LayoutFlow: Flow Matching for Layout Generation

Julian Jorge Andrade Guerreiro[1], Naoto Inoue[2], Kento Masui[2], Mayu Otani[2], and Hideki Nakayama[1]

[1] The University of Tokyo, Japan
{guerreiro,nakayama}@nlab.ci.i.u-tokyo.ac.jp
[2] CyberAgent, Japan
{inoue_naoto,masui_kento,otani_mayu}@cyberagent.co.jp

Abstract. Finding a suitable layout represents a crucial task for diverse applications in graphic design. Motivated by simpler and smoother sampling trajectories, we explore the use of Flow Matching as an alternative to current diffusion-based layout generation models. Specifically, we propose LayoutFlow, an efficient flow-based model capable of generating high-quality layouts. Instead of progressively denoising the elements of a noisy layout, our method learns to gradually move, or flow, the elements of an initial sample until it reaches its final prediction. In addition, we employ a conditioning scheme that allows us to handle various generation tasks with varying degrees of conditioning with a single model. Empirically, LayoutFlow performs on par with state-of-the-art models while being significantly faster. The project page, including our code, can be found at `https://julianguerreiro.github.io/layoutflow/`.

Keywords: Layout Generation · Flow Matching · Generative Models

1 Introduction

Layout design describes the process of arranging various elements, such as images, text, or other components, on a page or screen. Finding an appropriate layout forms a crucial part of creating documents, user interfaces, graphic designs, and other compositions since the arrangement of different elements can substantially impact how the intended message or purpose is communicated. Over the years, several approaches have been explored to automate layout generation in a data-driven manner using machine learning methods. Nonetheless, current layout generation models still leave room for improvement in terms of layout quality and sampling speed.

Motivated by this observation, we propose LayoutFlow, a generative framework based on Flow Matching [1,32,33,44] that can produce high-quality layouts while requiring less time than previous methods. Flow Matching has recently been introduced as a powerful generative framework and demonstrated strong performance on various tasks, including image generation [34]. Intuitively, flow-based models try to learn a *flow* that moves samples from a base distribution to
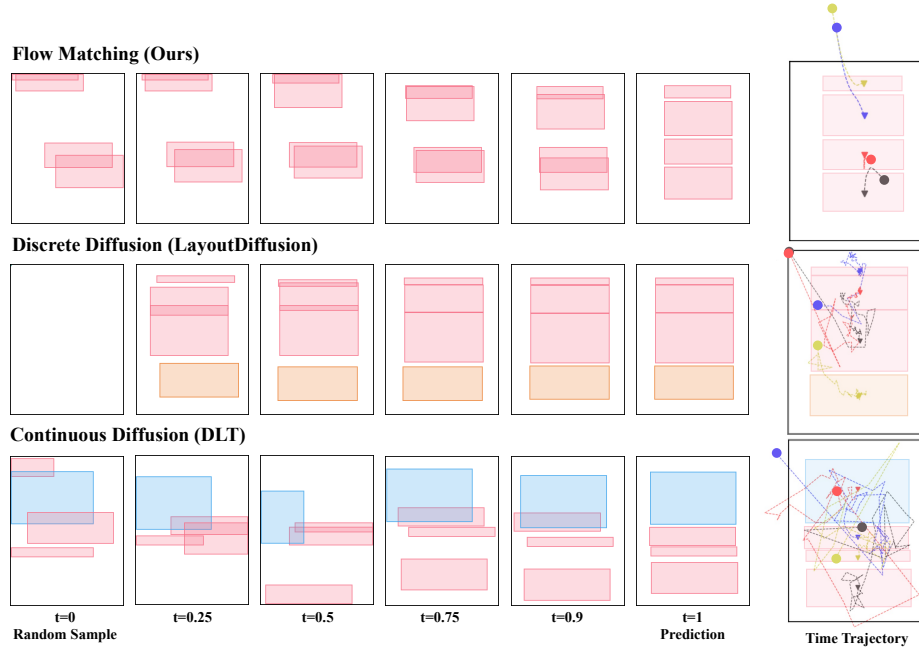
Fig. 1: Comparison of different layout generation trajectories. Given a randomly initialized layout at time $t = 0$ with fixed element sizes, we visualize different states of the generation process until the final layout at $t = 1$. In addition, we overlay the trajectory, which can be interpreted as the movement of the elements over time, on top of the final layout. A circle marks the location of the initial sample and a triangle marks the final location. Flow Matching produces smooth and directed paths, whereas both diffusion models slowly converge to the final prediction under noisy trajectories with a long path length. As a result, flow-based models require fewer evaluation steps than diffusion, leading to faster sampling.

a target distribution defined by the training data. In this work, we investigate how to apply Flow Matching for layout generation and show that flows offer a more intuitive generation process from a geometrical interpretation compared to previous models.

Most recent layout generation models [4, 5, 8, 15, 20, 22, 28, 48] have been based on the diffusion framework [18] which has become the standard for various generative tasks [39]. While diffusion models are commonly described as models that learn to gradually remove random noise that was added to the training data, the generation process can also be interpreted in analogy to flow-based models as moving samples from a Gaussian base distribution to a data distribution. In the diffusion case, however, additional randomness is introduced along the way by adding noise. Mathematically, this corresponds to a Stochastic Differential Equation (SDE), whereas a flow is defined by a simpler Ordinary Differential Equation (ODE), which we describe in more detail in Sec. 3.

In the context of layout generation, the differences between diffusion-based and flow-based models become apparent when visualizing the trajectories created by the generation process, as illustrated in Fig. 1. The trajectories show how the sample from the base distribution is moved by the flow or backward diffusion process until it reaches the final layout prediction. While our LayoutFlow model produces smooth trajectories with short path lengths, diffusion-based models follow a noisy trajectory that constantly changes directions due to the added noise. As a result, diffusion models tend to require more sampling steps, *i.e.*, longer inference time, than flow-based models. Altogether, we argue that learning how to move the initial sample straight toward the final prediction is much more natural than trying to do so under additional noise. Overall, our contributions can be summarized as follows:

1. Motivated by its geometrical interpretation, we apply Flow Matching to the task of layout generation and propose a novel model called LayoutFlow, which can handle various tasks utilizing a conditioning mechanism.
2. On top of generating simpler trajectories, flow-based models generally also offer more choices than diffusion models, such as different prior distributions or training trajectories. We, therefore, extensively explore this additional flexibility offered by Flow Matching in the context of layout generation.
3. Empirically, we demonstrate that our proposed flow-based model significantly outperforms previous diffusion-based layout generation models of similar size and performs on par compared to a significantly larger model. In either case, our model greatly speeds up inference, requiring only a fraction of the time previous models need to generate samples.

2 Related Work

2.1 Layout Generation

Over the years, layout generation has been investigated using various methods. Early works started exploring generating layouts by minimizing an energy function based on pre-defined constraints [36, 37]. With the advent of generative machine learning techniques, however, research shifted to a more data-centric learning approach and started to use Variational Autoencoders (VAE) [2, 13, 27, 47], and Generative Adversarial Networks (GAN) [25, 29, 30]. While earlier works explored a variety of different architectures, such as Graph Convolutional Networks [27], Long-Short Term Memories (LSTM) [13, 24] or Transformers [14, 45, 47], most recent research has converged to using a Transformer-based architecture due to its flexibility and strong performance.

Diffusion-based Models. As diffusion models have proven to generate more diverse data and are generally easy to train, especially compared to GANs, most recent research has shifted towards exploring the diffusion process for layout generation tasks. Since layouts are represented by categorical data defining types of elements and continuous numerical values describing the element position and

size, applying standard diffusion models is not straightforward. As a result, different approaches have incorporated a diffusion loss into their training process, which can be divided into continuous [18] and discrete diffusion models [3]. For the discrete case, the continuous coordinates are quantized and interpreted as different states along with the categorical element type [15, 20, 22, 48]. In this discrete scenario, the forward diffusion process is modeled as a random walk between discrete states that, in addition to the quantized states, also include a mask state, which removes elements from the canvas. While LayoutDM [22] and LDGM [20] rely on the mask-and-replace strategy proposed in [12], LayoutDiffusion [48] introduces a new mild forward process that is closer to the continuous process while still increasingly masking elements over time.

On the other hand, continuous diffusion models have been less common as they have not attained the same performance as discrete models. In particular, continuous models have struggled to produce layouts with well-aligned elements. The first continuous layout generation model based on diffusion was proposed by Chai *et al.* [4], but is restricted to generating continuous element coordinates given categorical type data as a condition. Cheng *et al.* [8] introduced a latent diffusion model for layout design focusing on user constraints by embedding the layout into continuous representation, on which the diffusion process is performed. The first continuous diffusion-based layout generation model acting on the layout representation and able to handle unconditional generation was proposed by Chen *et al.* [5]. Their work addresses the alignment issue observed for continuous diffusion by introducing a specific regularization loss. Lastly, Levi *et al.* [28] proposed DLT, which applies a continuous diffusion process on the element coordinates and a discrete process on the element types.

Large Language Models. Motivated by the strong generalization capabilities of Large Language Models (LLM), the most recent works [11, 31, 43] have investigated the layout generation abilities of LLMs. While LLMs provide some interesting applications, such as zero-shot synthesis, they typically consist of billions of parameters, resulting in long inference times. Moreover, current models are limited to conditional layout generation. In contrast, our proposed approach aims to offer a lightweight model for unconditional and conditional layout generation.

2.2   Flow Matching and Diffusion Models

Flow-based models are based on learning a mapping between samples from a simpler distribution to a data distribution. Over the years, there have been various methods, such as Normalizing Flows or Continuous Normalizing Flows, which have used neural networks to estimate such a mapping, which can subsequently be used to sample from the learned distribution [26]. Until recently, the biggest issue with flow-based models has been the need to backpropagate through an ODE during training. However, recent models have proposed training algorithms that only require explicitly solving an ODE during inference [1, 32, 33, 35, 44] demonstrating impressive results and outperforming diffusion models on the image generation task [34]. Our proposed model builds on this improvement by

utilizing Flow Matching for layout generation while offering an improved geo-metrical interpretation.

Diffusion models, which were first proposed by Sohl-Dickenstein *et al.* [40] and later popularized by Ho *et al.* [18], can be formulated similarly to flow-based models. In fact, the term Flow Matching was inspired by the similar Score Matching loss [21] used by diffusion models. However, flows are characterized by an ODE, whereas the diffusion process can be formulated as an SDE with an additional stochastic component described by Brownian Motion [42]. As a result, diffusion models are restricted to a Gaussian as a base distribution, while flow-based models allow for more flexibility, which we also explore in our experiments. Notably, Song *et al.* [41] proposed DDIM, an alternative sampling method that makes it possible to sample using an ODE formulation on a model trained with an SDE forward process, effectively leading to smoother trajectories. In the context of layout generation, however, models have been using the SDE formulation following DDPM [18]. Our argument for introducing Flow Matching instead of just using DDIM for smoother trajectories lies in its increased flexibility and simplicity. Moreover, if the goal is to sample using an ODE, it seems more natural to also use an ODE during training instead of the more complex SDE used in diffusion.

3   Preliminary: Flow Matching

In this section, we first present the mathematical definition of a flow and then introduce how to train a neural network to estimate a flow via Flow Matching. The term *flow* in Flow Matching refers to a mapping between samples of two distributions. In general, flow-based models aim to estimate the flow between a known source probability distribution $p_0(x)$, *e.g.*, a Gaussian distribution, and the typically more complex data distribution $p_1(x)$. Given a data point $x \in \mathbb{R}^d$, the flow $\phi : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ can be defined by the following Ordinary Differential Equation (ODE)

$$\frac{d}{dt}\phi_t(x) = v_t(\phi_t(x)), \quad \phi_0(x) = x_0, \tag{1}$$

where the time-dependent vector field $v_t : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ is said to construct the flow [32] with the boundary condition $\phi_0(x) = x_0$. In other words, the flow $\phi_t(x)$ describes how an initial sample x_0 is transported over time and is given as the solution to the ODE in Eq. (1). As the flow transports samples with respect to time, the associated probability distribution is transformed according to the change of variables formula and creates a probability density path

$$p_t = [\phi_t]_* p_0 = p_0(\phi_t^{-1}(x)) \det \left[\frac{d\phi_t^{-1}}{dt}(x) \right] \tag{2}$$

with ϕ_t^{-1} denoting the inverse of the flow and $*$ representing the push-forward operator. If the flow constructed by the vector field v_t satisfies Eq. (2), it is

said that v_t generates the probability path p_t. This is the case if v_t follows the continuity equation, which holds for our considerations in this paper.

Instead of estimating the flow directly, Flow Matching trains a neural network $u_\theta(t, x)$ with weights θ to match the vector field v_t and then solves the ODE for a given boundary condition x_0 to obtain the flow. The Flow Matching loss function is defined as the expectation over uniformly sampled timesteps and samples along the probability density path as follows

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t \sim \mathcal{U}(0,1), x \sim p_t(x)} \|u_\theta(t, x) - v_t(x)\|^2. \tag{3}$$

However, this loss alone does not allow us to train the model as we usually do not have access to p_t nor v_t. As an extension, Lipman et $al.$ [32] have proposed a Conditional Flow Matching objective and is given as

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t \sim \mathcal{U}(0,1), z \sim q(z), x \sim p_t(x|z)} \|u_\theta(t, x) - v_t(x|z)\|^2, \tag{4}$$

with $v_t(x|z)$ denoting a conditional vector field generating the conditional probability path $p_t(x|z)$ and $q(z)$ being a distribution over a latent conditioning variable z. By introducing the conditional formulation of the Flow Matching objective, training a neural network becomes possible by choosing and training with a conditional vector field and its associated probability path. For an overview of the various choices and more details on Conditional Flow Matching, we recommend [32, 44].

4 Conditional Flow Matching for Layout Generation

4.1 Data Representation

A layout typically consists of various elements, such as text, tables, or images placed on a canvas. Mathematically, we can describe a layout $\mathcal{A}$ as a set of N elements $\mathbf{e}^k$ with $k \in [1, N]$ in the following way

$$\mathcal{A} = \{\mathbf{e}^k = (\mathbf{g}^k, \mathbf{a}^k) | k \in [1, N]\}, \tag{5}$$

where we split each element into its geometrical information $\mathbf{g}^k = (c_x^k, c_y^k, w^k, h^k)$ described by the bounding box center (c_x^k, c_y^k), width w^k and height h^k, and its category information $\mathbf{a}^k$ describing the element type, such as text or image. In our considerations, the elements on the canvas do not possess an inherent order, which is why the canvas is considered a set of elements. Layout generation can be divided into various tasks depending on the conditioning assumptions. For example, in the unconditional generation setting, it is assumed that no prior knowledge is available, while for the refinement task, a complete but coarse layout is given. Since Flow-Matching requires the input data to be continuous, we convert the categorical information $\mathbf{a}^k$ into a continuous embedding, denoted as $\tilde{\mathbf{a}}^k$, using B Analog Bits [7]. To learn the flow, we define a data sample $\mathbf{x} \in \mathbb{R}^{(4+B) \cdot N_{max}}$ as the concatenation of all the elements $\mathbf{g}^k$ and $\tilde{\mathbf{a}}^k$ in a single layout. If the number of elements is smaller than N_{max}, the maximum number of elements contained in a single layout across the entire dataset, we pad the remaining dimensions.
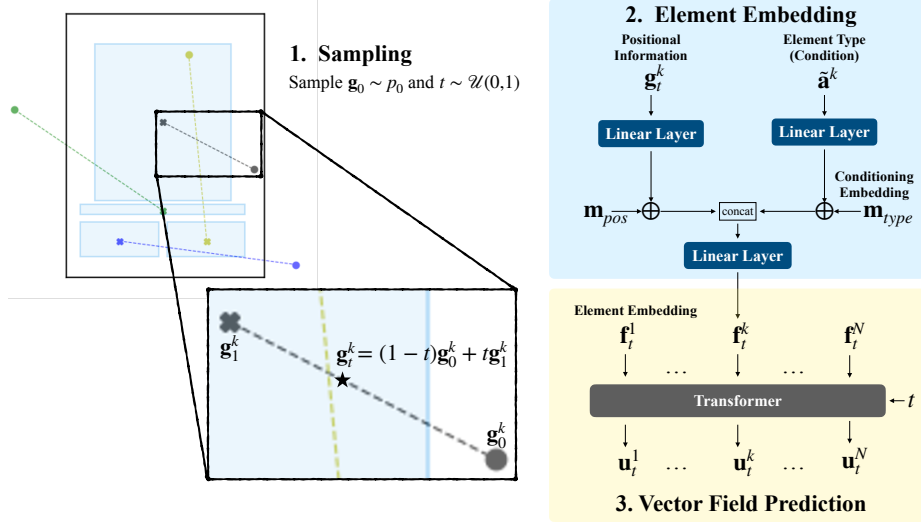

Algorithm 1: Training

$\mathbf{x}_0 \sim p_0(\mathbf{x})$, $\mathbf{x}_1 \sim p_{data}(\mathbf{x})$, $t \sim \mathcal{U}(0,1)$ // Sampling from distributions
$\mathbf{x}_t \leftarrow (1-t)\mathbf{x}_0 + t\mathbf{x}_1$ // Linear interpolation
$\mathbf{v}_t \leftarrow \mathbf{x}_1 - \mathbf{x}_0$ // Calculate the constant vector field
$\mathbf{u}_t \leftarrow f_\theta(\mathbf{x}_t, t)$ // Get vector field prediction from network
$\mathcal{L}(\theta) = \|\mathbf{u}_t - \mathbf{v}_t\|^2 + \lambda\|\mathbf{u}_\theta^{\mathbf{g}}(t, \mathbf{x_t}) - (\mathbf{g_1} - \mathbf{g_0})\|_1$ // Compute loss

---

Fig. 2: Overview of the training procedure of LayoutFlow for the type-conditioned scenario. First, we sample an initial layout from a base distribution and a time t. Then, an intermediate sample $\mathbf{g}_t$ is calculated by linearly interpolating between the initial sample and the ground truth layout. Each intermediate element is embedded jointly with the given element condition $\tilde{\mathbf{a}}^k$. Lastly, the Transformer architecture takes all the element embeddings to predict a vector field.

4.2 Training

To generate layouts, we want to find a way to sample from a data distribution $p_{\text{data}}(\mathbf{x})$, whereas we only have access to a limited amount of samples $\mathbf{x}$ through our training data. More specifically, our goal is to train a neural network that allows us to sample from the underlying data distribution and guide the sampling based on certain conditional constraints. To that end, we propose LayoutFlow, a layout generation model based on Flow Matching.

An overview of the training procedure for the type-conditioned scenario is illustrated in Fig. 2 and summarized in Algorithm 1 for the unconditional case. In the first step, we randomly sample a layout $\mathbf{x}_0$ from a prior distribution p_0, *e.g.*, a Gaussian distribution. This sample will typically not be well-aligned, and some elements might not be located on the canvas. In addition, we also randomly select a time t, which determines the intermediate sample $\mathbf{x}_t$ on the flow trajec-

tory used during training. For LayoutFlow, we train on a simple linear trajectory following [32,44]. Therefore, the intermediate sample is simply obtained by linearly interpolating between the ground truth layout $\mathbf{x}_1$ and the initial sample $\mathbf{x}_0$. The same trajectory is not used again during training since $\mathbf{x}_0$ and t change with every iteration.

The network f_θ is trained to output the conditional vector field, corresponding to the derivative of the trajectory as defined in Eq. (1). This derivative is simply a constant for the linear trajectory case, specifically the difference between $\mathbf{x}_0$ and $\mathbf{x}_1$. Intuitively, the network learns to output a direction pointing toward a data sample, which can be used during sampling to move the initial data samples toward a better prediction. While we train our network using linear trajectories, this does not mean that the model will also produce straight trajectories during sampling. As mentioned in Eq. (4), the linear trajectory, which is derived by the conditional vector field in Eq. (4), acts as a proxy to learn the actual vector field in Eq. (3) defined by Flow Matching.

In our experiments, we found that using only the Mean Squared Error (MSE) as a loss function tends to lead to poorly aligned layouts, which results in suboptimal perceptual quality. This observation is similar to the issue described in [5]. We hypothesize that this can be attributed to the fact that the MSE penalizes all mistakes within the same distance equally, regardless of direction, whereas the alignment between elements is usually only measured along the horizontal and vertical direction. As a result, the minimization objective does not fully align with the perceptual quality. To tackle this issue, we add an L_1 regularization on the geometrical part of the output $u_\theta^{\mathbf{g}}$ as the L_1 loss encourages sparsity of the error vector, which can also be interpreted as minimizing the error along the axis dimensions. Therefore, our final training loss can be written as

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathrm{CFM}}(\theta) + \lambda \mathcal{L}_1(\theta) = \mathcal{L}_{\mathrm{CFM}}(\theta) + \lambda \|u_\theta^{\mathbf{g}}(t, \mathbf{x_t}) - (\mathbf{g_1} - \mathbf{g_0})\|_1, \quad (6)$$

where λ is a hyperparameter.

4.3   Sampling

After training our neural network to predict a vector field, we can generate new layouts by sampling from our initial distribution and solving the ODE describing the flow as defined in Eq. (1). In practice, this can be done using any numerical ODE solver. Figure 3 illustrates the sampling process using the Euler method [10] to solve the ODE. For a fixed number of evaluation steps T, the initial sample x_0 is moved along the direction predicted by the network in an autoregressive manner. Therefore, the backward process can be denoted as
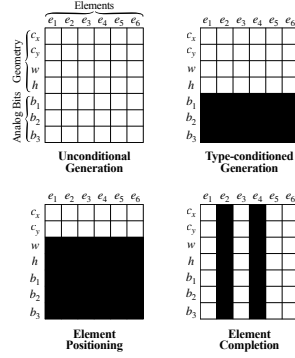
$$\mathbf{x}_{\frac{i+1}{T}} = \mathbf{x}_{\frac{i}{T}} + \frac{1}{T} u_\theta \left(\frac{i}{T}, \mathbf{x}_{\frac{i}{T}} \right), \quad (7)$$

with $i \in \mathbb{N}$ and $i \in [0, T-1]$ indicating how far along the sample has been moved.
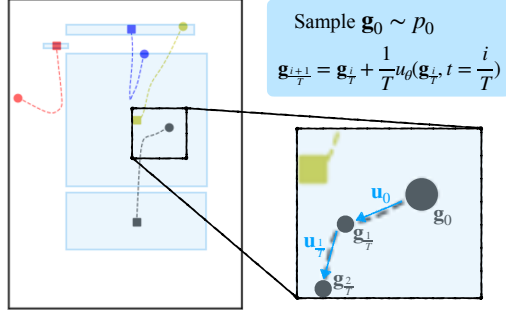
Fig. 3: Overview of the type-conditioned inference process. The initial sample is autoregressively moved in the predicted direction.

Fig. 4: Condition masks. Black indicates parts given as conditions.

4.4 Network Architecture

Here, we describe the actual implementation of the network f_θ to efficiently handle the layout. We first embed the elements of the sampled intermediate layout $\mathbf{x}_t$ and then use a Transformer to predict the conditional vector field as depicted in Fig. 2. First, we separately embed the geometrical information $\mathbf{g}_t^k$ and the element type $\tilde{\mathbf{a}}_t^k$ using a linear layer. A conditioning mask is embedded and added to the geometrical and type embedding to inform the network about which parts of the input act as a condition. The example in Fig. 2 illustrates this for the type-conditioned case, where the network is informed through the masks $\mathbf{m}_{type}$ and $\mathbf{m}_{pos}$ that $\tilde{\mathbf{a}}^k$ is taken from $\mathbf{x}_1$ while only the geometrical information $\mathbf{g}_t^k$ was taken from the training trajectory. In a subsequent step, we concatenate both embeddings and pass them through another linear layer to obtain a high-dimensional embedding $\mathbf{f}_t^k$ representing an element of the intermediate layout $\mathbf{x}_t$. Next, all element embeddings $\mathbf{f}_t^k$ are processed by a transformer encoder network together with the temporal information t incorporated in each layer. Finally, the output of the Transformer is projected back to the same dimension as the input layout using a linear layer representing the vector field.

4.5 Conditioning Mechanism

As layout generation involves various tasks with different assumptions about available knowledge, we employ a conditioning mechanism to perform these tasks with just a single model. Similarly to [5, 20, 28], we mark the conditioning input by including the information in the element embedding while distinguishing between the four different scenarios depicted in Fig. 4. During training, we simply randomly sample each mask with the same probability. Alternatively, it is also possible to only train an unconditional model and impose the conditions during sampling through a guidance strategy, similar to [22, 48]. However, we empiri-

cally found that including conditioning during training works better. For a more detailed analysis, we refer to the supplementary material.

5   Experiments

To compare LayoutFlow with other methods, we first perform a series of experiments on various layout generation tasks. In addition, we conduct an ablation study on our proposed method.

5.1   Experimental Settings

Datasets. We evaluate the performance of our network on the RICO [9] and PubLayNet [49] datasets, following previous methods. RICO consists of over 66k User Interface (UI) layouts with 25 element categories, while PubLayNet comprises over 360k document layouts annotated with 5 different element types. We train our network using the dataset split described in [23,48], which discards layouts containing more than 20 elements.

   Implementation Details. Similar to LayoutDM and DLT, we employ a 4-layer Transformer network with 8 attention heads and a model dimension of 512 for a fair comparison. The model is trained using the AdamW optimizer with a learning rate of 0.0005. In addition to the MSE Flow Matching loss, we regularize the training by calculating the L_1 loss between the geometrical information as described in Eq. (6) with $\lambda = 0.2$. For inference, we obtain the flow using a NeuralODE solver [6] for 100 steps with the Euler method.

   Evaluation Metrics. Following previous work [4, 5, 28, 48], we evaluate our method as well as comparable approaches using *Frechet Inception Distance (FID)* [16], *Maximum Intersection over Union (mIoU)* [25], *Alignment* [27], and *Overlap* [29]. *FID* and *mIoU* inherently evaluate both fidelity and diversity [17] of the generated results. For the *FID* calculation, we use the same network with identical weights as LayoutDiffusion [48] to measure the similarity between the generated layouts and the original dataset based on the feature space. The *mIoU* metric compares sets of layouts and measures their similarity by optimally matching generated and real layouts that maximize the average *IoU*. On the other hand, *Alignment* and *Overlap* capture the fidelity property. Nonetheless, both metrics should be judged with respect to a reference dataset, where the best outputs lead to similar values.

   Comparison with Existing Approaches. To validate the performance of our proposed method, we compare it to state-of-the-art techniques. Since there is no standard way of splitting the datasets, we need to retrain the models that do not follow the dataset split used in LayoutDiffusion [48] for a fair comparison. This fact limits us to only compare to methods that have made their code publicly available. Therefore, we compare our method against the two discrete diffusion models LayoutDM [4] and LayoutDiffusion [48] as well as the discrete-continuous diffusion model DLT [28]. Where applicable, we additionally compare

Table 1: Quantitative results for various layout generation tasks on the RICO and PubLayNet datasets. The two best results are highlighted in **bold** and <u>underlined</u>. The → symbol indicates best results are the ones closest to the validation data. Models marked with * have been retrained.

Task	Model	RICO				PubLayNet			
		FID↓	Ali→	Ove→	mIoU↑	FID↓	Ali→	Ove→	mIoU↑
Un-Gen	LayoutTransformer	24.32	0.037	0.542	0.587	30.05	0.067	0.005	0.359
	LayoutFormer++	20.20	<u>0.051</u>	0.546	**0.634**	47.08	0.228	<u>0.001</u>	0.401
	LayoutDM*	4.43	0.143	0.584	0.582	36.85	0.180	0.132	0.382
	DLT*	13.02	0.271	0.571	0.566	12.70	0.117	0.036	**0.431**
	LayoutDiffusion	<u>2.49</u>	**0.069**	<u>0.502</u>	<u>0.620</u>	**8.63**	<u>0.065</u>	**0.003**	0.417
	LayoutFlow (ours)	**2.37**	0.150	**0.498**	0.570	<u>8.87</u>	**0.057**	0.009	<u>0.424</u>
Gen-Type	NDN-none	13.76	0.560	0.550	<u>0.350</u>	35.67	0.350	0.170	0.310
	LayoutFormer++	2.48	<u>0.124</u>	0.537	**0.377**	10.15	**0.025**	<u>0.009</u>	0.333
	LayoutDM*	2.39	0.222	0.598	0.341	39.12	0.267	0.139	0.348
	DLT*	6.64	0.303	0.616	0.326	7.09	0.097	0.040	<u>0.349</u>
	LayoutDiffusion	<u>1.56</u>	**0.124**	**0.491**	0.345	<u>3.73</u>	<u>0.029</u>	**0.005**	0.343
	LayoutFlow (ours)	**1.48**	0.176	<u>0.517</u>	0.322	**3.66**	0.037	0.011	**0.350**
Gen-TypeSize	LayoutDM*	<u>1.76</u>	**0.175**	<u>0.606</u>	0.424	29.91	0.246	0.160	<u>0.436</u>
	DLT*	6.27	0.332	0.609	0.424	<u>5.35</u>	<u>0.130</u>	<u>0.053</u>	0.426
	LayoutFlow (ours)	**1.03**	<u>0.283</u>	**0.523**	**0.470**	**1.26**	**0.041**	**0.031**	**0.454**
	Validation Data	2.10	0.093	0.466	0.658	8.10	0.022	0.003	0.434

LayoutFlow to the non-diffusion-based models LayoutTransformer [14], Layout-Former++ [23], NDN-none [27] and RUITE [38].

Tasks. We test LayoutFlow and existing approaches on various common unconditional and conditional layout generation tasks. *Un-Gen* describes the layout generation task without any constraints. For the conditional scenario, we consider *Gen-Type* as assuming the element types as given, while *Gen-TypeSize* assumes element types and sizes are known. Lastly, we consider the *Completion* task, which takes in a partial layout that is missing up to 20% of its elements, and the *Refinement* task following [48] with a standard deviation of 0.01.

5.2 Evaluation

Quantitative Evaluation. In Tab. 1, we report the results of our experiments comparing LayoutFlow with existing approaches for different layout generation tasks. In terms of *FID*, our model outperforms state-of-the-art methods across all three tasks, except for a close second place on the PubLayNet *Un-Gen* task, proving its strong capabilities to model the underlying sample distribution. In particular, the *FID* score of 2.37 produced by LayoutFlow on the unconditional generation for RICO almost matches the score obtained when comparing the validation with the test dataset. Note that for the conditional tasks, the *FID* score can become even lower than the validation dataset since the generated

Table 2: Quantitative results for the completion and refinement tasks on the RICO and PubLayNet datasets. The two best results are highlighted in **bold** and <u>underlined</u>. Models marked with * have been retrained.

Task	Model	RICO				PubLayNet			
		FID↓	Ali→	Ove→	mIoU↑	FID↓	Ali→	Ove→	mIoU↑
Completion	LayoutDM*	6.80	**0.054**	0.630	0.678	25.02	0.169	0.107	0.678
	LayoutFlow (ours)	**1.51**	0.150	**0.474**	**0.741**	**1.10**	**0.054**	**0.127**	**0.746**
Refinement	RUITE	7.93	0.177	0.492	0.658	7.89	0.073	0.038	0.637
	LayoutFormer++	3.67	<u>0.141</u>	0.503	0.656	2.94	0.042	0.013	0.642
	LayoutDM*	2.91	0.143	0.575	0.437	48.61	0.286	0.173	0.372
	LayoutDiffusion	**0.55**	**0.102**	**0.469**	**0.719**	<u>2.05</u>	<u>0.035</u>	<u>0.008</u>	<u>0.660</u>
	LayoutFlow (ours)	<u>0.77</u>	0.152	<u>0.455</u>	<u>0.700</u>	**0.18**	**0.020**	**0.005**	**0.723**
	Validation Data	2.10	0.093	0.466	0.658	8.10	0.022	0.003	0.434

images share more similarities through conditioning. Regarding the geometrical metrics *Alignment*, *Overlap*, and *mIoU*, our method consistently outperforms LayoutDM and DLT while usually producing values close to LayoutDiffusion, a significantly larger model with 85M parameters compared to around 15M used by LayoutFlow. A comparison between the quality and the sampling speed is illustrated in Fig. 5, clearly showing that our method closes the gap between inference speed and performance. While non-diffusion models outperform diffusion models on some geometrical metrics, the drastically higher *FID* score implies that the generated results lack diversity.

On the completion task, shown in Tab. 2, LayoutFlow is able to clearly outperform LayoutDM, the only diffusion model that can handle that task, except for falling behind on *Alignment* for RICO. On the refinement task, LayoutFlow significantly improves the noisy layouts on PubLayNet, while being the second-best model for RICO, underlining its broad capabilities.

Qualitative Evaluation. We provide some qualitative examples in Fig. 6 for the unconditional Fig. 7 as well as for the conditional generation task on the RICO dataset. More samples for the other tasks and PubLayNet can be found in the supplementary material. Overall, LayoutFlow shows a strong performance across all tasks, producing visually pleasing results that resemble real layouts.

Limitations. We discuss the limitations of our proposed methods in the supplementary material.

5.3   Ablation Study

In order to justify our design choices, we perform an ablation study on unconditional layout generation using the RICO dataset.

Diffusion vs. Flow. We show that employing Flow Matching as the generative model is essential by also training the model used for LayoutFlow and only substituting Flow Matching with diffusion. We test two methods for sampling
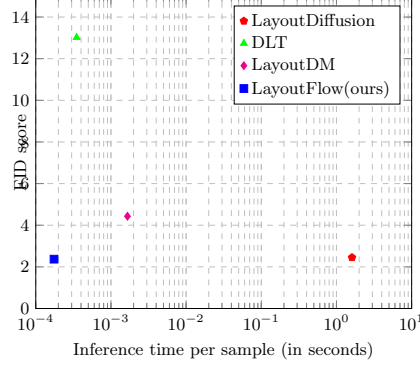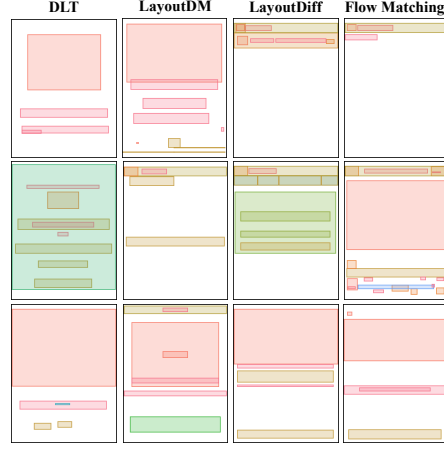
Fig. 5: Quality-Speed Comparison

Fig. 6: Examples for Un-Gen

(a) Examples for Gen-Type
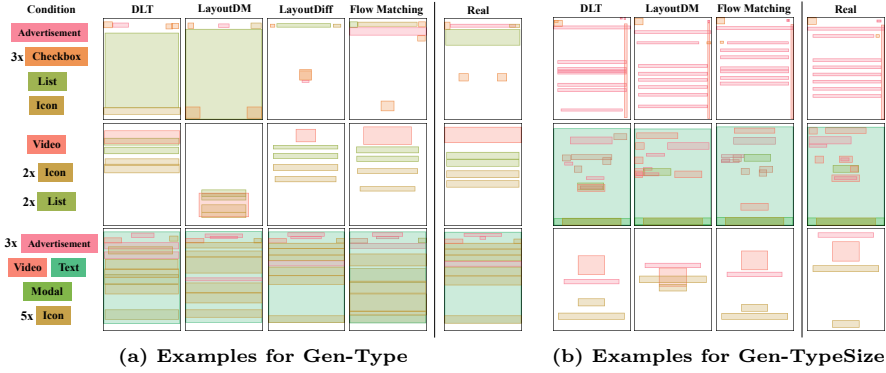

(b) Examples for Gen-TypeSize

Fig. 7: Conditional Generation Examples

from the diffusion-based model, i.e., DDPM and DDIM. As shown in Tab. 3, neither approach is able to reach the same performance as LayoutFlow, and both seem to be particularly lacking in terms of *Overlap*. Interestingly, DDIM outperforms DDPM by a large margin, which might be related to DDIM sampling based on an ODE instead of an SDE.

Regularization Loss. We validate the efficacy of our regularization loss proposed in Eq. (6) and the choice of λ in Tab. 4. It can be clearly observed that a stronger regularization using the L1-loss provides better alignment but increases the *FID*. Therefore, we considered the trade-off across all metrics and chose $\lambda = 0.2$ for LayoutFlow.

Initial Distribution. Since flow-based models allow for arbitrary initial distributions, we explore two alternatives. First, we try a uniform distribution to ensure all initial samples are placed on the canvas. In addition, as Analog

Table 3: Ablation study on the choice of training approaches using the same architecture

Approach	FID↓	Ali→	Ove→	mIoU↑
Diff.(DDPM)	34.89	**0.128**	0.335	0.116
Diff.(DDIM)	3.18	0.181	0.561	0.548
Flow	**2.37**	0.150	**0.498**	**0.570**

Table 4: Ablation study on different loss functions

Loss	FID↓	Ali→	Ove→	mIoU↑
$\mathcal{L}_{\mathrm{CFM}}$	2.27	0.194	0.507	0.570
$\mathcal{L}_{\mathrm{CFM}} + 0.1\mathcal{L}_1$	**2.20**	0.155	**0.495**	0.559
$\mathcal{L}_{\mathrm{CFM}} + 0.2\mathcal{L}_1$	2.37	0.150	0.498	0.570
$\mathcal{L}_{\mathrm{CFM}} + 0.3\mathcal{L}_1$	2.40	0.144	0.498	0.581
$\mathcal{L}_{\mathrm{CFM}} + 0.4\mathcal{L}_1$	2.60	**0.134**	0.498	0.579
L_1-loss only	26.64	0.159	0.625	**0.586**

Table 5: Ablation study on different initial distributions

Distribution	FID↓	Ali→	Ove→	mIoU↑
Gaussian	**2.37**	0.150	0.498	0.570
Uniform	2.61	**0.115**	**0.481**	**0.584**
Mixture	2.52	0.160	0.494	0.562

Table 6: Ablation study on different training trajectories

Trajectory	FID↓	Ali→	Ove→	mIoU↑
Linear	2.37	**0.150**	0.498	**0.570**
Sine/Cosine	**2.33**	0.172	0.533	0.557
Sine	2.48	0.152	**0.480**	0.565

Bits have a smaller sample space, we tested using a Gaussian for the geometrical elements and a uniform distribution for the Analog Bits. Altogether, the results in Tab. 5 indicate that a Gaussian initial distribution provides the best overall performance measured by the *FID* score, whereas a uniform distribution provides stronger results in terms of geometrical metrics, such as *Alignment* and *Overlap*.

Training Trajectories. As long as the conditional vector field fulfills the criteria presented in Sec. 3, it can be freely chosen for training. In addition to the linear training trajectory proposed in [32,44], we also explore the sine/cosine interpolation introduced in [1] and a sine-based interpolation, as shown in Tab. 6. The choice of training trajectories only slightly affects the performance.

6 Conclusion

In this paper, we explored the application of Flow Matching for layout generation and, as a result, proposed LayoutFlow, a flow-based model that is able to handle various layout generation tasks. Our model is able to significantly speed up inference compared to other diffusion-based models while providing state-of-the-art performance. Even though we tested a diverse set of design options, there still remains room for further exploration into applying Flow Matching to layout generation, for example, different training trajectories, conditioning methods, or initial distributions. Furthermore, LayoutFlow might also be extended to generate layouts considering the content in the elements, similar to related work [19, 46]. Overall, we demonstrate that Flow Matching provides a highly flexible and powerful tool for layout generation that offers a natural geometrical interpretation.

Acknowledgments

References

1. Albergo, M.S., Vanden-Eijnden, E.: Building normalizing flows with stochastic interpolants. In: ICLR (2023)
2. Arroyo, D.M., Postels, J., Tombari, F.: Variational transformer networks for layout generation. In: CVPR (2021)
3. Austin, J., Johnson, D.D., Ho, J., Tarlow, D., Van Den Berg, R.: Structured denoising diffusion models in discrete state-spaces. In: NeurIPS (2021)
4. Chai, S., Zhuang, L., Yan, F.: Layoutdm: Transformer-based diffusion model for layout generation. In: CVPR (2023)
5. Chen, J., Zhang, R., Zhou, Y., Chen, C.: Towards aligned layout generation via diffusion model with aesthetic constraints. In: ICLR (2024)
6. Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. In: NeurIPS (2018)
7. Chen, T., ZHANG, R., Hinton, G.: Analog bits: Generating discrete data using diffusion models with self-conditioning. In: ICLR (2023)
8. Cheng, C.Y., Huang, F., Li, G., Li, Y.: Play: Parametrically conditioned layout generation using latent diffusion. In: ICML (2023)
9. Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., Kumar, R.: Rico: A mobile app dataset for building data-driven design applications. In: UIST (2017)
10. Euler, L.: Institutiones calculi integralis, vol. 4. Academia Imperialis Scientiarum (1794)
11. Feng, W., Zhu, W., Fu, T.J., Jampani, V., Akula, A.R., He, X., Basu, S., Wang, X.E., Wang, W.Y.: LayoutGPT: Compositional visual planning and generation with large language models. In: NeurIPS (2023)
12. Gu, S., Chen, D., Bao, J., Wen, F., Zhang, B., Chen, D., Yuan, L., Guo, B.: Vector quantized diffusion model for text-to-image synthesis. In: CVPR (2022)
13. Guo, S., Jin, Z., Sun, F., Li, J., Li, Z., Shi, Y., Cao, N.: Vinci: an intelligent graphic design system for generating advertising posters. In: CHI (2021)
14. Gupta, K., Lazarow, J., Achille, A., Davis, L.S., Mahadevan, V., Shrivastava, A.: Layouttransformer: Layout generation and completion with self-attention. In: ICCV (2021)
15. He, L., Lu, Y., Corring, J., Florencio, D., Zhang, C.: Diffusion-based document layout generation. In: ICDAR (2023)
16. 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: NeurIPS (2017)
17. 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: NeurIPS (2017)
18. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. In: NeurIPS (2020)
19. Hsu, H.Y., He, X., Peng, Y., Kong, H., Zhang, Q.: Posterlayout: A new benchmark and approach for content-aware visual-textual presentation layout. In: CVPR (2023)

20. Hui, M., Zhang, Z., Zhang, X., Xie, W., Wang, Y., Lu, Y.: Unifying layout generation with a decoupled diffusion model. In: CVPR (2023)
21. Hyvärinen, A., Dayan, P.: Estimation of non-normalized statistical models by score matching. JMLR **6**(4) (2005)
22. Inoue, N., Kikuchi, K., Simo-Serra, E., Otani, M., Yamaguchi, K.: Layoutdm: Discrete diffusion model for controllable layout generation. In: CVPR (2023)
23. Jiang, Z., Guo, J., Sun, S., Deng, H., Wu, Z., Mijovic, V., Yang, Z.J., Lou, J.G., Zhang, D.: Layoutformer++: Conditional graphic layout generation via constraint serialization and decoding space restriction. In: CVPR (2023)
24. Jyothi, A.A., Durand, T., He, J., Sigal, L., Mori, G.: Layoutvae: Stochastic scene layout generation from a label set. In: ICCV (2019)
25. Kikuchi, K., Simo-Serra, E., Otani, M., Yamaguchi, K.: Constrained graphic layout generation via latent optimization. In: ACM MM (2021)
26. Kobyzev, I., Prince, S.J., Brubaker, M.A.: Normalizing flows: An introduction and review of current methods. IEEE TPAMI **43**(11) (2020)
27. Lee, H.Y., Jiang, L., Essa, I., Le, P.B., Gong, H., Yang, M.H., Yang, W.: Neural design network: Graphic layout generation with constraints. In: ECCV (2020)
28. Levi, E., Brosh, E., Mykhailych, M., Perez, M.: Dlt: Conditioned layout generation with joint discrete-continuous diffusion layout transformer. In: ICCV (2023)
29. Li, J., Yang, J., Hertzmann, A., Zhang, J., Xu, T.: Layoutgan: Generating graphic layouts with wireframe discriminators. In: ICLR (2018)
30. Li, J., Yang, J., Zhang, J., Liu, C., Wang, C., Xu, T.: Attribute-conditioned layout gan for automatic graphic design. IEEE TVCG **27**(10) (2020)
31. Lin, J., Guo, J., Sun, S., Yang, Z.J., Lou, J.G., Zhang, D.: Layoutprompter: Awaken the design ability of large language models. In: NeurIPS (2023)
32. Lipman, Y., Chen, R.T.Q., Ben-Hamu, H., Nickel, M., Le, M.: Flow matching for generative modeling. In: ICLR (2023)
33. Liu, X., Gong, C., et al.: Flow straight and fast: Learning to generate and transfer data with rectified flow. In: ICLR (2022)
34. Ma, N., Goldstein, M., Albergo, M.S., Boffi, N.M., Vanden-Eijnden, E., Xie, S.: Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. arXiv preprint arXiv:2401.08740 (2024)
35. Neklyudov, K., Severo, D., Makhzani, A.: Action matching: A variational method for learning stochastic dynamics from samples. In: ICML (2023)
36. O'Donovan, P., Agarwala, A., Hertzmann, A.: Designscape: Design with interactive layout suggestions. In: CHI (2015)
37. O'Donovan, P., Agarwala, A., Hertzmann, A.: Learning layouts for single-pagegraphic designs. IEEE TVCG **20**(8) (2014)
38. Rahman, S., Sermuga Pandian, V.P., Jarke, M.: Ruite: Refining ui layout aesthetics using transformer encoder. In: 26th International Conference on Intelligent User Interfaces-Companion (2021)
39. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: CVPR (2022)
40. Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S.: Deep unsupervised learning using nonequilibrium thermodynamics. In: ICML (2015)
41. Song, J., Meng, C., Ermon, S.: Denoising diffusion implicit models. In: ICLR (2020)
42. Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B.: Score-based generative modeling through stochastic differential equations. In: ICLR (2021)
43. Tang, Z., Wu, C., Li, J., Duan, N.: LayoutNUWA: Revealing the hidden layout expertise of large language models. In: ICLR (2024)

44. Tong, A., Malkin, N., Huguet, G., Zhang, Y., Rector-Brooks, J., Fatras, K., Wolf, G., Bengio, Y.: Improving and generalizing flow-based generative models with mini-batch optimal transport. In: ICMLW (2023)
45. Xie, Y., Huang, D., Wang, J., Lin, C.Y.: Canvasemb: Learning layout representation with large-scale pre-training for graphic design. In: ACM MM (2021)
46. Xu, C., Zhou, M., Ge, T., Jiang, Y., Xu, W.: Unsupervised domain adaption with pixel-level discriminator for image-aware layout generation. In: CVPR (2023)
47. Yamaguchi, K.: Canvasvae: Learning to generate vector graphic documents. In: ICCV (2021)
48. Zhang, J., Guo, J., Sun, S., Lou, J.G., Zhang, D.: Layoutdiffusion: Improving graphic layout generation by discrete diffusion probabilistic models. In: ICCV (2023)
49. Zhong, X., Tang, J., Yepes, A.J.: Publaynet: largest dataset ever for document layout analysis. In: ICDAR (2019)