LayoutFlow: Flow Matching for Layout Generation

Supplementary Material

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

In this document, we provide additional explanations and evaluations to supplement the contents of the main paper. Specifically, we present further discussions on training trajectories, conditioning, and the limitations of our method, as well as additional quantitative and qualitative results.

A Training Trajectories

The conditional flow, or training trajectory, $\phi_t(\mathbf{x}) = \mathbf{x}_t$ can be chosen freely as long as the conditional vector field follows the continuity equation and $\phi_0(\mathbf{x}) = \mathbf{x}_0$ and $\phi_1(\mathbf{x}) = \mathbf{x}_1$. While LayoutFlow is trained using a linear trajectory following [8,9], we also tested the sine/cosine interpolation introduced in [1] and a sine-based interpolation. More details are shown in Tab. 1. Our main motivation for the sine-based interpolation was to preserve the direction of the conditional vector field but apply a different time schedule to the trajectory. In the linear scenario, the conditional vector field remains independent of the time and always points towards x_1. On the other hand, the sine/cosine trajectory changes its direction over time, only gradually leading to x_1 in a circular trajectory. The sine-based interpolation maintains the direction of the linear trajectory but weighs it depending on the time. At an early time of the trajectory, the magnitude of the direction is large, whereas towards the end, the derivative starts to slow down. In analogy to time importance sampling in diffusion models [7], we hypothesized that this might help focus on the later stages of the flow process during training, where more detailed features become important. However, based on the paper's ablation results, the cosine-based interpolation does not seem to affect the performance. Nonetheless, this scheduling property of conditional fields might be further explored with other functions of the general form:

$$v_t = \kappa(t)(\mathbf{x}_1 - \mathbf{x}_0), \tag{1}$$

Table 1: Overview of different training trajectories and conditional vector fields. Note that the conditional vector field is the time-derivative of the training trajectory.

Name	Training Trajectory $\mathbf{x}_t$	Conditional Vector Field v_t
Linear	$(1-t)\mathbf{x}_0 + t\mathbf{x}_1$	$\mathbf{x}_1 - \mathbf{x}_0$
Sine/Cosine	$\cos(\frac{\pi}{2}t)\mathbf{x}_0 + \sin(\frac{\pi}{2}t)\mathbf{x}_1$	$\cos(\frac{\pi}{2}t)\mathbf{x}_1 - \sin(\frac{\pi}{2}t)\mathbf{x}_0$
Sine	$(1-\sin(\frac{\pi}{2}t))\mathbf{x}_0 + \sin(\frac{\pi}{2}t)\mathbf{x}_1$	$\cos(\frac{\pi}{2}t)(\mathbf{x}_1 - \mathbf{x}_0)$

while still fulfilling the conditions mentioned above. In the cosine case of $\kappa(t) = \cos(\frac{pi}{2}t)$, the schedule might have been too close to the linear schedule to make a noticeable change, and different choices might provide better results.

B Inference Speed Analysis

Since previous layout generation approaches rely on different architectures, we provide some insights on how the architectural components, *i.e.*, the model size measured by the numbers of trainable parameters and the number of tokens needed to represent a single layout, influence the inference speed in Table 2. In addition, we also report the number of steps each method requires to generate a layout that is tied to the choice of the generative model. Overall, it can be seen that the speed-up for LayoutFlow can be attributed to architectural improvements as well as changing from a diffusion model to a flow-base model due to the reduced number of generation steps required.

Table 2: Overview of different factors contributing to inference speed

	Model Size	Token per Layout	Generation Steps	Inference Speed
LayoutDiffusion	86M	>100	160	1600.00ms
LayoutDM	12.4M	100	100	16.60ms
DLT	9.0M	20	100	3.50ms
LayoutFlow (ours)	12.7M	20	50	**1.75ms**

C Quality Speed Trade-off

In Fig. 1, we compare the number of function evaluations of our model, corresponding to the number of steps taken to solve the ODE, to the performance on the RICO dataset. While only 5 function evaluations result in a bad score of an *FID* of roughly 50, just three more evaluations can bring down the *FID* to 13. The improvement observed by increasing the number of steps plateaus at

around 40. In comparison, diffusion models typically require at least 100 steps to perform well. Overall, LayoutFlow offers a strong quality-speed trade-off, where slightly reducing the number of steps, *e.g.*, from 30 to 20, only has a small impact on performance.

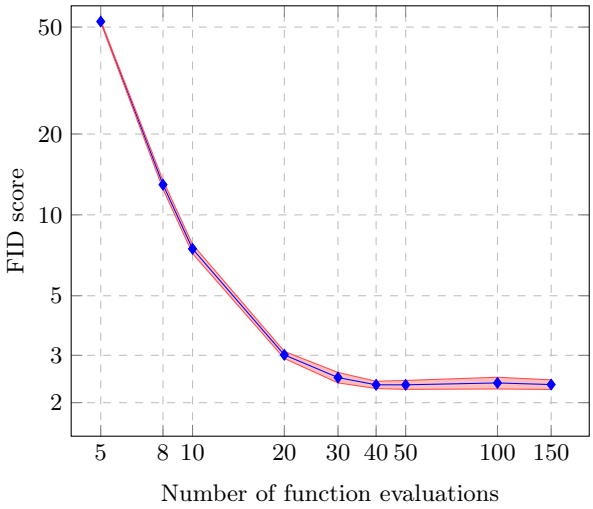

Fig. 1: Quality-Speed Trade-Off. We investigate the relationship between the step size employed by the Euler method, which translates to the number of function evaluations, and the *FID* score using LayoutFlow trained on the RICO dataset.

D Conditioning Analysis

There are several ways to introduce conditions to a generative task. For layout generation, given conditions can differ significantly depending on the task. For LayoutFlow, we employ a masking strategy during training that indicates which part of the input serves as a condition, similar to [3, 4, 7]. Intuitively, the masking mechanism can be considered a switch that allows training multiple flow-based task-specific models with a single shared architecture in just one model. Alternatively, it is possible only to train an unconditional model and impose the conditions during sampling through a guidance strategy as done in [5, 10]. This strategy only requires training the model on the unconditional task and inserting the condition during inference, effectively altering the trajectory.

To condition the trajectory during inference, we employ a mask $\mathbf{m}$ that is zero for the condition dimension, *i.e.*, all ones in the case of unconditional generation. Essentially, we use the mask only to update the direction of the condition dimension to $\mathbf{u}'$ and can be described as

Table 3: Quantitative comparison of training LayoutFlow on multiple tasks using a masking strategy and inserting conditions to an unconditional LayoutFlow model by modifying the trajectory during inference. The best results are highlighted in **bold**.

		RICO			
Task	Condition	FID↓	Ali→	Ove→	mIoU↑
Un-Gen	None	**2.28**	0.155	0.511	**0.610**
	Mask	2.37	**0.150**	**0.498**	0.570
Gen-Type	Trajectory	33.38	0.202	0.635	0.160
	Mask	**1.48**	**0.176**	**0.517**	**0.322**
Gen-TypeSize	Trajectory	174.66	0.862	0.728	0.181
	Mask	**1.03**	**0.283**	**0.523**	**0.470**
Completion	Trajectory	7.96	0.193	0.520	0.657
	Mask	**1.51**	**0.150**	**0.474**	**0.741**
Validation Dataset		2.10	0.093	0.466	0.658

$$\mathbf{u'}_{\frac{k+1}{T}} = \mathbf{m} \odot \mathbf{u}_\theta(\mathbf{x}_{\frac{k}{T}} + \frac{1}{T}\mathbf{u'}_{\frac{k}{T}}) + (1 - \mathbf{m}) \odot (\mathbf{x}_c - \mathbf{x}_{\frac{k}{T}}),$$

where $\mathbf{x}_c$ denotes the conditions and $\odot$ represents the Hadamard product. For the conditional task, we update the direction only while $k < 0.8T$ as we find it to work better empirically. Table 3 compares both conditioning methods and illustrates that a training-based approach significantly outperforms conditioning during inference. While we tried several other methods to introduce the conditions during inference, there might be more sophisticated conditioning methods that perform better and might be a potential future research direction.

E Results Using Different Data Split

Due to the different data splits used by previous methods, comparing the numbers reported by other papers has become difficult. The FID values depend on a separately trained network, making comparing even more difficult as the same weights and network are required. Our experiments used the same settings as in [6, 10]. As a result, we had to retrain LayoutDM [5] and DLT [7] with that specific data split. To ensure that LayoutFlow works robustly across different dataset settings, we also trained a model on the data split used in LayoutDM [2], which includes up to 25 elements per layout as opposed to only 20 elements in the other setting. Additionally, this allows us to compare with a concurrent work called LACE [3], which uses a continuous diffusion model.

Table 4: Quantitative results on a different dataset split [5] for the PubLayNet dataset. The best result is highlighted in **bold**. The → symbol indicates best results are the ones closest to the validation data.

Task	Model	PubLayNet FID↓	Ali→	Ove→	mIoU↑
Un-Gen	LACE	8.45	0.141	0.075	-
	LayoutDM	13.90	0.195	0.134	-
	LayoutFlow (ours)	**7.48**	**0.066**	**0.015**	0.423
Gen-Type	LACE	5.14	0.046	0.018	**0.383**
	LayoutDM	7.95	0.106	0.164	0.310
	LayoutFlow (ours)	**3.58**	**0.046**	**0.018**	0.349
Gen-TypeSize	LACE	2.66	0.061	**0.034**	0.418
	LayoutDM	4.25	0.119	0.189	0.381
	LayoutFlow (ours)	**0.80**	**0.052**	0.036	**0.443**
Validation Data		6.25	0.021	0.003	0.438

Table 5: Quantitative results for the completion task with different percentages of the missing elements on the RICO and PubLayNet datasets. The best results are highlighted in **bold**. Models marked with * have been retrained.

Task	Model	RICO FID↓	Ali→	Ove→	mIoU↑	PubLayNet FID↓	Ali→	Ove→	mIoU↑
Completion (20%)	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**
Completion (80%)	LayoutDM*	5.21	**0.094**	0.658	0.574	28.73	0.223	0.146	0.385
	LayoutFlow (ours)	**3.59**	0.182	**0.605**	**0.628**	**4.02**	**0.050**	**0.024**	**0.445**
Validation Dataset		2.10	0.093	0.466	0.658	8.10	0.022	0.003	0.434

F Completion Task

The completion task describes generating a complete layout given an incomplete layout. In this scenario, the given incomplete layout acts as a strong condition. There are different ways to define the completion task, *e.g.*, Inoue *et al.* assume that up to 20% of the layout is already given. While this scenario is closer to unconditional generation as it still offers a high degree of flexibility, we also looked into completing almost finished layouts containing more than 80% of their elements. To show that LayoutFlow can handle either scenario, we seperately trained models with the respective conditional mask and present the results in Tab. 5 compared to LayoutDM and show the qualitative results in Sec. H.

G Limitations and Broader Impact

While our work aims to improve the state of layout generation further and help make design tasks easier, there are also some potential negative effects. Easier and better layout generation might be misused to build spam websites or large amounts of content or potentially be used for scams, particularly phishing attempts. The potential harm does not come from enabling such actions but derives from the scale that automation allows. Nonetheless, we believe the democratization of technologies like ours outweighs these risks.

Our proposed model, LayoutFlow, has shown remarkable results. However, some areas can still be improved in the future. Alignment remains one of the largest challenges, especially for models working in the continuous data space. While LayoutFlow utilizes an L1 regularization loss to improve the alignment, the issue is not fully resolved. For the future, it is crucial to find a better loss function that can reflect the perceptual error between layouts rather than relying too heavily on metrics that do not necessarily correlate with important design aspects such as alignment.

H More Qualitative Results

Due to limited space in the main paper, we present more qualitative results for each task and dataset in this section.

References

1. Albergo, M.S., Vanden-Eijnden, E.: Building normalizing flows with stochastic interpolants. In: ICLR (2023)
2. Chai, S., Zhuang, L., Yan, F.: Layoutdm: Transformer-based diffusion model for layout generation. In: CVPR (2023)
3. Chen, J., Zhang, R., Zhou, Y., Chen, C.: Towards aligned layout generation via diffusion model with aesthetic constraints. In: ICLR (2024)
4. Hui, M., Zhang, Z., Zhang, X., Xie, W., Wang, Y., Lu, Y.: Unifying layout generation with a decoupled diffusion model. In: CVPR (2023)
5. Inoue, N., Kikuchi, K., Simo-Serra, E., Otani, M., Yamaguchi, K.: Layoutdm: Discrete diffusion model for controllable layout generation. In: CVPR (2023)
6. 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)
7. Levi, E., Brosh, E., Mykhailych, M., Perez, M.: Dlt: Conditioned layout generation with joint discrete-continuous diffusion layout transformer. In: ICCV (2023)
8. Lipman, Y., Chen, R.T.Q., Ben-Hamu, H., Nickel, M., Le, M.: Flow matching for generative modeling. In: ICLR (2023)

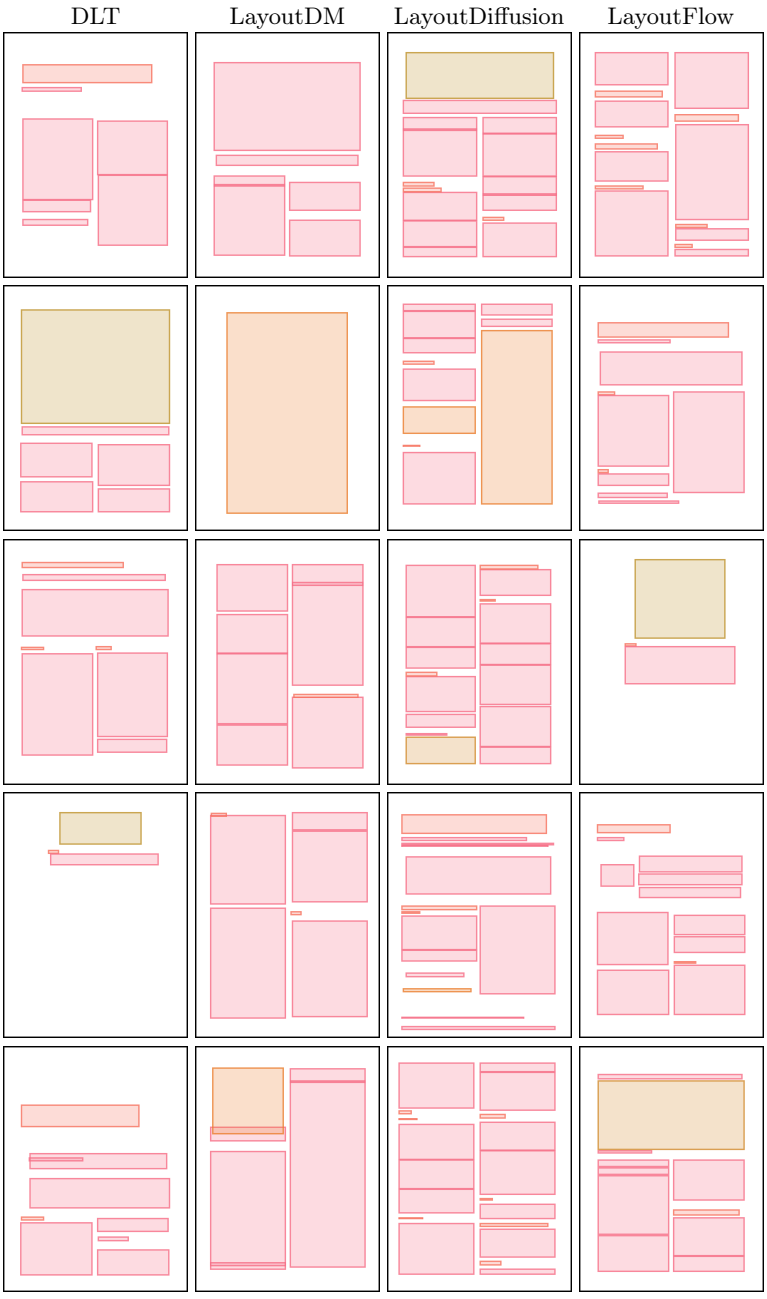

Table 6: Unconditional Generation on PubLayNet

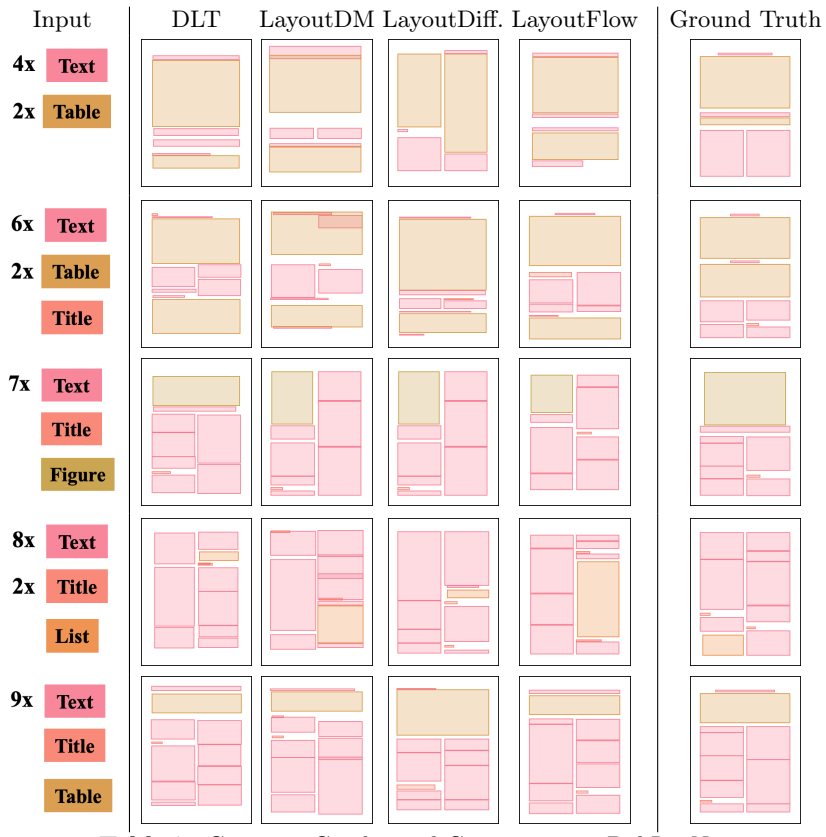

Table 7: Category Conditional Generation on PubLayNet

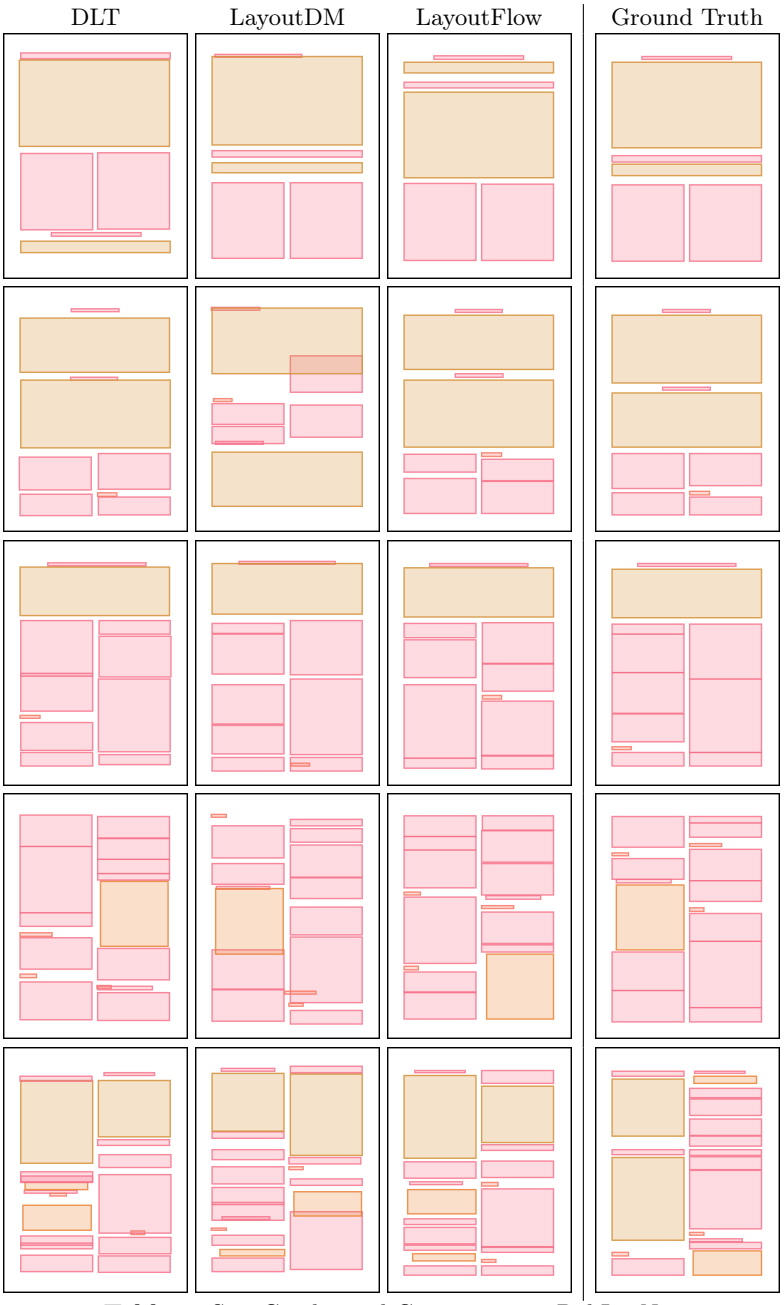

Table 8: Size Conditional Generation on PubLayNet

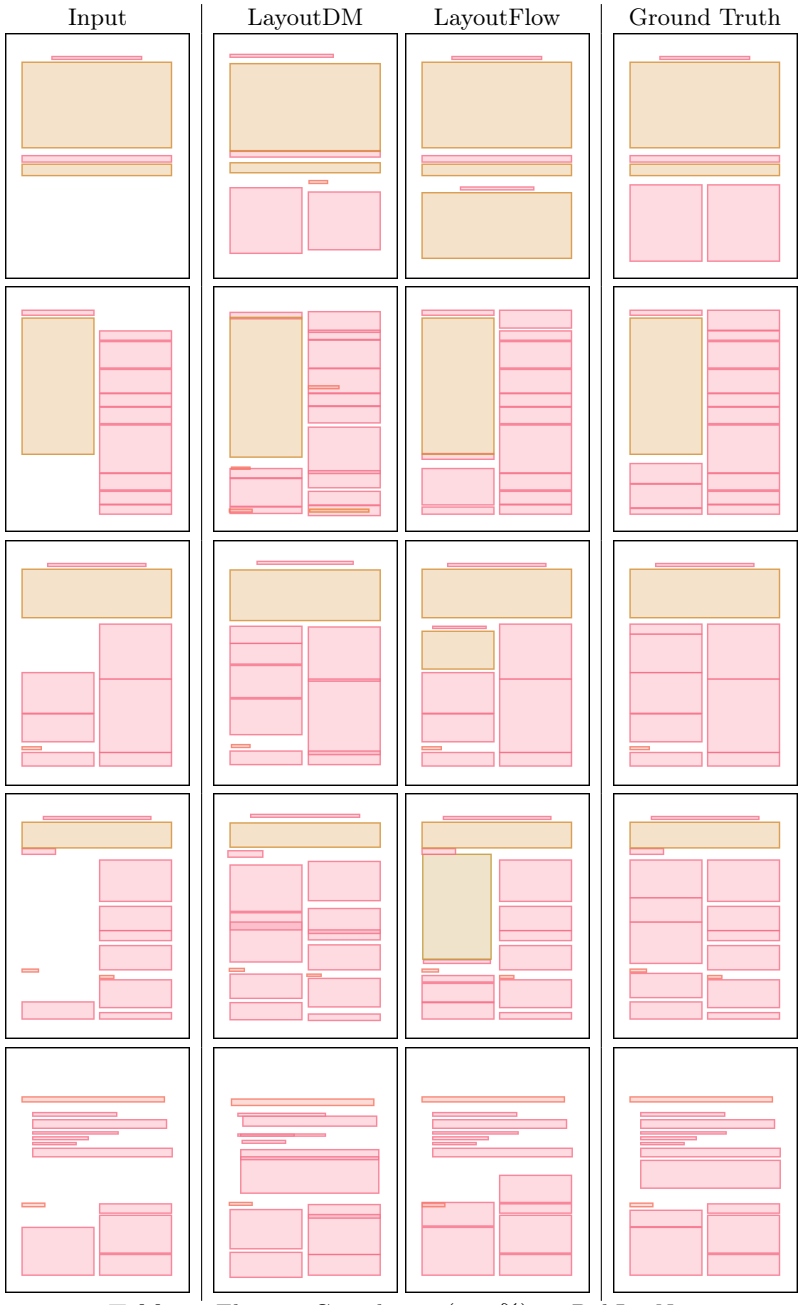

Table 9: Element Completion (0-20%) on PubLayNet

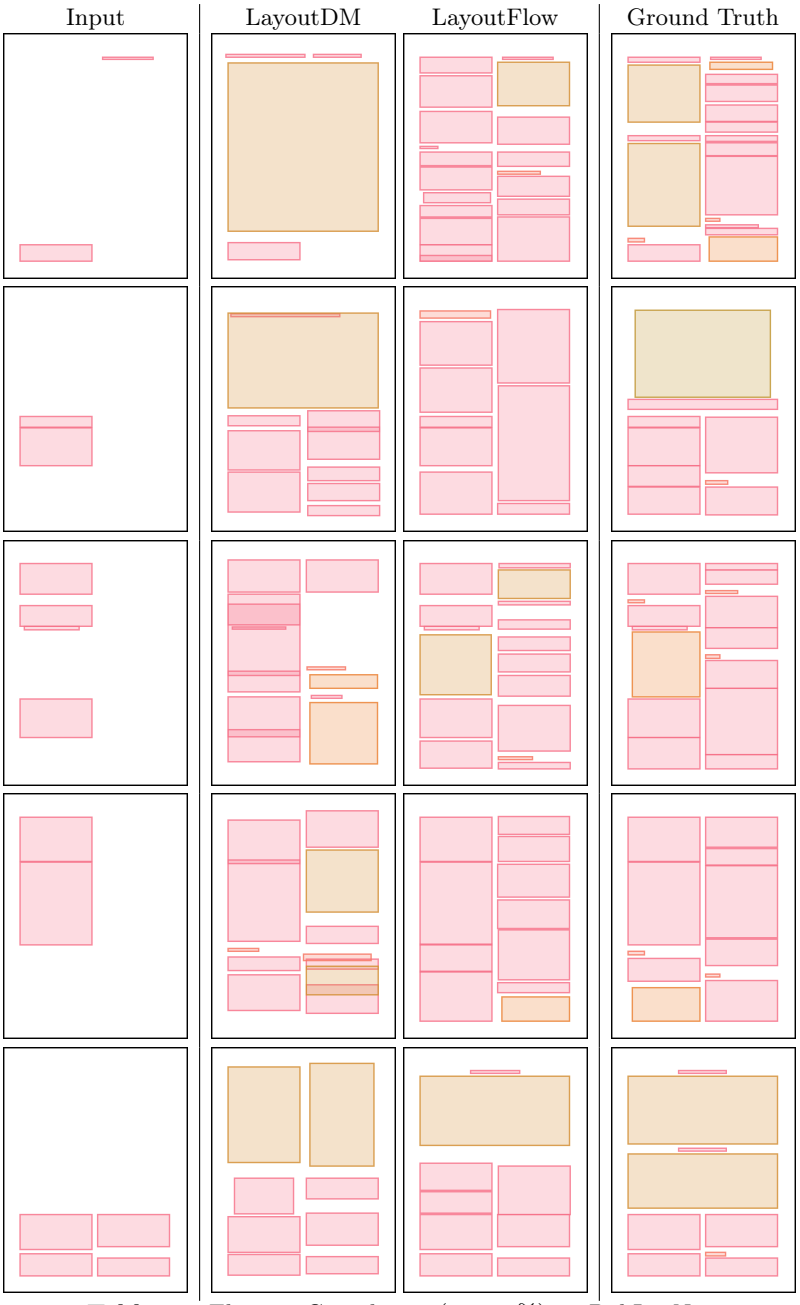

Table 10: Element Completion (80-100%) on PubLayNet

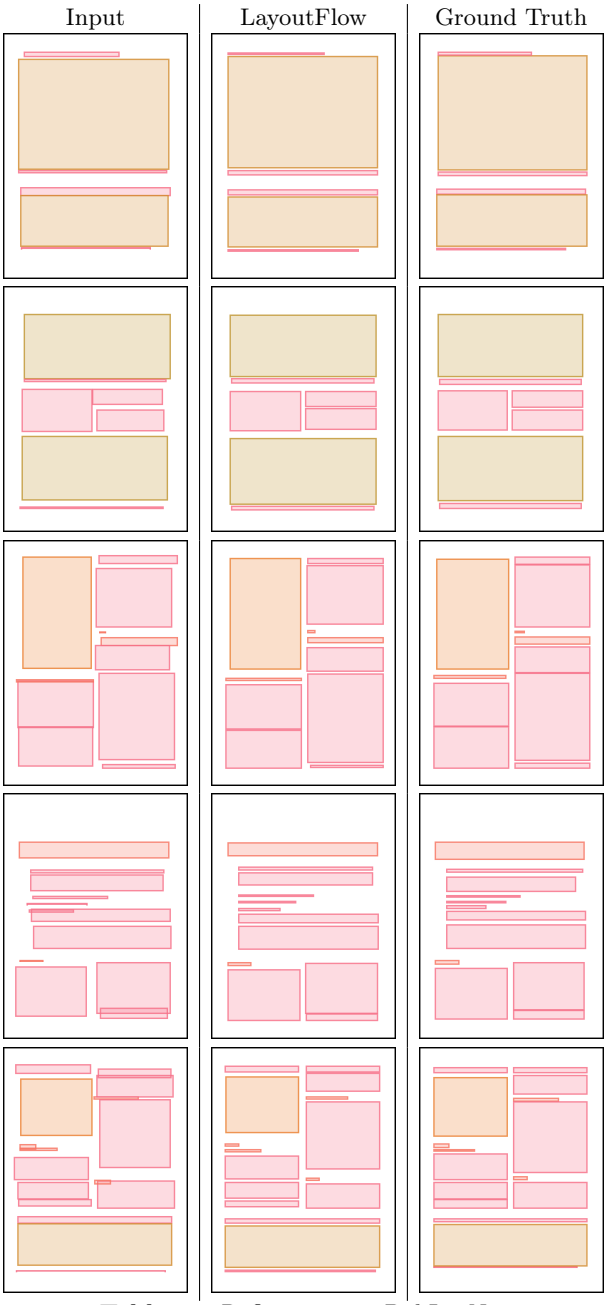

Table 11: Refinement on PubLayNet

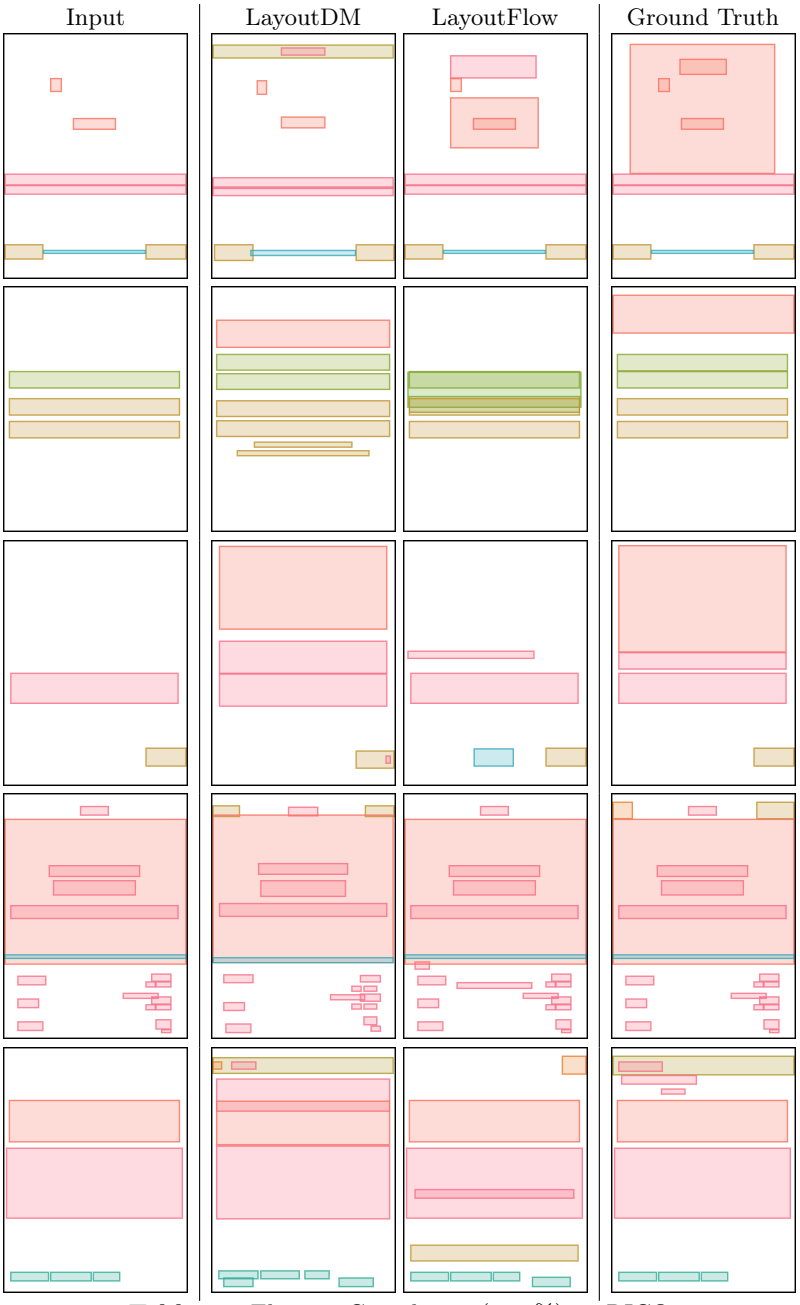

Table 12: Element Completion (0-20%) on RICO

Table 13: Element Completion (80-100%) on RICO

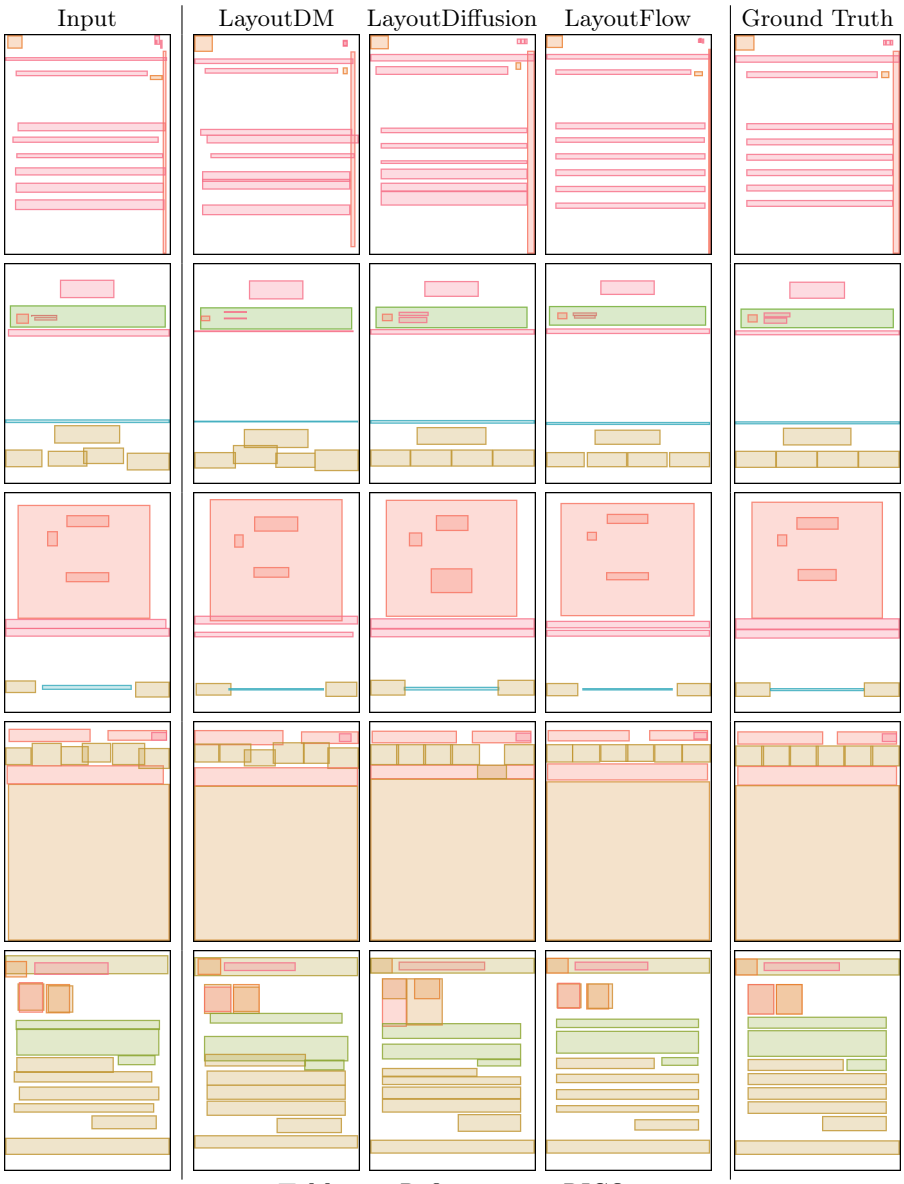

Table 14: Refinement on RICO

9. 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 minibatch optimal transport. In: ICMLW (2023)
10. Zhang, J., Guo, J., Sun, S., Lou, J.G., Zhang, D.: Layoutdiffusion: Improving graphic layout generation by discrete diffusion probabilistic models. In: ICCV (2023)