arXiv:2611.32327v1 [cs.CV] 24 Nov 2026

FlowVecFont：通过潜在流匹配实现可缩放矢量字体生成

Takumu Fujioka[1] Gouhei Tanaka[1,2]1名古屋大学工学研究科计算机科学系，名古屋，466-8555，日本2东京大学国际神经智能研究中心，东京，113-0033，日本

t.fujioka.494@stn.nitech.ac.jp, gtanaka@nitech.ac.jp

摘要

字体的原生表示形式是矢量轮廓。许多现有的矢量字体生成研究依赖专门的生成或细化机制、位图表示，或外部预训练模型，以应对学习矢量表示所面临的挑战。这些方法往往会增加系统复杂度，带来额外的计算和实现成本，并引入对外部资源的依赖。我们表明，只需扩大简单模型的规模，无需外部预训练即可实现高质量的纯矢量字体生成。我们提出了*FlowVecFont*，这是一种基于潜在流匹配、仅使用矢量轮廓训练的可扩展矢量字体生成模型。基于*Transformer*的变分自编码器将字形序列映射为固定长度的连续潜在标记，从而支持标准的条件流匹配，用于生成矢量字形。该模型支持以类别标签或参考字形作为条件；后者使模型能够泛化至未见字符和字体。在一项实验中，我们使用从 *Google Fonts* 收集的 *500* 万个字形（包括非拉丁字符），结果表明，在不同条件策略下，生成质量均随模型规模扩大而提升。在拉丁字形对比实验中，参数量为 *8* 亿的 *FlowVecFont* 优于 *DeepVecFont-v2* 和 *DesigNet*，其生成质量可与经过预训练的 *270*亿参数的 *VecGlypher* 相媲美。我们还提供了 *TorchFont*，这是一个用于矢量字体的机器学习库。

1. 引言

开发字体需要专业知识和大量时间。尤其是开发中日韩（中文、日文和韩文）字体，由于包含的字符数量庞大，可能需要数年时间。近年来，这些挑战促使人们对基于机器学习的字体生成产生了越来越浓厚的兴趣。

表1. 用于表示矢量轮廓的绘图命令。

命令	参数	描述
MoveTo	x, y	在 $(x,\ y)$ 处开始新的子路径，但不绘制。
直线到	x, y	从当前点绘制一条直线到 $(x,\ y)$。
二次曲线到	c^x, c^y x, y	从以下位置绘制二次 B´ezier 曲线当前点绘制二次贝塞尔曲线至 $(x,\ y)$，控制点为 $(c^x,\ c^y)$。用于 TrueType（glyf）轮廓。
CurveTo	c_1^x, c_1^y c_2^x, c_2^y x, y	从当前点绘制一条三次贝塞尔曲线，至点(x, y)，控制点为(c_1^x, c_1^y)和(c_2^x, c_2^y)。用于 CFF/CFF2 轮廓。
Close	$\varnothing$	用一条从当前点到其起点。

字体生成与传统图像生成有很大不同，因为字体通常使用矢量表示，而非位图表示。位图图像以像素网格表示视觉内容，而矢量图像则使用线条和曲线来表示形状并勾勒其轮廓，因此字体能够以各种规模使用而不会损失质量。

表 1 列出了用于构建字体轮廓的绘图命令。字体轮廓由这些命令组成的长度可变的序列构成，类似于手写轨迹。由于矢量图形和位图图像具有根本不同的数据结构，为位图图像设计的传统方法难以理解和生成矢量图形。

表 2 比较了用于矢量图形和字体生成的代表性方法。学习矢量表示仍然具有挑战性，其中许多方法

表2。矢量图形与字体生成方法的比较。方法表示模型类别：变分自编码器（VAE）或编码器–解码器、扩散或流匹配，以及大语言模型（LLM）。字体支持表示训练和评估中不同字符的数量：少于1K、1K至10K，以及10K或以上。无位图表示没有位图输入或监督，不包括评估阶段。无需预训练表示训练或生成时不使用外部预训练权重，包括损失网络，但不包括仅用于评估的指标。叉号旁的名称表示具有代表性的预训练组件，并非完整列表。

方法	方法类别	字体支持	矢量输出	无位图	无需预训练
DeepSVG（2020年）[4]	VAE	拉丁文	✓	✓	✓
Aoki & Aizawa（2022年）[1]	编码器–解码器	中文	✓	✓	✓
DeepVecFont-v2（2023年）[24]	VAE	拉丁文、中文	✓	✗	✗ VGG-19损失 [20]
DualVector（2023年）[15]	VAE	拉丁文	✓	✗	✗ LPIPS [33]
DesigNet（2026年）[7]	VAE	拉丁文、中文	✓	✓	✓
SVGFusion（2024年）[27]	潜在扩散	✗	✓	✗	✗ DINOv2 [17]
VecFusion（2024年）[21]	扩散	多种	✓	✗	✓
Zhao 等人（2024年）[34]	扩散	拉丁文	✓	✗	✓
FontDiffuser（2024年）[29]	扩散	中文、韩文	✗	✗	✗ VGG-16 损失 [20]
FM-Font（2026年）[22]	流匹配	中文	✗	✗	✗ DINOv2 [17]
OmniSVG（2025年）[28]	多模态大语言模型	✗	✓	✗	✗ Qwen2.5-VL [2]
VecGlypher（2026年）[11]	多模态大语言模型	拉丁文	✓	✗	✗ Gemma 3 [5]
VecFontLLM（2026年）[31]	多模态大语言模型	中文	✓	✗	✗ StarCoder-1B [12]
FlowVecFont（本文提出）	潜在流匹配	多个	✓	✓	✓

通过专门的生成或细化机制、位图表示，或外部预训练来应对这一难题。VecFusion [21], 是一种具有代表性的基于扩散的方法，但仅使用矢量表示无法实现令人满意的生成质量。相反，它采用多阶段生成方法：首先生成位图图像，然后以此作为条件进行矢量生成。VecGlypher [11]和 VecFontLLM [31] 等基于语言模型的方法，依赖通过大规模外部预训练获得的知识，以此应对直接从矢量表示中学习的挑战。这些方法可能增加系统复杂性，并引入对外部资源的依赖。尤其是，它们需要额外的计算，但其优势往往是通过移除组件来评估的，而没有匹配计算预算。这使得人们难以确定性能提升在多大程度上源于这些设计选择，而非额外的计算。

作为一种替代方案，我们展示了通过扩展一个用于纯矢量字体生成的简单模型，也能实现具有竞争力的质量。我们提出了 FlowVecFont，这是一种基于潜在流匹配的可扩展矢量字体生成模型 [13], , 无需位图输入、位图监督或外部预训练权重。

我们的主要贡献如下：

- 我们提出了 FlowVecFont，该模型将流匹配应用于由 Transformer 变分自编码器学习得到的定长字形潜变量，从而将流计算与轮廓长度解耦。
- 我们通过同时以矢量参考字形作为字体风格和字符内容的条件，展示了对未见字体和字符的泛化能力。字体风格和字符内容。
- 在 Google Fonts 上的实验 [6] 表明，在不同条件策略下，随着模型规模扩大，生成质量都能持续提升。我们的 8 亿参数模型在拉丁字形上的表现可与一个经过预训练的 270 亿参数模型相媲美。

- 我们提供了 TorchFont，这是一个用于矢量字体的机器学习库（https://github.com/torchfont/torchfont）。

2. 相关工作

2.1. 矢量图形与字体生成

变分自编码器和编码器–解码器模型。 DeepSVG [4], 是一种早期的矢量图形和字体生成方法，引入了层次化 Transformer 变分自编码器，分别在不同层级对路径和绘图命令进行建模。后续方法基于 DeepSVG 的矢量表示或模型设计展开，包括 Aoki 和 Aizawa [1] 以及 DesigNet [7]。Aoki 和 Aizawa 利用 Transformer 编码器–解码器和几何损失，根据内容和风格参考生成中文矢量字体。DesigNet 通过对曲线连续性和轴对齐进行可微细化，扩展了层次化变分自编码器建模，并融入专业字体设计中使用的几何约束。

另一类研究，包括 DeepVecFont [23], DeepVecFont-v2 [24], 和 DualVector [15], , 提出使用位图表示进行矢量字体生成。DeepVecFont-v2 结合图像和序列特征，并采用基于 Transformer 的轮廓细化方法。DualVector 从字形图像中学习正向和负向路径，并通过布尔运算构建轮廓。这些

研究报告称，利用位图信息能够提升生成质量。

扩散与流匹配模型。 VecFusion [21]和 Zhao 等人 [34] 将扩散模型应用于矢量字体生成。VecFusion 首先生成位图字形和控制点场，再利用它们引导矢量生成。Zhao 等人联合建模矢量扩散与位图扩散，并在每个去噪步骤中通过交叉注意力交换特征。这两种方法在训练和生成过程中都依赖位图信息。

相关方法针对通用 SVG 生成或位图字体生成。SVGFusion [27] 将潜在扩散应用于通用 SVG 图形，而非字体字形，但它使用预训练 DINOv2 编码器提取的位图特征来学习潜在表示。FontDiffuser [29] 和 FM-Font [22] 分别将扩散和流匹配应用于位图字体生成。二者都无法生成矢量轮廓。

大语言模型。 SVG 是一种基于 XML 的文本格式，通过元素和属性描述图形，因此可以将 SVG 合成表述为语言模型的代码生成任务。OmniSVG [28] 通过微调 Qwen2.5-VL，使其在文本或图像条件下预测表示绘图命令、坐标和颜色的离散词元，从而将预训练视觉语言模型适配于通用 SVG 生成。

此方法随后被 VecGlypher [11] 和 VecFontLLM [31]扩展至矢量字体生成。VecGlypher 对 Gemma 3 进行微调，使其能够根据文本风格描述或参考字形图像自回归地生成 SVG 轮廓。VecFontLLM 则采用基于 StarCoder-1B 的多阶段生成流程，先预测锚点布局，再补全 B´ezier 控制点，以构成复杂的汉字字形。

2.2. 生成模型

去噪扩散模型 [10]通过学习逆转逐步加噪的过程来生成样本。流匹配 [13]的理念同样是将噪声逐步转换为数据，但它会直接学习预设概率路径上的速度场，其中包括与扩散过程相关的路径。通过对学得的速度场进行积分即可生成样本。整流流 [14] 是流匹配的一种形式，它在噪声与数据样本之间采用直线插值。

潜在扩散和潜在流匹配将表示学习与生成分离，在学习得到的潜在空间中对数据分布进行建模 [19]。DiT [18] 和 SiT [16] 等基于 Transformer 的模型展现了这种方法的可扩展性。我们的方法将这一框架应用于矢量绘图命令序列。

3. 方法

3.1. 概述

FlowVecFont 使用潜在流匹配生成矢量字形，并以字符内容和字体风格为条件。

对矢量轮廓进行直接生成建模，需要同时处理离散命令类型和连续坐标，因此需要专门的表示方法。VecFu- sion [21] 将路径归属和网格单元坐标编码为二进制位，并将其作为连续值进行扩散，在生成后再进行离散化。Zhao 等人 [34] 将命令类型编码为二维数值代码，并对这些代码和几何参数进行联合扩散。此外，当扩散直接作用于轮廓元素时，提高最大序列长度会增加每个去噪步骤的计算成本。

FlowVecFont通过使用Transformer变分自编码器和潜在流匹配来应对这些挑战。VAE编码器通过自注意力将轮廓信息聚合为固定数量的可学习潜在令牌，从而将可变长度的命令序列压缩为固定规模的表示。它将离散命令类型和连续坐标联合编码到学习得到的连续潜在空间中。随后，我们在该空间中训练一个以字符内容和字体风格为条件的流匹配模型。在推理时，流模型将高斯噪声转换为字形潜变量，再由VAE解码器将其转换为绘图命令序列。

这种设计使流模型能够完全在固定大小的连续表示上运行，而无需直接对离散命令类型和坐标进行建模。借助固定数量的潜在令牌，可以在不改变流架构或其每步计算成本的情况下，在变分自编码器一侧增加所支持的轮廓序列长度。这种形式使矢量字体生成能够使用与图像潜在流匹配基本相同的生成架构。

3.2. 预处理

我们首先排除轮廓序列长度超过最大长度 L 的字形。字体使用不同的坐标规模，因此我们随后将轮廓坐标除以每种字体的每 em单位数，以获得统一的规模。所得坐标大致位于 $[0, 1]$ 中。

我们使用六种命令类型表示字形轮廓：MoveTo、直线到、CurveTo、Close、结束和填充。二次 B´ezier 曲线（QuadTo）会被转换为三次 B´ezier 曲线（CurveTo）。结束命令终止序列，而填充命令则填充未使用的位置，以便进行批处理。

我们将每个字形的命令序列定义如下：

$$\mathcal{G} = \big[(t_1, \mathbf{p}_1), \ldots, (t_N, \mathbf{p}_N)\big], \tag{1}$$

其中， N 为序列长度， t_i 表示命令

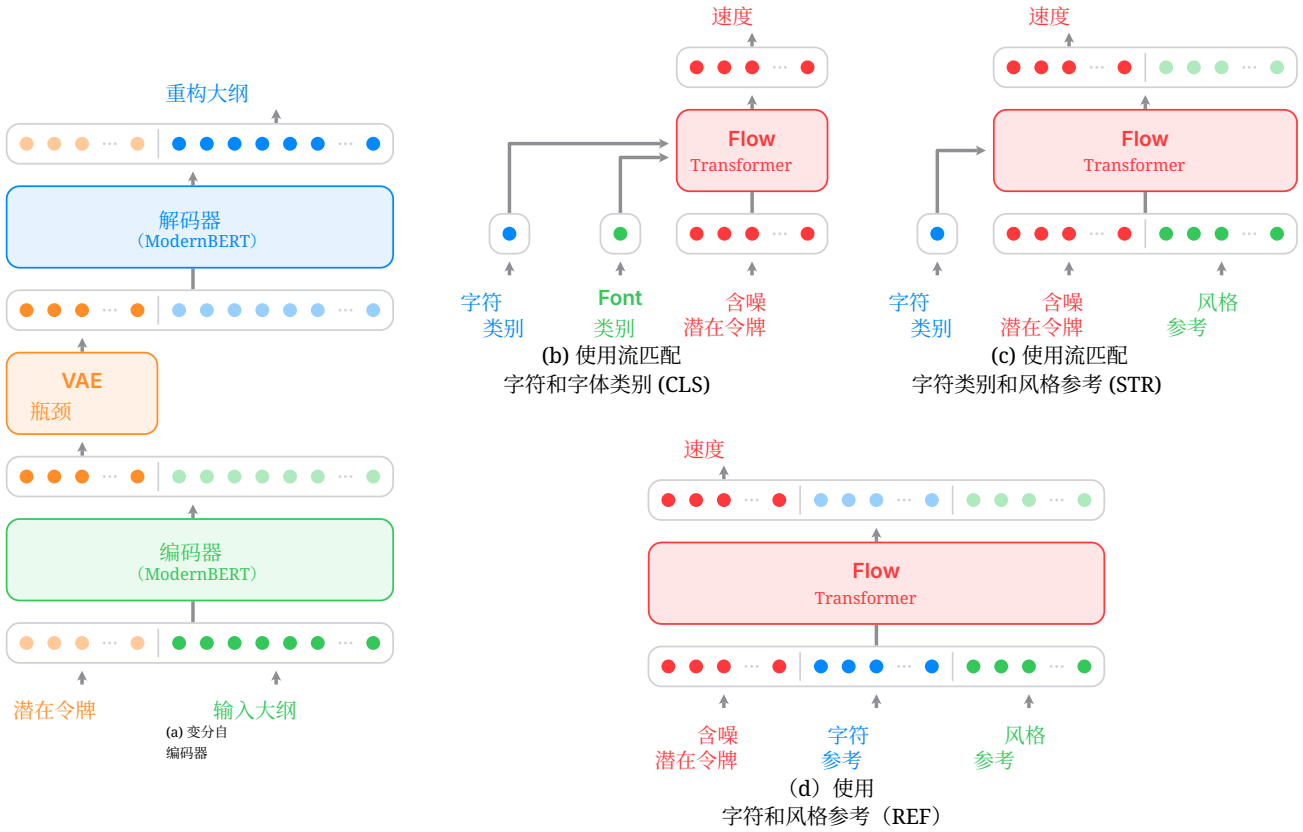


图 1. FlowVecFont 架构。（a）变分自编码器对矢量轮廓进行编码和重建。（b）–（d）潜在流匹配使用不同的条件策略生成字形表示。

类型，且 $\mathbf{p}_i$ 是其坐标向量。坐标向量$\mathbf{p}_i \in \mathbb{R}^6$ 定义如下（见表 1）：

$$\mathbf{p}_i = (c^x_{i,1}, c^y_{i,1}, c^x_{i,2}, c^y_{i,2}, x_i, y_i)^\top. \tag{2}$$

其中包含最多两个控制点的坐标以及一个端点。未使用的坐标条目均设为零。

随后，我们应用轮廓归一化和数据增强。这些变换包括轮廓绕行调整、子路径起点选择、子路径重新排序、几何变换、重叠移除、线段拆分以及子路径丢弃。变换的选择、变换强度，以及将其用于归一化还是随机数据增强，取决于模型类型、条件方案以及训练或评估设置。

这些变换可能会增加序列长度。当序列超过最大长度 L 时，我们会移除完整的子路径，直到序列长度符合要求。最后，我们使用填充标记将序列填充至批处理所需的最大长度。

3.3. 变分自编码器

为了学习用于生成的潜在表示，我们训练一个变分自编码器，以按照输入命令序列的原始路径排序进行重建。不同于 DeepSVG [4], ，我们不强制要求对路径排序保持不变性。如图1a所示，编码器和解码器采用 TiTok 的一维图像标记器设计 [30]。二者均采用 ModernBERT 架构 [26] ，并从头开始训练。解码器是非自回归的。

如图2所示，我们分别嵌入命令类型和坐标。我们将得到的 d维命令令牌定义如下：

$$\mathbf{e}_i = \mathbf{W}_{\text{type}}\,\text{onehot}(t_i) + \mathbf{W}_{\text{coord}}\mathbf{p}_i, \tag{3}$$

其中，$\text{onehot}(t_i) \in \mathbb{R}^6$ 是 t_i的独热编码，而$\mathbf{W}_{\text{type}} \in \mathbb{R}^{d\times 6}$ 和 $\mathbf{W}_{\text{coord}} \in \mathbb{R}^{d\times 6}$ 是可学习矩阵。坐标保持连续，不进行量化。

编码器将命令序列压缩为 K个潜在令牌。我们将 K 个可学习令牌与命令嵌入拼接起来，并使用自注意力处理所得序列。潜在令牌位置上的输出用于参数化潜在表示$\mathbf{Z} \in \mathbb{R}^{K\times d_z}$，其中 d_z 是每个令牌的维度。

解码器通过自注意力从潜在表示中重建 L 命令令牌，然后通过反嵌入层将其映射为命令类型逻辑值和六维坐标。

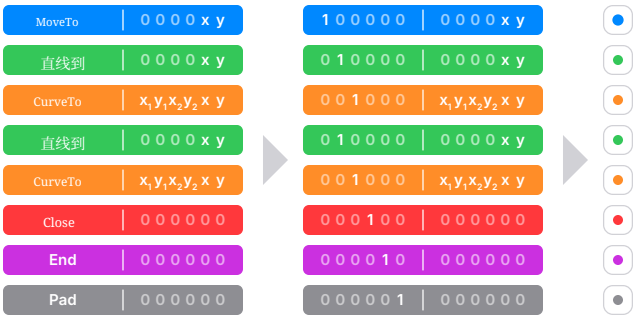


图2. 绘图命令嵌入。命令类型采用独热编码后进行嵌入，而坐标则通过可学习线性层进行嵌入。将这两种嵌入相加，得到一个命令令牌。

我们将变分自编码器的训练目标定义如下：

$$\mathcal{L}_{\text{VAE}} = \lambda_{\text{type}}\mathcal{L}_{\text{type}} + \lambda_{\text{coord}}\mathcal{L}_{\text{coord}} + \lambda_{\text{KL}}\mathcal{L}_{\text{KL}}, \quad (4)$$

其中， $\mathcal{L}_{\text{type}}$ 是非填充位置上的命令类型交叉熵损失， $\mathcal{L}_{\text{coord}}$ 是下文定义的坐标损失， $\mathcal{L}_{\text{KL}}$ 是标准的变分自编码器KL正则化损失。权重 λ_{type}、 λ_{coord}和 λ_{KL} 控制这三种损失的贡献。

此前的工作（例如 DesigNet） [7]使用均方误差进行坐标重建。受栅格图像生成中采用 L1重建损失的启发 [19], ，我们对坐标使用 L1损失，以防止模型崩溃。对于每个字形，我们将掩码L1坐标损失定义如下：

$$\mathcal{L}_{\text{coord}} = \sum_{i=1}^{L} \|\mathbf{m}(t_i) \odot (\hat{\mathbf{p}}_i - \mathbf{p}_i)\|_1 , \quad (5)$$

其中， $\hat{\mathbf{p}}_i$ 是预测坐标向量， $\mathbf{m}(t_i)$是一个掩码，用于选取目标命令所使用的坐标。我们将此掩码定义如下：

$$\mathbf{m}(t) = \begin{cases} (0,0,0,0,1,1)^\top & t \in \{\text{MoveTo}, \text{LineTo}\}, \\ (1,1,1,1,1,1)^\top & t = \text{CurveTo}, \\ (0,0,0,0,0,0)^\top & t \in \{\text{Close}, \text{End}, \text{Pad}\}. \end{cases} \quad (6)$$

两种重建损失都在每个字形内求和，并在批次上取平均。

3.4. 流匹配

我们采用整流流 [14]，在训练好的 VAE 潜在空间中进行条件流匹配 [13] 。对于每个训练字形 $\mathcal{G}$，我们使用 VAE 编码器获取其潜在表示$\mathbf{Z}_0 \in \mathbb{R}^{K \times d_z}$ 。我们通过线性插值定义含噪潜变量，如下所示：

$$\mathbf{Z}_t = (1-t)\mathbf{Z}_0 + t\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (7)$$

其中 $t \sim \mathcal{U}(0,\ 1)$， $t = 0$ 对应数据， $t = 1$ 对应噪声。对于每个采样对 $(\mathbf{Z}_0,\ \boldsymbol{\epsilon})$，目标速度 $\mathbf{u}$ 是$\mathbf{Z}_t$ 随时间的变化率，如下所示：

$$\mathbf{u} = \frac{\mathrm{d}\mathbf{Z}_t}{\mathrm{d}t} = \boldsymbol{\epsilon} - \mathbf{Z}_0. \quad (8)$$

我们训练速度场 $v_\theta(\mathbf{Z}_t, t, c)$ 来预测 $\mathbf{u}$，其中θ 表示可学习的模型参数， c表示字符内容和字体风格条件。我们按照 DiT [18]，使用 Transformer 对 v_θ 进行参数化。我们将均方误差目标定义如下：

$$\mathcal{L}_{\text{FM}} = \mathbb{E}\left[\|v_\theta(\mathbf{Z}_t, t, c) - \mathbf{u}\|^2\right]. \quad (9)$$

在推理时，我们初始化 $\mathbf{Z}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$。相应的常微分方程定义如下：

$$\frac{\mathrm{d}\mathbf{Z}_t}{\mathrm{d}t} = v_\theta(\mathbf{Z}_t, t, c). \quad (10)$$

我们使用欧拉方法沿时间反向求解该方程 $t = 0$，将初始噪声传输至条件潜在数据分布。VAE解码器将得到的潜变量转换为绘制命令序列。

训练期间，我们使用无分类器引导（CFG），随机丢弃字符条件和风格条件 [9], 。生成时，我们结合有条件和无条件的速度预测，以控制条件的强度。

3.5. 条件策略

如图 1b–1d所示，我们比较了三种用于字符内容和字体风格的条件策略：字符和字体类别（CLS）、字符类别与风格参考（STR），以及字符和风格参考（REF）。风格参考是目标字体中的字形，而字符参考则是其他字体中的目标字符字形。

对于类别条件，我们遵循 DiT [18]，通过 AdaLN-Zero 注入可学习的类别嵌入。

对于参考条件，我们使用与目标字形相同的、经过训练的 VAE 编码器，对每个矢量参考字形进行编码。采样期间，参考潜变量保持固定且不含噪声。我们将参考潜变量投影到 Transformer维度，并将其与含噪目标令牌拼接，作为上下文条件。随后，通过自注意力对拼接后的序列进行联合处理。位置嵌入用于标识潜变量令牌的位置，而学习得到的角色嵌入和参考嵌入则用于区分目标、风格参考、字符参考，以及各组内的单个参考。我们仅使用目标令牌位置上的输出预测速度。

类别条件依赖于为各个类别ID学习的嵌入，在这种形式下无法直接处理未见字体或字符。相反，参考条件通过字形轮廓指定风格和内容，无需为每种字体或

字符学习嵌入。因此，在给定合适参考的情况下，STR原则上可以处理未见字体，而REF可以同时处理未见字体和未见字符。

4. 实验

4.1. 实现细节

VAE编码器和解码器采用 ViT-Tiny 的模型规模配置 [32]。对于每种条件策略，我们使用 DiT-S、DiT-B、DiT-L 和 DiT-XL 的模型规模配置训练流模型 [18]。我们根据条件策略和模型规模为各变体命名，例如 FlowVecFont-CLS/B 或 FVF-CLS/B。

命令序列的最大长度为 $L = 256$。变分自编码器使用 $K = 32$个潜在令牌，每个令牌的维度为 $d_z = 16$ 。变分自编码器损失的权重为 $\lambda_{\text{type}} = 1$、$\lambda_{\text{coord}} = 3$和 $\lambda_{\text{KL}} = 10^{-4}$。

我们使用 AdamW 和恒定学习率 10^{-4} 以及 BF16混合精度训练变分自编码器和流模型。变分自编码器训练 400K 步，随后每个流模型训练 1M 步。变分自编码器的批次大小为 64；CLS、STR 和 REF 的批次大小分别为 256、64 和 32。我们对模型权重使用衰减率为 0.9999 的指数移动平均。

对于参考条件，每种参考类型使用四个字形。我们从各组中随机抽取参考字形，不重复抽取，并排除目标字形。对于 CFG，训练期间以 0.1 的概率丢弃条件；推理时的引导缩放为 3.0，使用 32 个欧拉步数。

4.2. 数据集与评估

我们使用来自 Google Fonts [6], 的矢量字形，并将每个字体面–代码点对视为一个样本。我们收集以 Apache许可证、SIL Open Font License 和 Ubuntu字体许可证发布的 TTF文件，但排除 Adobe Blank。我们收集映射到码点的轮廓字形，排除未编码连字和替代字形，且不将字符集限制在特定语言或文字体系。排除长度超过 L 的字形后，数据集包含约 490 万个样本，来自 3800 个字体字形和 10.9 万个不同码位。每个可变字体均计为一个字体面，不受其变化轴设置影响。

我们采用位图字体生成中使用的已见/未见划分方案，例如 FontDiffuser [29], 用于矢量字体生成。根据目标字体和字符在训练期间是否出现过，我们定义四个组：已见字体和已见字符（SFSC）、已见字体和未见字符（SFUC）、未见字体和已见字符（UFSC），以及未见字体和未见字符（UFUC）。

字体字形和字符代码点分别随机分区，其中 95% 被指定为已见，5%

表 3. 按条件策略划分的评估数据划分、参考池和样本数量。Train 表示 SFSC 训练子集。

策略	目标	风格参考	字符参考	样本
CLS	SFSC	—	—	44.5K
STR	SFSC	训练	—	44.5K
STR	UFSC	UFSC	—	211.8K
REF	SFSC	训练	训练	43.9K
REF	SFUC	训练	SFUC	219.5K
REF	UFSC	UFSC	训练	209.6K
REF	UFUC	UFSC	SFUC	10.3K

为未见。SFSC 组进一步按 98%、1% 和 1% 的比例划分为训练集、验证集和测试子集。在结果中，SFSC 表示该组的测试子集。所有模型均使用相同的分区。

表3总结了 CLS、STR 和 REF 的评估数据划分与参考池。省略参考数量不足的目标。

我们使用对称倒角距离（CD） [3], 学习感知图像块相似度（LPIPS） [33], Fr´echet Inception 距离（FID） [8], 和评估失败率来评估字形。评估失败包括空轮廓以及评估过程中出现的某些错误。质量指标不包括失败样本及其目标。

4.3. 条件策略与模型扩展

图3 比较了 SFSC 上各条件策略和模型规模下的 VAE 重建与生成字形。重建结果与目标轮廓几乎完美重合。较小的模型会产生明显的伪影，尤其是在复杂字形中，例如笔画较多的中文字；而较大的模型通常能生成更干净的轮廓。REF 能够生成形态良好的字形，但其轮廓与目标不同。

如表4所示，变分自编码器的倒角距离为 0.0029，LPIPS 为 0.0045，且没有评估失败。这些重建误差远低于生成误差，表明变分自编码器为潜变量生成提供了合适的表示。对于生成任务，在每种条件策略下，随着模型规模从 S 增至 XL，所有四项指标均持续改善。在 SFSC 上，CLS 在每种模型规模下的四项指标中均表现最佳。不过，各策略使用的批次大小不同，因此结果未必能反映单纯由条件策略带来的影响。

图 4 展示了 FID 与估算的流模型训练计算量之间的关系。我们按照 DiT [18]估算训练浮点运算次数。对于每个模型和条件策略，FID 都会随着训练计算量的增加而降低，并且更大的模型在训练结束时能够达到更低的 FID。这表明，生成质量会随着模型规模和计算量的扩展而提升。对于 CLS，随着训练

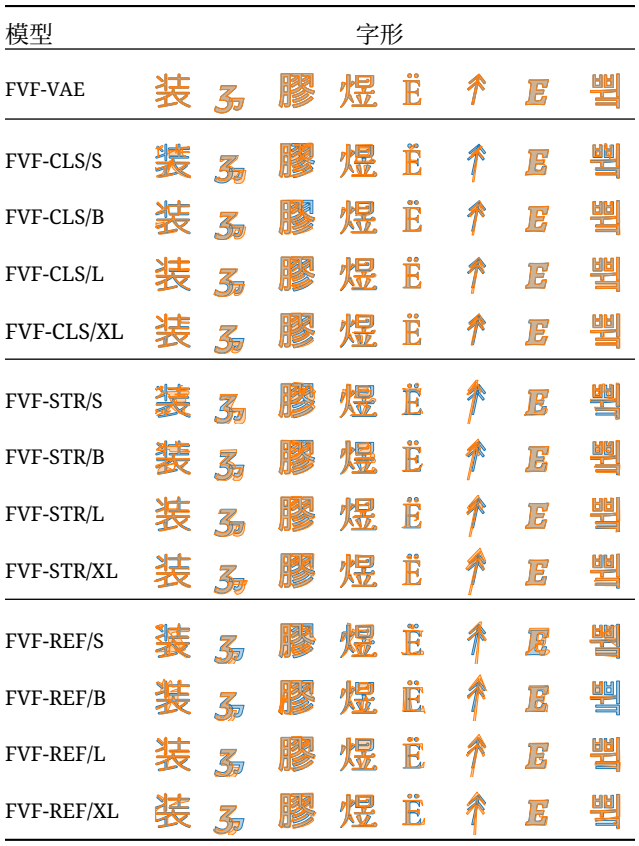


图3. SFSC 上的变分自编码器重建字形与生成字形。真实轮廓（蓝色）与重建结果或生成字形（橙色）叠加显示。各列使用相同的目标；样本并非经过挑选。

计算量增加，B 超过了 S，L 超过了 B。这表明，更大的模型能够更高效地利用训练计算量，这与 DiT 的观察结果一致。

4.4. 对未见字体和字符的泛化

图 5展示了 FlowVecFont-REF/B 在全部四个测试划分上生成的字形。在各个划分中，拉丁字母和符号通常生成效果良好，而中文字往往会出现伪影。从这些示例中，看不出各划分之间存在明显的视觉质量差异。

表 5显示，各划分上的倒角距离相近，而未见字体划分上的 LPIPS 和 FID 高于 SFSC。UFUC 上的失败率也最高。尽管如此，模型在未见字体划分上的表现仍接近 SFSC，展现了参考条件在未见字体和字符上泛化矢量字形生成的潜力。

4.5. 与现有方法的比较

我们将 FlowVecFont 与选定的、公开提供权重的现有方法进行比较：英文版 DeepVecFont-v2（DVF-v2）检查点、DesigNet 模型，以及

表4. SFSC 上的 VAE 重建与生成结果。规模表示总参数量，包括共享冻结 VAE。失败率表示评估失败率（%）。

模型	Size	CD ↓	LPIPS ↓	FID ↓	失败率 ↓
FVF-VAE	14M	0.0029	0.0045	0.09	0.00
FVF-CLS/S	90M	0.0194	0.1135	23.11	2.12
FVF-CLS/B	230M	0.0157	0.0853	11.86	1.17
FVF-CLS/L	586M	0.0136	0.0685	6.51	0.74
FVF-CLS/XL	817M	0.0131	0.0657	5.86	0.55
FVF-STR/S	88M	0.0279	0.1554	35.09	3.11
FVF-STR/B	227M	0.0267	0.1448	28.20	1.89
FVF-STR/L	582M	0.0209	0.1016	11.53	1.43
FVF-STR/XL	813M	0.0204	0.0991	10.62	1.18
FVF-REF/S	47M	0.0292	0.1611	37.03	3.80
FVF-REF/B	144M	0.0268	0.1414	26.87	2.67
FVF-REF/L	471M	0.0238	0.1166	14.80	1.94
FVF-REF/XL	688M	0.0232	0.1123	12.72	1.71

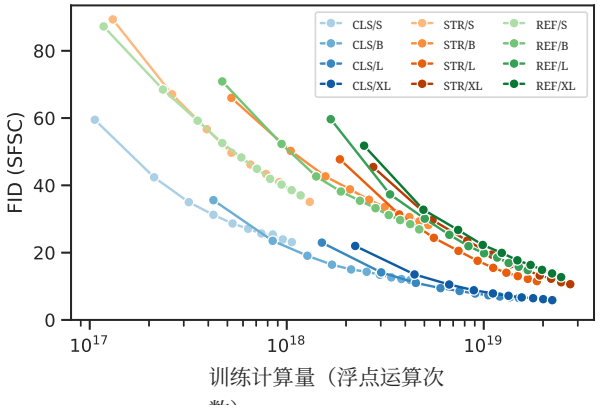


图 4。SFSC 上的 FID 与估计的流模型训练计算量之间的关系。每条曲线表示一个模型，其检查点每经过 100K 训练步数保存一次。

使用参考图像的 VecGlypher-27B。我们采用 UFSC 测试集划分，因此 FlowVecFont 训练期间未见过这些目标字体。比较方法共同支持的字符集包括大写和小写拉丁字母。我们评估满足所有方法表示约束的字形，并排除 DesigNet 固定的参考字符。我们使用 FlowVecFont-STR/XL 来匹配比较设置，其中字符身份和风格参考共同指定目标字形。由于评估协议与先前实验不同，这些分数不能直接比较。

DeepVecFont-v2、VecGlypher 和 FlowVecFont 对每个目标都使用相同的四个风格参考，而 DesigNet 使用其固定的“Hamburge”参考。DeepVecFont-v2 根据预测位图，从 20 个矢量候选中选出一个。

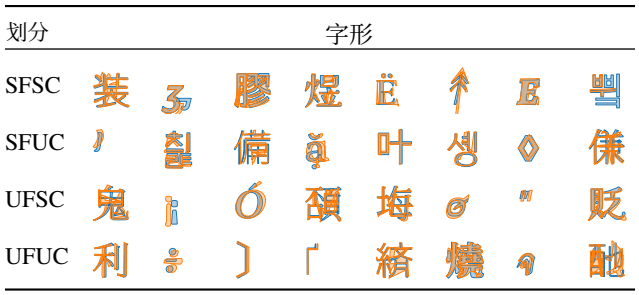


图5。FVF-REF/B 在四个测试划分上生成的字形。真实值以蓝色显示，生成字形以橙色显示。样本并非经过精选。

表5。FVF-REF/B 在四个测试划分上的结果。

划分	CD ↓	LPIPS ↓	FID ↓	失败率 ↓
SFSC	0.0268	0.1414	26.87	2.67
SFUC	0.0264	0.1428	28.22	2.66
UFSC	0.0273	0.1473	32.08	3.04
UFUC	0.0268	0.1485	35.10	3.39

图6 显示，在所展示的示例中，FlowVecFont 生成的字形轮廓与目标十分接近。表6 显示，FlowVecFont 在所有报告的指标上均优于 DeepVecFont-v2 和 DesigNet，而 VecGlypher 的质量评分最高。FlowVecFont 和 VecGlypher 均未出现评估失败。FlowVecFont 拥有 8 亿个参数，远少于 VecGlypher 的 270 亿个参数。除了 Google Fonts，VecGlypher 还使用一个未公开的、包含 3.9 万种商业字体的数据集进行训练；与 FlowVecFont 不同，其训练数据中还包含部分目标字体和字形。尽管存在这些差异，FlowVecFont 仍取得了具有竞争力的质量表现，证明了其在未见字体上进行矢量字形生成的有效性。

4.6. 局限性

VecGlypher [11]采用字体家族级划分，并针对拉丁文进行筛选和去重，以评估模型在不同字体家族间的泛化能力；但要将这一方法直接扩展到所有 Google Fonts 并不容易。单纯采用字体家族级划分，也可能导致不同划分之间存在共享轮廓，因为相关字体或重新打包的字体（如 Noto 变体）可能属于不同的字体家族。因此，我们采用更简单的字体面级划分，同一字体家族中的不同字体面可以被分到不同分区。不同码位可能对应相同或相似的字形，例如相似的拉丁字母和西里尔字母，或中日韩兼容表意文字。在一些非文本字体和 TeX 数学字体中，码位并不对应其通常表示的字符。划分数据集时，我们没有考虑这些情况。高度装饰性的字体（如 Rubik Pixels）中的字形可能会超过序列长度上限。

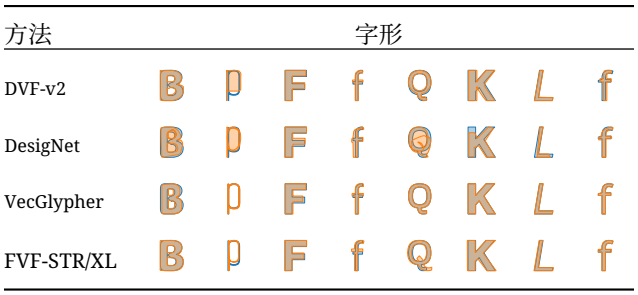


图6。在 UFSC 上与现有方法进行定性比较。真实值以蓝色显示，生成字形以橙色显示。各列使用相同的目标；样本并非经过精选。

表6。使用来自 74 个 Google Fonts 字体面的 3,170 个 UFSC 字形与现有方法进行比较。质量指标是在使用每个字形自身的边界框进行归一化后计算的。

方法	Size	CD ↓	LPIPS ↓	FID ↓	失败率 ↓
DVF-v2	121M	0.0211	0.1627	24.99	0.41
DesigNet	14M	0.0269	0.2146	50.03	0.63
VecGlypher	27B	**0.0129**	**0.0849**	**3.27**	**0.00**
FVF-STR/XL	813M	0.0134	0.0917	4.86	**0.00**

5. 结论

我们介绍了 FlowVecFont，这是一种用于纯矢量字体生成的可扩展模型。其 Transformer 变分自编码器将字形映射为固定长度的潜在令牌，从而能够进行流匹配，同时使流计算与轮廓长度脱钩。实验表明，在不同条件策略下，随着模型规模扩大，生成质量也会提升；此外，矢量参考还支持为未见字体和字符进行生成。在拉丁字形上，FlowVecFont 的表现优于 DeepVecFont-v2 和 DesigNet，生成质量也能与经过预训练的 270亿参数的 VecGlypher 相媲美。这些结果表明，基于大规模矢量数据训练简单模型并扩大其规模，是一种颇具吸引力的方案，可替代依赖位图表示或外部预训练来实现高质量矢量字体生成的做法。

致谢

本研究部分获得日本学术振兴会科研补助金（项目编号：JP25K24744）的资助。

我们使用 Codex（OpenAI）和 Claude Code（Anthropic）以交互方式开展写作、编码、实验和制图，并对其输出进行了审阅和修订。一些多步骤任务，例如文献调研和实验监控，则由工具自主完成。研究想法源自我们自身，这些工具仅帮助我们对其进行完善。我们对所有内容承担全部责任。

参考

[1] Haruka Aoki 和 Kiyoharu Aizawa。使用 Transformer 为汉字生成 SVG 矢量字体。在《2022年 IEEE 国际图像处理会议》（ICIP）上，第 646–650 页，2022年。2[2] Shuai Bai、Keqin Chen、Xuejing Liu、Jialin Wang、Wenbin Ge、Sibo Song、Kai Dang、Peng Wang、Shijie Wang、Jun Tang、Humen Zhong、Yuanzhi Zhu、Mingkun Yang、Zhao- hai Li、Jianqiang Wan、Pengfei Wang、Wei Ding、Zheren Fu、Yiheng Xu、Jiabo Ye、Xi Zhang、Tianbao Xie、Zesen Cheng、Hang Zhang、Zhibo Yang、Haiyang Xu 和 Jun- yang Lin。Qwen2.5-VL 技术报告。arXiv 预印本 arXiv:2502.13923，2025年。2[3] Harry G. Barrow、Jay M. Tenenbaum、Robert C. Bolles 和 Helen C. Wolf。参数化对应与倒角匹配：两种新的图像匹配技术。在第五届国际人工智能联合会议论文集上，第 659–662 页，1977 年。6, 14[4] Alexandre Carlier、Martin Danelljan、Alexandre Alahi 和 Radu Timofte。DeepSVG：用于矢量图形动画的层次生成网络。在《神经信息处理系统进展》上，第 16351–16361 页，2020 年。2, 4[5] Gemma Team 等。Gemma 3 技术报告。arXiv 预印本 arXiv:2503.19786，2025年。2[6] 谷歌。Google Fonts 字体文件。https://github.com/ google/fonts，未注明日期。2, 6[7] Tomas Guija-Valiente 和 Iago Su´arez。DesigNet：像设计师一样学习绘制矢量图形。《计算机与图形学》，137:104627，2026年。2, 5, 13[8]Martin Heusel、Hubert Ramsauer、Thomas Unterthiner、Bernhard Nessler 和 Sepp Hochreiter。通过双时间尺度更新规则训练的生成对抗网络收敛于局部纳什均衡。在《神经信息处理系统进展》上，第 6626–6637 页，2017 年。6, 14[9] Jonathan Ho 和 Tim Salimans。无分类器扩散引导。在 NeurIPS 深度生成模型与下游应用研讨会上，2021年。5[10] Jonathan Ho、Ajay Jain 和 Pieter Abbeel。去噪扩散概率模型。在《神经信息处理系统进展》上，第 6840–6851 页，2020 年。3[11] Xiaoke Huang、Bhavul Gauri、Kam Woh Ng、Tony Ng、Mengmeng Xu、Zhiheng Liu、Weiming Ren、Zhaochong An、Zijian Zhou、Haonan Qiu、Yuyin Zhou、Sen He、Ziheng Wang、Tao Xiang 和 Xiao Han。VecGlypher：使用语言模型统一生成矢量字形。在 IEEE/CVF 计算机视觉与模式识别会议论文集上，第 24373–24383 页，2026年。2, 3, 8[12] Raymond Li、Loubna Ben Allal、Yangtian Zi、Niklas Muen- nighoff、Denis Kocetkov、Chenghao Mou、Marc Marone、Christopher Akiki、Jia Li、Jenny Chim、Qian Liu、Evgenii Zheltonozhskii、Terry Yue Zhuo、Thomas Wang、Olivier De- haene、Mishig Davaadorj、Joel Lamy-Poirier、Jo˜ao Monteiro、Oleh Shliazhko、Nicolas Gontier、Nicholas Meade、Armel Zebaze、Ming-Ho Yee、Logesh Kumar Umapathi、Jian Zhu、Benjamin Lipkin、Muhtasham Oblokulov、Zhiruo Wang、Rudra Murthy、Jason Stillerman、Siva Sankalp Patel、Dmitry Abulkhanov、Marco Zocca、Manan Dey、Zhihan Zhang、Nour Fahmy、Urvashi Bhattacharyya、Wenhao Yu、Swayam Singh、Sasha Luccioni、Paulo Villegas、Maxim Kunakov、Fedor Zh-danov、Manuel Romero、Tony Lee、Nadav Timor、Jennifer Ding、Claire Schlesinger、Hailey Schoelkopf、Jan Ebert、Tri Dao、Mayank Mishra、Alex Gu、Jennifer Robinson、Carolyn Jane Anderson、Brendan Dolan-Gavitt、Danish Contractor、Siva Reddy、Daniel Fried、Dzmitry Bahdanau、Yacine Jernite、Carlos Mu˜noz Ferrandis、Sean Hughes、Thomas Wolf、Arjun Guha、Leandro von Werra 和 Harm de Vries。StarCoder：愿源代码与你同在！《机器学习研究汇刊》，2023年。2[13] Yaron Lipman、Ricky T. Q. Chen、Heli Ben-Hamu、Maximilian Nickel 和 Matt Le。生成建模的流匹配。载于《国际学习表征会议》，2023年。2、3、5[14] Xingchao Liu、Chengyue Gong 和 Qiang Liu。流动笔直且快速：利用整流流学习生成和迁移数据。载于《国际学习表征会议》，2023年。3、5[15] Ying-Tian Liu、Zhifei Zhang、Yuan-Chen Guo、Matthew Fisher、Zhaowen Wang 和 Song-Hai Zhang。DualVector：利用双重部分表示进行无监督矢量字体合成。载于《IEEE/CVF计算机视觉与模式识别会议论文集》，第14193–14202页，2023年。2。[16] Nanye Ma、Mark Goldstein、Michael S. Albergo、Nicholas M. Boffi、Eric Vanden-Eijnden 和 Saining Xie。SiT：利用可扩展插值变换器探索流与基于扩散的生成模型。载于《欧洲计算机视觉会议》，第23–40页，2024年。3[17] Maxime Oquab、Timoth´ee Darcet、Th´eo Moutakanni、Huy V. Vo、Marc Szafraniec、Vasil Khalidov、Pierre Fernandez、Daniel Haziza、Francisco Massa、Alaaeldin El-Nouby、Mahmoud Assran、Nicolas Ballas、Wojciech Galuba、Russell Howes、Po-Yao Huang、Shang-Wen Li、Ishan Misra、Michael Rabbat、Vasu Sharma、Gabriel Synnaeve、Hu Xu、Herv´e J´egou、Julien Mairal、Patrick Labatut、Armand Joulin 和 Piotr Bojanowski。DINOv2：无需监督学习稳健的视觉特征。《机器学习研究汇刊》，2024年。2[18] William Peebles 和 Saining Xie。利用 Transformer实现可扩展扩散模型。载于《IEEE/CVF国际计算机视觉会议论文集》，第4195–4205页，2023年。3、5、6。[19]Robin Rombach、Andreas Blattmann、Dominik Lorenz、Patrick Esser 和 Bj¨orn Ommer。利用潜在扩散模型进行高分辨率图像合成。载于《IEEE/CVF计算机视觉与模式识别会议论文集》，第10684–10695页，2022年。3、5[20] Karen Simonyan 和 Andrew Zisserman。用于大规模图像识别的超深卷积网络。载于《国际学习表征会议》，2015年。2[21]Vikas Thamizharasan、Difan Liu、Shantanu Agarwal、Matthew Fisher、Michael Gharbi、Oliver Wang、Alec Jacobson 和 Evangelos Kalogerakis。VecFusion：利用扩散进行矢量字体生成。载于《IEEE/CVF

计算机视觉与模式识别会议论文集》，第7943–7952页，2024年。2、3[22] Xiaoyan Wang、Tao Liu、Qun Cao 和 Shenglan Peng。FM-Font：利用解耦双分支编码进行流匹配的少样本中文字体生成。《IEEE Access》，14：105976–105990，2026年。2、3[23] Yizhi Wang 和 Zhouhui Lian。DeepVecFont：通过双模态学习合成高质量矢量字体。《ACM图形学汇刊》，40(6)：265:1–265:15，2021年。2[24] Yuqing Wang、Yizhi Wang、Longhui Yu、Yuesheng Zhu 和 Zhouhui Lian。DeepVecFont-v2：利用Transformer合成质量更高的矢量字体。载于《IEEE/CVF计算机视觉与模式识别会议论文集》，第18320–18328页，2023年。2、13[25] Zhou Wang、Alan C. Bovik、Hamid R. Sheikh 和 Eero P. Simoncelli。图像质量评估：从误差可见性到结构相似性。《IEEE图像处理汇刊》，13(4)：600–612，2004年。14[26] Benjamin Warner、Antoine Chaffin、Benjamin Clavi´e、Orion Weller、Oskar Hallstr¨om、Said Taghadouini、Alexis Gallagher、Raja Biswas、Faisal Ladhak、Tom Aarsen、Griffin Thomas Adams、Jeremy Howard 和 Iacopo Poli。更智能、更优秀、更快速、更持久：用于快速、高效内存及长上下文微调与推理的现代双向编码器。载于《第63届计算语言学协会年会论文集》（第1卷：长篇论文），第2526–2547页，2025年。4[27] Ximing Xing、Juncheng Hu、Ziteng Xue、Jing Zhang、Buyu Li、Sheng Wang、Dong Xu 和 Qian Yu。SVGFusion：用于矢量图形生成的VAE扩散Transformer。arXiv预印本 arXiv:2412.10437，2024年。2、3[28] Yiying Yang、Wei Cheng、Sijin Chen、Xianfang Zeng、Fukun Yin、Jiaxu Zhang、Liao Wang、Gang Yu、Xingjun Ma 和 Yu-Gang Jiang。OmniSVG：统一的可扩展矢量图形生成模型。载于《神经信息处理系统进展》，第113670–113696页，2025年。2、3[29] Zhenhua Yang、Dezhi Peng、Yuxin Kong、Yuyi Zhang、Cong Yao 和 Lianwen Jin。FontDiffuser：通过结合多尺度内容聚合与风格对比学习的去噪扩散实现单样本字体生成。《AAAI人工智能会议论文集》，38(7)：6603–6611，2024年。2、3、6[30] Qihang Yu、Mark Weber、Xueqing Deng、Xiaohui Shen、Daniel Cremers 和 Liang-Chieh Chen。一幅图像等于重建与生成所需的32个标记。载于《神经信息处理系统进展》，第128940–128966页，2024年。4[31] Hao Yuan、Yuxuan Luo、Xing Chen 和 Zhouhui Lian。VecFontLLM：锚点引导的中文矢量字体直接合成。arXiv预印本 arXiv:2607.17251，2026年。2、3[32] Xiaohua Zhai、Alexander Kolesnikov、Neil Houlsby 和 Lucas Beyer。扩展视觉Transformer。载于《IEEE/CVF计算机视觉与模式识别会议论文集》，第12104–12113页，2022年。6[33] Richard Zhang、Phillip Isola、Alexei A. Efros、Eli Shechtman 和 Oliver Wang。深层特征作为感知度量的非凡有效性。载于《*IEEE*计算机视觉与模式识别会议，第 586–595 页，2018 年。2，6，14[34] Zhongyin Zhao、Ye Chen、Zhangli Hu、Xuanhong Chen 和 Bingbing Ni。通过相互脉冲双域扩散实现矢量图形生成。载于*IEEE/CVF*计算机视觉与模式识别会议论文集，第 4420–4428 页，2024年。2，3

FlowVecFont：通过潜在流匹配实现可缩放矢量字体生成

补充材料

A. 数据集统计

图S1 展示了本研究所用完整 Google Fonts 字体集合中的字符覆盖率，统计范围为预处理或序列长度筛选之前。字符和字体标签的样本数量均呈长尾分布。最低码点处的密集区域包括 ASCII（基本拉丁文）和拉丁文扩展，这些字符受到许多字体的支持。其他集中区域对应更多拉丁文字符和常用符号。较高码点处的宽广带状区域对应中日韩统一表意文字和谚文音节，而支持这两类字符的字体相对较少。

图 S2展示了同一集合中的原始轮廓序列长度，其中可变字体按默认轴设置测量。序列的均值和中位数分别为 93.8 和 62 个元素，其中包括结束标记。命令序列长度上限$L = 256$ 涵盖了 95.9% 的所有字形。超过此上限的字形数量最多的字体家族是 Shippori MinchoB1、Chiron Sung HK 和 Bpmf Zihi KaiStd。Rubik Pixels、Handjet 和 Rubik 80s Fade 等装饰字体的字形序列也特别长。

B. 预处理

VAE、CLS、STR 和 REF 使用通用的预处理框架，并根据每个模型和条件方案调整变换及其强度。我们以 REF 作为代表性示例，因为它同时使用了风格参考和字符参考。

图 S3 展示了条件 REF 训练

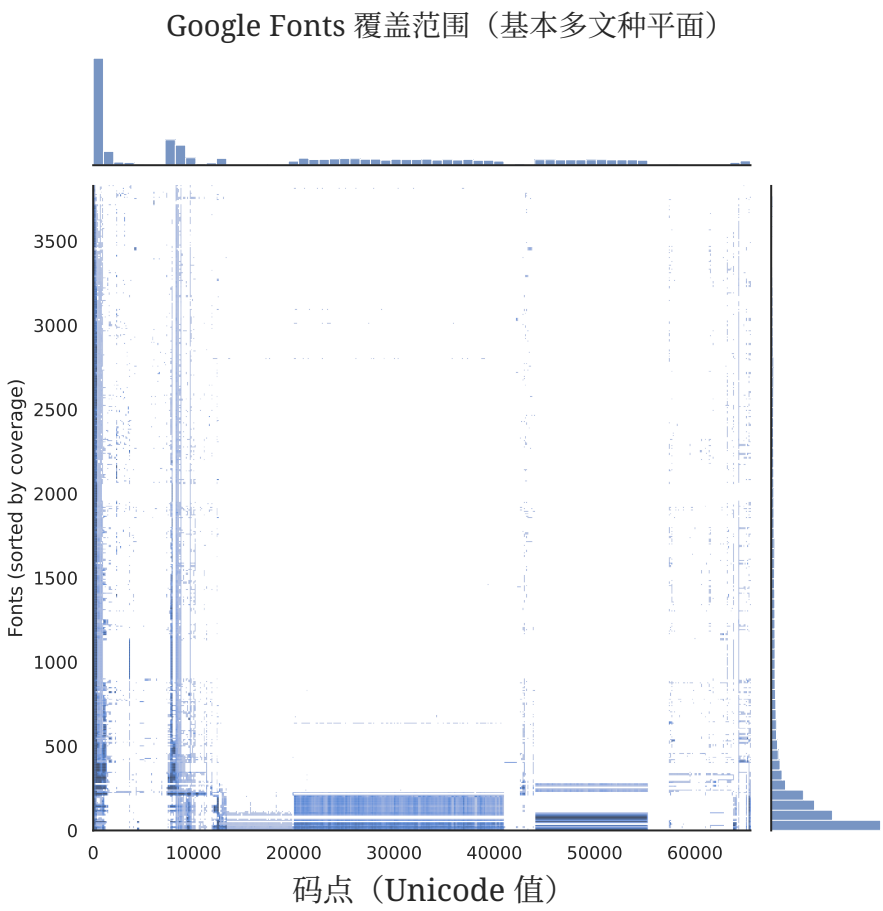


图 S1。Google Fonts 在基本多文种平面（BMP）中的字符覆盖率。彩色分箱表示观测到的字体字形与码位配对，其中较深的蓝色表示每个分箱中的配对数量更多。

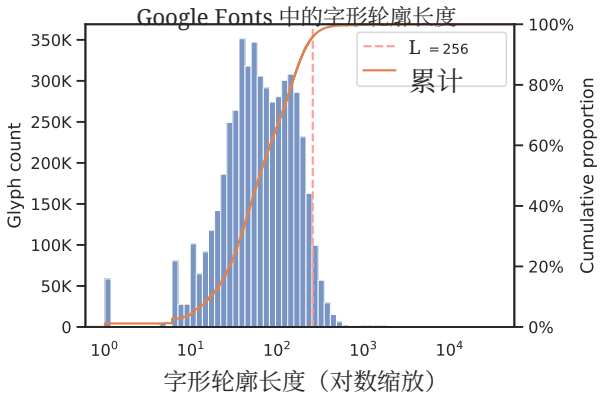


图 S2. 原始轮廓序列长度（包括结束标记），右侧轴显示累计比例。虚线标示 $L = 256$，其覆盖了 95.9% 的所有字形。

流程包含一个目标、四个风格参考和四个字符参考。对于可变字体，我们通过随机采样轴设置来改变字形外观；目标和风格参考共享这些设置，而每个字符参考则独立采样。随后，我们将二次曲线转换为三次贝塞尔曲线，并合并兼容的相邻线段，以获得紧凑且一致的轮廓表示。

我们对每个风格参考分别应用子路径丢弃。随后，对所有字形应用重叠处理、随机缩放、仿射变换和弹性变形。对于重叠处理，我们以相同概率选择保持轮廓不变、移除随机选定的轮廓组内的重叠，或移除所有重叠。目标和风格参考共用这些操作的应用决策和变换参数，而每个字符参考则独立处理。

其余随机变换对每个字形独立采样。我们应用线段拆分和高斯坐标噪声，并对字符参考施加更强的扰动。我们对目标和风格参考的绕行、子路径起始点和子路径顺序进行规范化。对于字符参考，这些操作中的每一项都会独立选择规范化或随机化。最后，我们从超过 $L = 256$ 的序列中随机移除完整子路径，直至序列符合长度限制。

评估时，所有四个模型的预处理都是确定性的。对于可变字体，我们使用默认轴设置，并采用相同的曲线转换和合并步骤。我们对绕行、子路径起点和子路径顺序进行规范化，然后在序列超出长度限制时，于子路径边界处截断序列。



图 S3. 条件 REF 训练的预处理流程。方框覆盖接受各项操作的组。

C. 消融研究

C.1. 坐标重建损失

我们在 400K 个训练步数内，从训练稳定性和测试重建质量两方面比较 L1误差和均方误差（MSE）坐标重建损失。L1误差和 MSE 的坐标损失权重分别为 3 和 100。这些权重针对每种目标分别通过初步实验进行调优。我们使用指数移动平均（EMA）权重计算验证损失。

图S4 显示，L1验证损失在大部分训练过程中平稳下降。均方误差验证损失

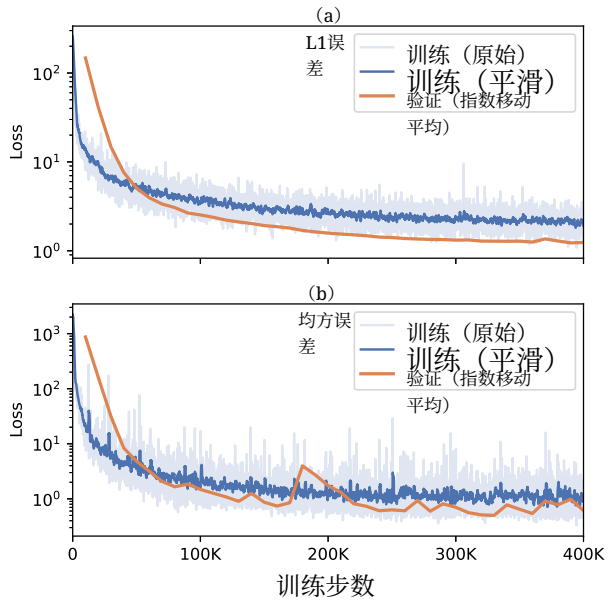


图 S4。L1 和均方误差变分自编码器变体的训练损失和验证损失。浅蓝色线条表示指数移动平均之前的原始训练损失，实蓝色线条表示跨越 1K 训练步数的尾随移动平均。橙色线条表示使用 EMA 权重每 10K 步计算一次的验证损失，未进行平滑处理。由于目标函数及其权重不同，绝对损失值不可直接比较。

呈现反复的尖峰，尤其是在约 180K 步数时。即使接近训练结束时，它仍不稳定。即使训练损失恢复，指数移动平均（EMA）仍会保留不稳定训练更新的影响。因此，增加训练步数并不能持续降低均方误差验证损失。

坐标异常大的罕见字形会导致这种不稳定性。图S5展示了两个示例：Ubuntu Regular 字标（U+F200）和 MonteCarlo 装饰性花饰（U+E16A）。在数据增强之前，它们的最大绝对坐标分别为 3.480 和 3.409 em，与 Ubuntu Regular 字母 A 的 0.693 em 相比明显更大。我们的输入坐标以 em 单位表示，并未针对每个字形的边界框进行归一化。因此，这些异常宽大的轮廓仍保留较大的坐标值。均方误差会对重建误差施加二次惩罚，使训练对这些字形较为敏感。L1误差采用线性惩罚，因此对它们不那么敏感。

表 S1显示，在 SFSC 上，L1误差的倒角距离、LPIPS、FID 和评估失败率也均低于均方误差。几何指标和基于图像的指标均有所提升，这支持在坐标重建中使用 L1误差，同时其验证轨迹也更加稳定。

Font	码点	轮廓
Ubuntu Regular	U+0041	
Ubuntu Regular	U+F200	
MonteCarlo	U+E16A	

图 S5。损失尖峰附近发现的大坐标训练字形。字母 A 提供缩放参考。

表 S1。使用 L1误差或均方误差坐标损失进行 400K 变分自编码器训练步数后，在 SFSC 上的测试重建质量。失败率表示失败率（%）。

Loss	CD ↓	LPIPS ↓	FID ↓	失败率 ↓
L1	**0.0029**	**0.0045**	**0.09**	**0.00**
MSE	0.0039	0.0093	0.30	0.01

表 S2。在经过 400K 训练步数后，SFSC 上的变分自编码器数据增强。Fail. 表示失败率（%）。

数据增强	CD ↓	LPIPS ↓	FID ↓	失败率 ↓
None	**0.0028**	**0.0039**	**0.05**	0.023
不使用轴采样	0.0031	0.0053	0.11	0.002
Full	0.0029	0.0045	0.09	**0.000**

C.2. 变分自编码器数据增强

我们比较了三种训练设置：不进行数据增强（无）；使用固定可变字体轴进行数据增强（不进行轴采样）；使用随机轴采样进行完整数据增强（完整）。评估时不进行数据增强。

表 S2 报告了其测试重建质量。不使用数据增强进行训练时，倒角距离、LPIPS 和 FID 最低。数据增强减少了评估失败，使用完整流程时未观察到评估失败。这表明，数据增强提高了重建的稳健性。在使用数据增强的训练设置中，与固定轴相比，随机轴采样同样改善了倒角距离、LPIPS 和 FID。

基于离线预处理的流程（例如 DeepVecFont-v2 [24] 和 DesigNet [7], ）使用提取的字形表示，而不包含字体的变体数据。这限制了

表 S3. 潜在令牌数量对 FVF-VAE 重建和 FVF-CLS/B 在 SFSC 上生成效果的影响。在此消融实验中，FVF-CLS/B 训练了 400K 步，而非主要实验中使用的 1M 步。Fail. 表示失败率（%）。

模型	K	CD ↓	LPIPS ↓	FID ↓	失败率 ↓
FVF-VAE	16	0.0041	0.0166	1.38	0.121
	32	0.0029	0.0045	0.09	**0.000**
	64	**0.0025**	**0.0030**	**0.04**	0.002
FVF-CLS/B	16	0.0191	0.1168	26.79	2.21
	32	0.0183	0.1018	16.40	1.71
	64	**0.0171**	**0.0861**	**7.22**	**0.44**

训练只能在有限的轴位置集合上进行。我们的库 TorchFont 可在运行时直接从字体文件加载轮廓。这使得训练期间能够从连续空间中随机采样新的轴位置。轴采样带来的改进证明了这一能力的优势。

C.3. 潜在令牌数量

我们比较了使用 16、32 和 64 个潜在令牌进行变分自编码器重建和字体生成的效果。每个变分自编码器及其对应的 FVF-CLS/B 模型均训练 400K 步。

表 S3 报告了在 SFSC 上的重建和生成质量。使用 64 个令牌时，两个模型的倒角距离、LPIPS 和 FID 均最低。变分自编码器重建在令牌数从 16 增至 32 时显著改善。相比之下，将令牌数从 32 增至 64，仅带来少量额外提升。FVF-CLS/B 在 64 个令牌时仍有所提升，但更长的潜在序列也会增加每个训练步数的计算成本。由于所有模型的训练步数相同，这些提升可能部分归因于额外的计算量。我们采用 32 个令牌，因为它们能提供足够的重建质量，同时所需计算量低于 64 个令牌。

C.4. 引导强度与欧拉步数

我们使用同一个训练了 1M 步的 CLS/B 检查点，评估无分类器引导（CFG）强度以及欧拉步数对 SFSC 的影响。我们比较了采用 32 个欧拉步数时的 CFG 缩放值 $\{1.0, 1.5, 2.0, 3.0, 4.0, 5.0\}$，以及 CFG 缩放值为 3.0 时的 $\{8, 16, 32, 64\}$ 个欧拉步数。

表 S4 显示，将 CFG 缩放从 1.0 增至 3.0 可改善 CD、LPIPS 和 FID。然而，将缩放增至 4.0 或 5.0 会使这三项指标均变差。因此，在我们比较的缩放值中，3.0 能带来最佳生成质量。

表 S5 显示，在我们比较的各个步数中，32 个欧拉步数可获得最佳生成质量。将步数增加到 64 会使倒角距离、LPIPS 和 FID 略微变差，但失败率也略有降低。根据这些结果，我们在推理时采用 CFG 缩放 3.0 和 32 个欧拉步数

表 S4. 使用 32 个欧拉步数时，SFSC 上 1M 步 CLS/B 检查点的引导强度消融实验。Fail. 表示失败率（%）。

CFG 缩放	CD ↓	LPIPS ↓	FID ↓	失败率 ↓
1.0	0.0207	0.1146	18.26	1.26
1.5	0.0167	0.0919	13.07	1.16
2.0	0.0159	0.0869	12.02	**1.14**
3.0	**0.0157**	**0.0853**	**11.87**	1.21
4.0	0.0158	0.0857	12.17	1.20
5.0	0.0159	0.0864	12.51	1.20

表 S5。SFSC 上 1M 步 CLS/B 检查点的欧拉步数消融实验。所有变体均使用 CFG 缩放 3.0。失败率表示失败率（%）。四舍五入前的最佳值以粗体显示。

欧拉步数	CD ↓	LPIPS ↓	FID ↓	失败率 ↓
8	0.0163	0.0933	20.63	1.79
16	0.0157	0.0859	13.01	1.33
32	**0.0157**	**0.0853**	**11.87**	1.21
64	0.0157	0.0856	12.05	**1.18**

。

D. 更多结果

表 S6 展示了不同模型尺寸和测试划分下完整的重建与生成结果。除对称倒角距离（CD） [3], 学习感知图像块相似度（LPIPS） [33], Fr´echet Inception 距离（FID） [8], 和评估失败率外，我们还报告平均绝对像素误差（L1）和结构相似性指数（SSIM） [25]。

变分自编码器在全部四个划分上都保持较低的重建误差。对于生成任务，在每种评估的条件策略和划分下，L1误差均随模型尺寸增大而降低，结构相似性指数则随之升高，这与主论文中的缩放趋势一致。

图 S6将训练计算量分析扩展至 SFSC 上的全部六项指标，使用每 100K 个训练步数保存的检查点。

图 S7、S8、S9和S10分别展示了 SFSC、SFUC、UFSC 和 UFUC 上的 VAE 重建字形与生成字形，涵盖了所有评估过的条件策略和模型尺寸。每幅图均包含对应划分中保存的全部 16 个示例。真实轮廓以蓝色显示，重建或生成的轮廓以橙色显示，便于直接比较轮廓对齐情况和局部伪影。

表S7 在 74 种字体面中的 3,170 个 UFSC 字形上，将 L1误差和结构相似性指数纳入与现有方法的比较。这些指标给出的排名与倒角距离、LPIPS 和 FID 相同：FVF-STR/XL 优于 DeepVecFont-v2 和 De-

sigNet，而 VecGlypher 的得分最佳。与主论文中的做法相同，本次比较中每个字形都使用其自身的边界框进行归一化，因此这些得分不能与表S6中的得分直接比较。

图S11 补充展示了同一评估中的全部 16 个保存示例，以完善这一量化比较。

表 S6。VAE 重建和字体生成的完整测试结果。CD 表示倒角距离。规模表示总参数数量，包括共享冻结 VAE。失败率表示评估失败率（%）。质量指标基于成功评估的字形计算。

模型	Size	划分	CD ↓	L1 ↓	结构相似性指数 ↑	LPIPS ↓	FID ↓	失败率 ↓
FVF-VAE	14M	SFSC	0.0029	0.0031	0.9947	0.0045	0.09	0.00
		SFUC	0.0029	0.0031	0.9947	0.0045	0.07	0.00
		UFSC	0.0031	0.0034	0.9945	0.0047	0.06	0.00
		UFUC	0.0031	0.0034	0.9945	0.0047	0.16	0.00
FVF-CLS/S	90M	SFSC	0.0194	0.0539	0.8422	0.1135	23.11	2.12
FVF-CLS/B	230M	SFSC	0.0157	0.0431	0.8677	0.0853	11.86	1.17
FVF-CLS/L	586M	SFSC	0.0136	0.0358	0.8863	0.0685	6.51	0.74
FVF-CLS/XL	817M	SFSC	0.0131	0.0347	0.8894	0.0657	5.86	0.55
FVF-STR/S	88M	SFSC	0.0279	0.0688	0.8091	0.1554	35.09	3.11
		UFSC	0.0285	0.0734	0.8000	0.1628	40.43	3.85
FVF-STR/B	227M	SFSC	0.0267	0.0661	0.8140	0.1448	28.20	1.89
		UFSC	0.0277	0.0714	0.8031	0.1546	32.48	2.20
FVF-STR/L	582M	SFSC	0.0209	0.0479	0.8528	0.1016	11.53	1.43
		UFSC	0.0217	0.0511	0.8458	0.1066	11.68	1.89
FVF-STR/XL	813M	SFSC	0.0204	0.0469	0.8557	0.0991	10.62	1.18
		UFSC	0.0212	0.0502	0.8480	0.1036	10.72	1.50
FVF-REF/S	47M	SFSC	0.0292	0.0697	0.8091	0.1611	37.03	3.80
		SFUC	0.0286	0.0708	0.8065	0.1627	38.98	3.66
		UFSC	0.0294	0.0728	0.8042	0.1663	43.63	4.32
		UFUC	0.0286	0.0739	0.8010	0.1678	48.22	4.19
FVF-REF/B	144M	SFSC	0.0268	0.0625	0.8219	0.1414	26.87	2.67
		SFUC	0.0264	0.0634	0.8196	0.1428	28.22	2.66
		UFSC	0.0273	0.0661	0.8155	0.1473	32.08	3.04
		UFUC	0.0268	0.0669	0.8128	0.1485	35.10	3.39
FVF-REF/L	471M	SFSC	0.0238	0.0529	0.8416	0.1166	14.80	1.94
		SFUC	0.0236	0.0536	0.8394	0.1178	15.34	2.07
		UFSC	0.0244	0.0560	0.8353	0.1216	16.59	2.34
		UFUC	0.0240	0.0572	0.8319	0.1233	18.86	2.28
FVF-REF/XL	688M	SFSC	0.0232	0.0516	0.8444	0.1123	12.72	1.71
		SFUC	0.0232	0.0526	0.8418	0.1140	13.11	1.79
		UFSC	0.0238	0.0547	0.8384	0.1172	13.84	2.04
		UFUC	0.0235	0.0558	0.8351	0.1186	16.28	2.11

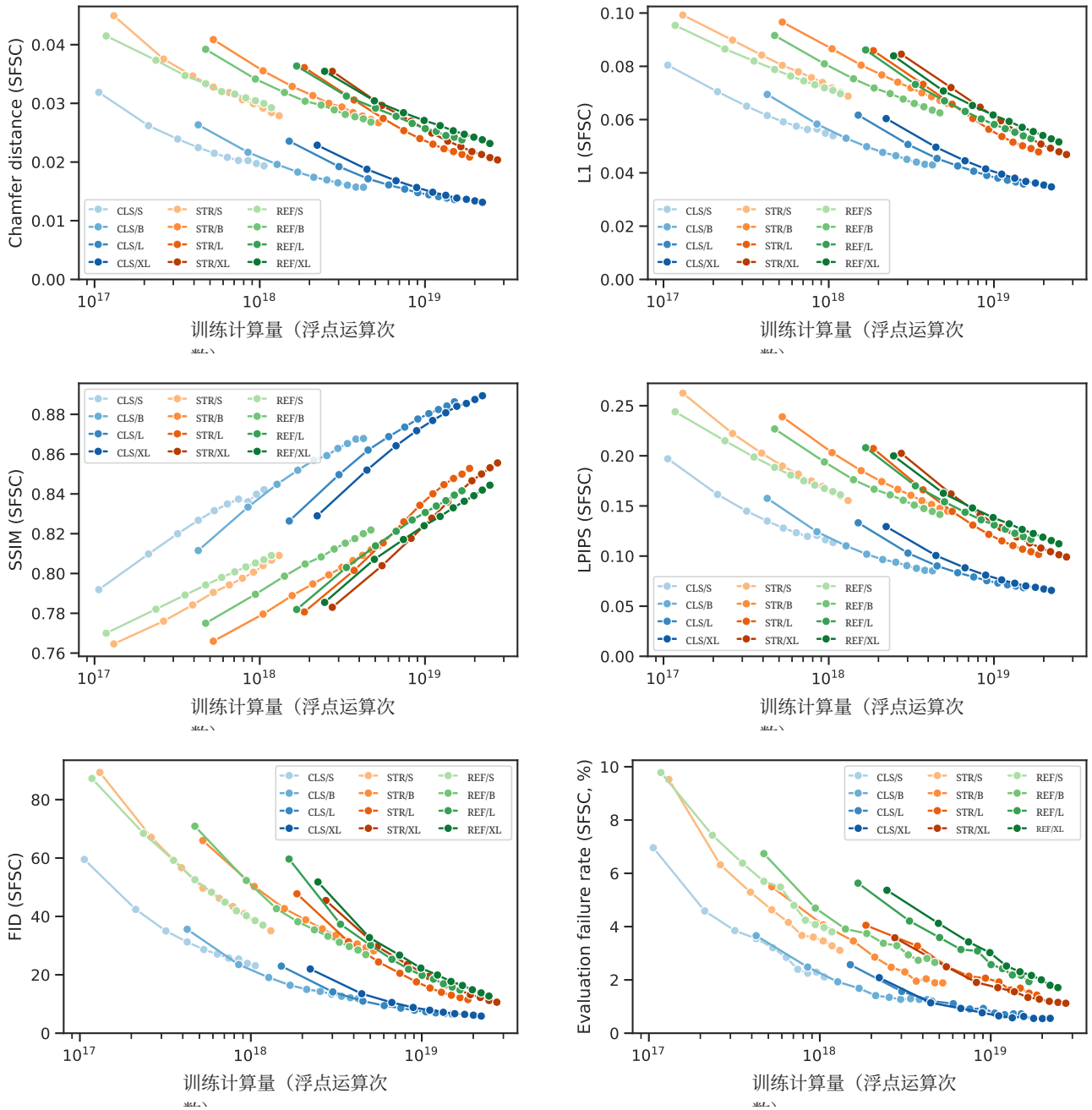


图 S6. SFSC 上的倒角距离、L1误差、结构相似性指数、LPIPS、FID 和评估失败率与训练计算量的关系。每个点对应一个每 100K 步保存的检查点。

模型	U+88C5	U+1DF18	U+81A0	U+715C	U+00CB	U+219F	U+0415	U+C038	U+2A91	U+2569	U+00A4	U+02C7	U+5E73	U+1E08	U+55B6	U+1EF5
FVF-VAE																
FVF-CLS/S																
FVF-CLS/B																
FVF-CLS/L																
FVF-CLS/XL																
FVF-STR/S																
FVF-STR/B																
FVF-STR/L																
FVF-STR/XL																
FVF-REF/S																
FVF-REF/B																
FVF-REF/L																
FVF-REF/XL																

图 S7。SFSC 上的定性结果。真实字形（蓝色）与变分自编码器重建字形或生成字形（橙色）叠加显示。每列展示同一个目标字形，并标注其 Unicode 码点；样本并非经过挑选。

模型	U+207E	U+CE49	U+5099	U+1EB7	U+53F6	U+C185	U+25CA	U+5094	U+00DD	U+B294	U+304E	U+6AAC	U+CCF7	U+00DD	U+B871	U+1E51
FVF-VAE																
FVF-REF/S																
FVF-REF/B																
FVF-REF/L																
FVF-REF/XL																

图 S8. SFUC 上的定性结果。真实字形（蓝色）与变分自编码器重建字形或生成字形（橙色）叠加显示。每列展示相同的目标字形，并标注其 Unicode 码点；样本并非经过筛选的。

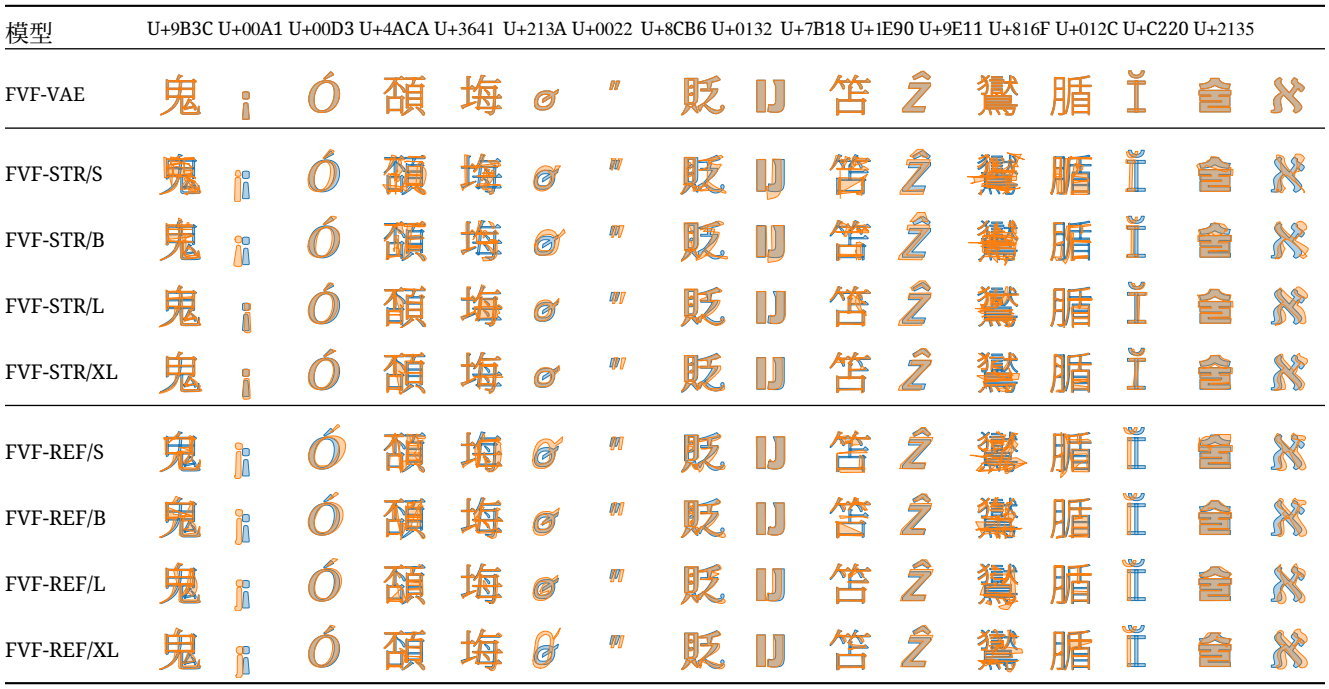


图 S9. UFSC 上的定性结果。真实字形（蓝色）与变分自编码器重建字形或生成字形（橙色）叠加显示。每列展示同一个目标字形，并标注其 Unicode 码点；样本并非经过挑选。

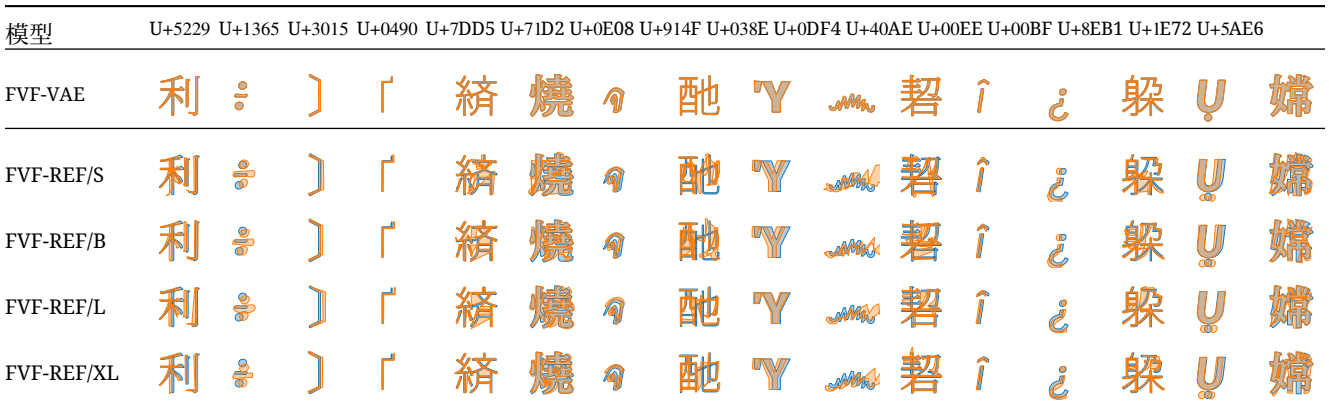


图 S10。UFUC 上的定性结果。真实字形（蓝色）与变分自编码器重建字形或生成字形（橙色）叠加显示。每一列展示相同的目标字形，并标注其 Unicode 码点；样本并非经过精挑细选。

表 S7。来自 74 个 Google Fonts 字体面的 3,170 个 UFSC 字形的完整对比结果。CD 表示倒角距离。质量指标是在使用每个字形自身的边界框进行归一化后计算的。规模表示参数数量，包括推理时使用的冻结组件。Fail. 表示评估失败率（%）。质量指标不包括失败样本。FVF-STR/XL 是我们提出的方法。最佳结果以粗体显示。

方法	Size	CD ↓	L1 ↓	结构相似性指数 ↑	LPIPS ↓	FID ↓	失败率 ↓
DVF-v2	121M	0.0211	0.0754	0.7793	0.1627	24.99	0.41
DesigNet	14M	0.0269	0.0989	0.7448	0.2146	50.03	0.63
VecGlypher	27B	**0.0129**	**0.0464**	**0.8502**	**0.0849**	**3.27**	**0.00**
FVF-STR/XL	813M	0.0134	0.0482	0.8353	0.0917	4.86	**0.00**

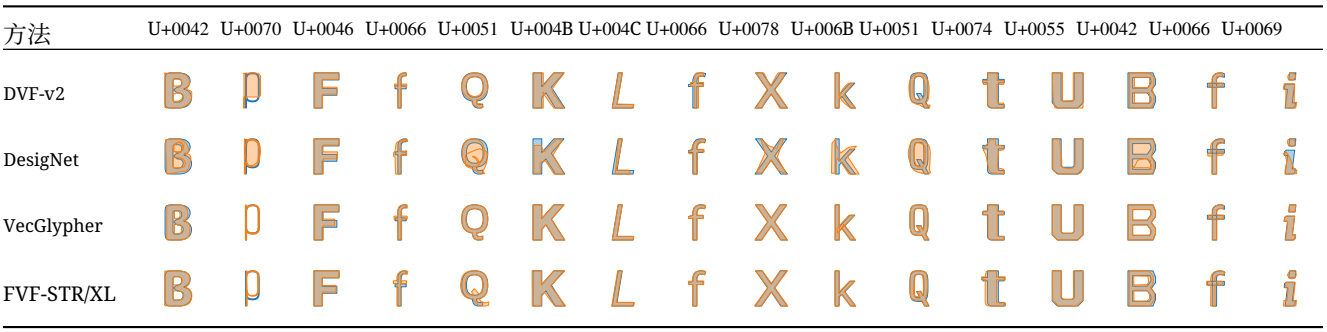


图 S11. 在 UFSC 上与现有方法的完整定性比较。FVF-STR/XL 是我们提出的方法。真实字形（蓝色）与生成字形（橙色）叠加显示。每列展示相同的目标字形，并标注其 Unicode 码点。所有 16 个保存的示例均按字体 ID 和码点升序排列。