGAN against the strongest baseline, Nano-Banana, using a 3-point scale (1, 2, 3), where 1 = completely inaccurate, 2 = decent but with minor issues, and 3 = completely accurate and ready to use; and (2) asking the designer to classify all 700 generated font-based logos as strictly **"Production-Ready"** or not.

In part 1, LoGAN significantly outperformed the baseline, achieving average scores of 2.8 for style transfer and 2.2 for layout/spacing. Nano-Banana performed poorly (average 1.0) in both categories, underscoring the difficulty of high-fidelity layout preservation. In addition, LoGAN was consistently accurate in translation (average 3.0), whereas Nano-Banana translated the logos correctly only 19 out of 80 times. For part 2, 52.9% of LoGAN's outputs were classified as Production-Ready, demonstrating the strong generalization of our method. Notably, none of Nano-Banana's logos passed this test.

4.5 More Ablation Studies

Single-Glyph Analysis. To mitigate the potential effects of positional discrepancies (e.g., kerning), we compare LoGAN against the strongest baseline, Nano-Banana, by cropping all generated logos into individual glyphs using tight crops to avoid spacing bias and evaluating them with the same five metrics as in the main paper. The results are presented in Table 3, showing that LoGAN outperforms Nano-Banana at the glyph level.

Table 3: Glyph-level comparison for LoGAN and Nano-Banana.

Method	PSNR↑	SSIM↑	LPIPS↓	DreamSim↓	CLIP↑
NB	15.75	0.726	0.179	0.122	0.958
LoGAN	**18.67**	**0.802**	**0.121**	**0.094**	**0.968**

Texture Expansion Analysis. To further validate the impact of the developed Texture-Expansion model, we conducted additional evaluations. We split the 300 test pairs into two groups (see more details in Section 4.1): *with texture* (50) and *without texture* (250). The results in Table 4 show that LoGAN with the Texture-Expansion model achieves better performance than state-of-the-art methods, such as Nano-Banana.

Table 4: Quantitative comparisons for LoGAN and Nano-Banana in texture/no texture split.

Group	Method	PSNR↑	SSIM↑	LPIPS↓	DSim↓	CLIP↑
With Texture	Nano-Banana	10.65	0.606	0.483	**0.100**	0.916
	LoGAN	**11.88**	**0.659**	**0.378**	0.118	**0.934**
Withou texture	Nano-Banana	7.85	0.647	0.504	0.081	0.924
	LoGAN	**9.00**	**0.699**	**0.391**	**0.049**	**0.951**

Efficiency of LoGAN. We benchmark LoGAN's runtime (3 runs, `GPT-5.2`) in two settings: (i) Stage 1: design extraction on a fixed-size synthetic reference image while varying the rendered text length from 1 to 15 complex CJK characters (the average length of a movie title); and (ii) Stages 2–3: generation with 30 denoising steps while varying the output length from 1 to 15 characters. In our experiments, Stage 1 runtimes range from 7.6 s (1 char) to 38.5 s (15 chars). Stages 2–3 take 25.5 s (1 char) to 43.2 s (15 chars) on an A100 (40GB). LoRA finetuning incurs a one-time cost of about 15 minutes. We emphasize that our goal is not to optimize baseline speed but to address their quality limitations.

VLM for Detecting Font Styles. In Stage 1 of our pipeline (Figure 3), we use `GPT-5.2` (VLM) to identify font styles and represent them in an abstract form. To evaluate the robustness of this approach, we collected a new dataset of diverse movie logos from 400 titles released in 2025 across multiple languages. We asked a font designer to group them into three categories (1, 2, 3 font styles or more complex designs beyond font) and asked `GPT-5.2` to classify the number of unique font styles in each sample. Our results show that `GPT-5.2` achieves 93.1% accuracy, validating that our VLM-based approach can robustly interpret diverse font styles and guide the subsequent tasks of LoGAN.

Pipeline Robustness Test. We also evaluated LoGAN's end-to-end pipeline robustness, which is crucial for production use. Here, we do not focus on assessing model quality. Instead, we evaluate whether the pipeline can consistently generate meaningful outputs (e.g., complete rendered texts). To this end, we conduct a large-scale pressure test on a different dataset of 1,800 movie (font-based) logo pairs (e.g., English vs other languages). We run LoGAN (with `GPT-5.2`) end-to-end on this dataset. The results show that LoGAN is highly robust, with only a 3% pipeline failure rate across 1,800 runs. Failures are primarily due to (i) API responses not conforming to the required JSON schema, or (ii) VLM errors (e.g., incorrect character detection or design transfer on minority languages), which lead to downstream failures.

5 Conclusion

We present LoGAN, a VLM-based agentic framework for end-to-end font localization. Prior work often lacks multilingual support and complete font rendering. LoGAN addresses these limitations by proposing several novel components, including a cross-language sampling algorithm, a multilingual glyph generation model, character-level text detection, a spacing and kerning transfer algorithm, and a texture expansion model. Our empirical results show that LoGAN outperforms font generation and state-of-the-art image editing baselines, and our pressure test on the end-to-end pipeline further indicates that LoGAN is a robust solution for real-world practical use.

In future work, we plan to improve LoGAN along three directions: (1) extending beyond single-font localization to support multi-font rendering and more flexible layout and arrangement; (2) improving fine-tuning efficiency—while fine-tuning is optional in our current design, making it faster and more lightweight would improve practicality; and (3) supporting more complex font/logo rendering. Currently, LoGAN relies on glyph-based generation and may struggle with connected characters, e.g., calligraphy fonts.

References

1. AI, S.: Stable diffusion 3.5 medium — model card. https://huggingface.co/stabilityai/stable-diffusion-3.5-medium (2024), accessed: 2025-11-12
2. Alayrac, J.B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al.: Flamingo: a visual language model for few-shot learning. Advances in neural information processing systems **35**, 23716–23736 (2022)
3. AuraDiffusion: 16ch-vae. https://huggingface.co/AuraDiffusion/16ch-vae (2025), model card on Hugging Face; accessed 2025-11-13
4. Baek, Y., Lee, B., Han, D., Yun, S., Lee, H.: Character region awareness for text detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9365–9374 (2019)
5. Bai, S., Cai, Y., Chen, R., Chen, K., Chen, X., Cheng, Z., Deng, L., Ding, W., Gao, C., Ge, C., Ge, W., Guo, Z., Huang, Q., Huang, J., Huang, F., Hui, B., Jiang, S., Li, Z., Li, M., Li, M., Li, K., Lin, Z., Lin, J., Liu, X., Liu, J., Liu, C., Liu, Y., Liu, D., Liu, S., Lu, D., Luo, R., Lv, C., Men, R., Meng, L., Ren, X., Ren, X., Song, S., Sun, Y., Tang, J., Tu, J., Wan, J., Wang, P., Wang, P., Wang, Q., Wang, Y., Xie, T., Xu, Y., Xu, H., Xu, J., Yang, Z., Yang, M., Yang, J., Yang, A., Yu, B., Zhang, F., Zhang, H., Zhang, X., Zheng, B., Zhong, H., Zhou, J., Zhou, F., Zhou, J., Zhu, Y., Zhu, K.: Qwen3-vl technical report. arXiv preprint arXiv:2511.21631 (2025)
6. Bhat, S.F., Mitra, N.J., Wonka, P.: LooseControl: Lifting ControlNet for generalized depth conditioning (2023)
7. Brian, t.c.o.F.: Free pbr materials: Hundreds of 3d texture downloads. https://freepbr.com/ (2025)
8. Brooks, T., Holynski, A., Efros, A.A.: Instructpix2pix: Learning to follow image editing instructions. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 18392–18402 (2023)
9. Cedar Lake Ventures: Vector magic (2025), https://vectormagic.com, version 1.20 (Desktop Edition)
10. Chen, H., Xu, X., Li, W., Ren, J., Ye, T., Liu, S., Chen, Y.C., Zhu, L., Wang, X.: Posta: A go-to framework for customized artistic poster generation. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 28694–28704 (2025)
11. Chen, J., Huang, Y., Lv, T., Cui, L., Chen, Q., Wei, F.: Textdiffuser: Diffusion models as text painters. Advances in Neural Information Processing Systems **36**, 9353–9387 (2023)
12. Chen, J., Huang, Y., Lv, T., Cui, L., Chen, Q., Wei, F.: Textdiffuser-2: Unleashing the power of language models for text rendering. In: European Conference on Computer Vision (2024)
13. Cheng, T.Y., Filoni, S., Jampani, V., Boss, M., Sharma, P.: ZeST: Zero-shot material transfer from a single image (2024)
14. Cheng, T.Y., Sharma, P., Boss, M., Jampani, V.: MARBLE: Material recomposition and blending in CLIP-Space. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2025)
15. Chung, J., Hyun, S., Heo, J.P.: Style injection in diffusion: A training-free approach for adapting large-scale diffusion models for style transfer. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 8795–8805 (2024)
16. Cui, C., Sun, T., Lin, M., Gao, T., Zhang, Y., Liu, J., Wang, X., Zhang, Z., Zhou, C., Liu, H., Zhang, Y., Lv, W., Huang, K., Zhang, Y., Zhang, J., Zhang, J., Liu, Y., Yu, D., Ma, Y.: Paddleocr 3.0 technical report (2025), https://arxiv.org/abs/2507.05595
17. Esser, P., Kulal, S., Blattmann, A., Entezari, R., Müller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., et al.: Scaling rectified flow transformers for high-resolution image synthesis. In: Forty-first international conference on machine learning (2024)
18. Fu, B., Yu, F., Liu, A., Wang, Z., Wen, J., He, J., Qiao, Y.: Generate like experts: Multi-stage font generation by incorporating font transfer process into diffusion models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 6892–6901 (2024)
19. Fu, B., Yu, F., Liu, A., Wang, Z., Wen, J., He, J., Qiao, Y.: Generate like experts: Multi-stage font generation by incorporating font transfer process into diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 6892–6901 (June 2024)

20. Fu, S., Tamir, N., Sundaram, S., Chai, L., Zhang, R., Dekel, T., Isola, P.: Dreamsim: Learning new dimensions of human visual similarity using synthetic data. arXiv preprint arXiv:2306.09344 (2023)
21. Fu, T.J., Hu, W., Du, X., Wang, W.Y., Yang, Y., Gan, Z.: Guiding instruction-based image editing via multimodal large language models. In: Proceedings of the International Conference on Learning Representations (ICLR) (2024)
22. Gao, Y., Lin, Z., Liu, C., Zhou, M., Ge, T., Zheng, B., Xie, H.: Postermaker: Towards high-quality product poster generation with accurate text rendering. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 8083–8093 (2025)
23. Google: Gemini 2.5 Flash Image (Nano Banana). Referenced in technical reports and comparative analyses, e.g., arXiv:2510.19808, arXiv:2510.26583 (2025)
24. Google: Gemini 3.1 pro (2026), `https://gemini.google.com`, large Language Model. Accessed: 2026-02-28
25. Google DeepMind, Google AI: Introducing Gemini 2.5 Flash Image, our state-of-the-art image model. `https://developers.googleblog.com/en/introducing-gemini-2-5-flash-image/` (August 2025), accessed: 2025-11-12
26. Gupta, A., Vedaldi, A., Zisserman, A.: Synthetic data for text localisation in natural images. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2315–2324 (2016)
27. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al.: Lora: Low-rank adaptation of large language models. ICLR **1**(2), 3 (2022)
28. Huang, X., Gauri, B., Ng, K.W., Ng, T., Xu, M., Liu, Z., Ren, W., An, Z., Zhou, Z., Qiu, H., Zhou, Y., He, S., Wang, Z., Xiang, T., Han, X.: Vecglypher: Unified vector glyph generation with language models. In: CVPR (2026)
29. Huang, Y., Xie, Z., Wang, Y., Yao, Y., Zhang, Q., Zhou, M.: SmartEdit: Exploring complex instruction-based image editing with multimodal large language models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2D2024)
30. Hunter, J.D.: Matplotlib: A 2d graphics environment. Computing in Science & Engineering **9**(3), 90–95 (2007). `https://doi.org/10.1109/MCSE.2007.55`
31. IMDb: Imdb: The internet movie database. `https://www.imdb.com/` (2025)
32. Labs, B.F.: Flux.1 fill [dev]. `https://huggingface.co/black-forest-labs/FLUX.1-Fill-dev` (2024–2025), accessed: June 30, 2026
33. Labs, B.F., Batifol, S., Blattmann, A., Boesel, F., Consul, S., Diagne, C., Dockhorn, T., English, J., English, Z., Esser, P., Kulal, S., Lacey, K., Levi, Y., Li, C., Lorenz, D., Müller, J., Podell, D., Rombach, R., Saini, H., Sauer, A., Smith, L.: FLUX.1 kontext: Flow matching for in-context image generation and editing in latent space (2025)
34. Labs, B.F., Batifol, S., Blattmann, A., Boesel, F., Consul, S., Diagne, C., Dockhorn, T., English, J., English, Z., Esser, P., et al.: Flux. 1 kontext: Flow matching for in-context image generation and editing in latent space. arXiv preprint arXiv:2506.15742 (2025)
35. Lan, R., Bai, Y., Duan, X., Li, M., Jin, D., Xu, R., Sun, L., Chu, X.: Flux-text: A simple and advanced diffusion transformer baseline for scene text editing. arXiv preprint arXiv:2505.03329 (2025)
36. Li, C., Taniguchi, Y., Lu, M., Konomi, S., Nagahara, H.: Cross-language font style transfer. Applied Intelligence **53**(15), 18666–18680 (2023)
37. Li, H., Lian, Z.: Hfh-font: Few-shot chinese font synthesis with higher quality, faster speed, and higher resolution. ACM Transactions on Graphics (TOG) **43**(6), 1–16 (2024)
38. Li, S., Kallidromitis, K., Gokul, A., Liao, Z., Kato, Y., Kozuka, K., Grover, A.: Omniflow: Any-to-any generation with multi-modal rectified flows. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 13178–13188 (2025)
39. Li, Z., Liu, Y., Liu, Q., Ma, Z., Zhang, Z., Zhang, S., Guo, Z., Zhang, J., Wang, X., Bai, X.: Monkeyocr: Document parsing with a structure-recognition-relation triplet paradigm (2025), `https://arxiv.org/abs/2506.05218`
40. Lian, L., Li, B., Yala, A., Darrell, T.: LLM-grounded diffusion: Enhancing prompt understanding of text-to-image diffusion models with large language models. Transactions on Machine Learning Research (TMLR) (2024)

41. Liu, X., Gong, C., Liu, Q.: Rectified flow: A general L_2-Wasserstein flow for generative modeling. In: The Eleventh International Conference on Learning Representations (ICLR) (2023)
42. Liu, Y., Lian, Z.: QT-Font: High-efficiency font synthesis via quadtree-based diffusion models. In: SIGGRAPH 2024 Conference Papers. pp. 1–10. ACM (2024)
43. Liu, Z., Liang, W., Liang, Z., Luo, C., Li, J., Huang, G., Yuan, Y.: Glyph-byt5: A customized text encoder for accurate visual text rendering. In: European Conference on Computer Vision. pp. 361–377. Springer (2024)
44. Mangrulkar, S., Gugger, S., Debut, L., Belkada, Y., Paul, S., Bossan, B.: PEFT: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft` (2022)
45. OpenAI: Introducing GPT-5. `https://openai.com/index/introducing-gpt-5/` (aug 2025), accessed: 2025-11-12
46. Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J., Rombach, R.: Sdxl: Improving latent diffusion models for high-resolution image synthesis. In: Proceedings of the International Conference on Learning Representations (ICLR) (2024)
47. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International conference on machine learning. pp. 8748–8763. PmLR (2021)
48. Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., Sutskever, I.: Zero-shot text-to-image generation. In: Proceedings of the 38th International Conference on Machine Learning (ICML). Proceedings of Machine Learning Research, vol. 139, pp. 8821–8831. PMLR (2021)
49. Rodriguez, J.A., Puri, A., Agarwal, S., Laradji, I.H., Rodriguez, P., Rajeswar, S., Vazquez, D., Pal, C., Pedersoli, M.: Starvector: Generating scalable vector graphics code from images and text (2024), `https://arxiv.org/abs/2312.11556`
50. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 10684–10695 (2022)
51. Selinger, P.: Potrace: A polygon-based tracing algorithm (2019), `https://potrace.sourceforge.net/`, accessed: 2025-11-20
52. Tschannen, M., Gritsenko, A., Wang, X., Naeem, M.F., Alabdulmohsin, I., Parthasarathy, N., Evans, T., Beyer, L., Xia, Y., Mustafa, B., et al.: Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. arXiv preprint arXiv:2502.14786 (2025)
53. Tuo, Y., Geng, Y., Bo, L.: Anytext2: Visual text generation and editing with customizable attributes. arXiv preprint arXiv:2411.15245 (2024)
54. Wang, H., Wang, Q., Bai, X., Qin, Z., Chen, A.: InstantStyle: Free lunch towards style-preserving in text-to-image generation (2024)
55. Wang, X., Darrell, T., Rambhatla, S.S., Girdhar, R., Misra, I.: InstanceDiffusion: Instance-level control for image generation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 6232–6242 (2024)
56. Wang, Z., Bao, J., Gu, S., Chen, D., Zhou, W., Li, H.: Designdiffusion: High-quality text-to-design image generation with diffusion models. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 20906–20915 (2025)
57. Wang, Z., Zhang, Z., Yao, Y., Zhang, Q., Zhou, M., Yao, T.: GenArtist: Multimodal LLM as an agent for unified image generation and editing. In: Advances in Neural Information Processing Systems (NeurIPS) (2024)
58. Wei, H., Sun, Y., Li, Y.: Deepseek-ocr: Contexts optical compression. arXiv preprint arXiv:2510.18234 (2025)
59. Wu, C., Li, J., Zhou, J., Lin, J., Gao, K., Yan, K., Yin, S.m., Bai, S., Xu, X., Chen, Y., et al.: Qwen-image technical report. arXiv preprint arXiv:2508.02324 (2025)
60. Wu, K., Otoo, E., Suzuki, K.: Optimizing two-pass connected-component labeling algorithms. Pattern Analysis and Applications **12**(2), 117–135 (2009)

61. Wu, T.H., Lian, L., Li, B., Wang, Z., Yala, A., Darrell, T.: Self-correcting LLM-controlled diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2024)
62. Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., Raffel, C.: Byt5: Towards a token-free future with pre-trained byte-to-byte models (2021)
63. Yang, Z., Peng, D., Kong, Y., Zhang, Y., Yao, C., Jin, L.: FontDiffuser: One-shot font generation via denoising diffusion with multi-scale content aggregation and style contrastive learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 38, pp. 6603–6611 (2024). https://doi.org/10.1609/aaai.v38i7.28482
64. Ye, H., Zhang, J., Liu, S., Han, X., Yang, W.: Ip-adapter: Text compatible image prompt adapter for text-to-image diffusion models. arXiv preprint arXiv:2308.06721 (2023)
65. Yeh, C.H., Wang, Y., Zhao, N., Zhang, R., Li, Y., Ma, Y., Singh, K.K.: Beyond simple edits: X-Planner for complex instruction-based image editing (2025)
66. Zhang, L., Rao, A., Agrawala, M.: Adding conditional control to text-to-image diffusion models. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 3836–3847 (2023)
67. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 586–595 (2018)
68. Zhou, Y., Gao, X., Chen, Z., Huang, H.: Attention Distillation: A unified approach to visual characteristics transfer. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2025)